MONASH University

Distributed Associative Memory Approach for Cloud Computing Environments

Amir Hossein Basirat
B.Sc. in Electrical Engineering (IUT)
Master of Information Technology (Adelaide)
M.Sc. (Minor Thesis) in Computer Science (Monash)

A thesis submitted for the degree of
Doctor of Philosophy (0190)
at Monash University in 2016
Faculty of Information Technology

Copyright Notice

Abstract

One of the main challenges for large-scale computer clouds dealing with massive real-time data is in coping with the rate at which unprocessed data is being accumulated. “Big data” demands abound in scientific and engineering applications including biotechnology (e.g., characterisation using synchrotrons) and global monitoring of fixed and mobile assets in industry, transport and defence that entail massive real-time streams from and to stationary or mobile sensors and actuators. Their dynamic and distributed nature, and not least their exponential growth make real-time data management complicated, and storage, updates and analytics costly. With emerging interest to leverage massive amounts of data that are available in open sources, such as the Web for solving long-standing information retrieval problems, the question as how to effectively process immense datasets is becoming increasingly relevant. This raises the question of whether our capability to recognise and process such immense data copes with our ability to generate them.

This question will be addressed in this thesis by first examining the capability of existing large-scale data-processing schemes to scale up with this outgrowth of data. To address some of their highlighted limitations, particularly regarding computational complexity and scalability, this research proposes a novel associative-memory-based scheme for big data processing that is scalable, distributable and lightweight, and that overcomes some of the issues encountered in traditional data access mechanisms for data storage and retrieval. Thus, the primary aim of this thesis is to apply an access scheme that will enable fast data retrieval across multiple records and data segments associatively. This will result in a new type of database-like functionality that is capable of scaling up or down over the available infrastructure continuously and dynamically without degradation. Having a highly distributable computational framework that operates with simple processing elements and adapts to the conditions will provide a scalable framework for managing cloud data deluge. In this regard, associative memory concepts open a new pathway for accessing data in a highly distributed environment that will facilitate a parallel-distributed computational model to automatically adapt to the dynamic data environment for optimised performance.

To achieve the above goal, a distributed data access scheme that enables data storage and retrieval by association is first developed to circumvent the partitioning issue experienced within referential data access mechanisms. In our model, data records are treated as patterns. As a result, data storage and retrieval are performed using a distributed pattern recognition approach that is implemented through the integration of loosely coupled computational networks, followed by a divide-and-distribute approach that facilitates the distribution of these networks within the cloud dynamically. To date, all implementations of MapReduce, including the Hadoop version, have interpreted data in a relational model, which limits its functionality when dealing with complex and unstructured data such as images. To address this, an associative-memory-based MapReduce is introduced to elevate the MapReduce key-value scheme to a higher level of functionality by replacing the purely quantitative key-value pairs with scalable associative-memory-based data structures that will improve parallel processing of data with complex relations. By having an associative key-value model, we can deal with data in any form and in any representation simply by using a pattern-matching model that treats data records as patterns and provides a distributed data access scheme that enables data storage and retrieval by association, thereby circumventing the scaling issue experienced within referential data access mechanisms. The principle of associative-memory-based learning is implemented through the use of connected layers in a hierarchical fashion; with local feature learning happening at the lowest layer while features are combined to form higher representations at upper layers. While the proposed scheme is fundamentally different from published approaches in data management, it provides comparable performance benchmarks when tested against well-known large-scale data management schemes like Distributed MapReduce, Pregel and GraphLab. For this purpose, a comprehensive series of analyses have been performed on recognition accuracy and computational complexity using various types of patterns ranging from facial images to sensory readings. These analyses were conducted to validate the proposed scheme as final proof of concept by developing a suitable test environment to ensure the applicability of the model for real-life datasets.

In addition, this thesis investigates the extension of the proposed distributed data management scheme for different data-intensive scenarios by improving upon the existing cloud data management models for fault tolerance and scalability and reducing MapReduce communication overheads by introducing data locality. In particular, three data-intensive scenarios are considered in detail: dealing with large datasets, handling large training volumes and a neural network with an

excessive number of processing neurons. We also investigate a number of innovative cloud applications that benefit from data that are universally available within the network, benchmarking and validating the results to find the asymptotical limits of the technique through rigorous testing and simulation. Moreover, the application of our associative-memory-based approach is examined as a case study in a cloud of wireless sensor networks (Cloud-WSNs) to investigate the capabilities of the scheme in performing large-scale pattern recognition operations in resource-constrained WSNs, and extending the scheme applicability to various platform types, from coarse-grained computer clouds to fine-grained wireless sensor networks. The outcomes of this study indicate that our distributed parallel processing model is highly capable of processing Internet-scale data using lightweight associative-memory-based techniques where data recognition results are obtained in real-time using computationally inexpensive parallel operations within the body of the network.

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature: basirat

Print Name: Amirhossein Basirat

Date: 06/09/2016

This thesis is dedicated to my beloved wife, my lovely parents, my gorgeous daughter, and my supporting brother and sister, who have inspired and supported me in my pursuit of higher education.

This page intentionally left blank.

Acknowledgements

Primarily, my humble thanks to God, who is the most Beneficent and the most Merciful, for the endless help He has given me to complete this thesis.

My PhD has been a rewarding journey full of wonderful experiences that would not have been possible without the support and encouragement of many people. Now that my journey is near its end, I would like to take the opportunity to express my sincere thanks to all of the amazing people who have helped me along the way. First, I offer my profoundest gratitude to my supervisors, Dr Asad I. Khan and Professor Bala Srinivasan, who gave me the opportunity to pursue my studies in their group. I would like to thank them both for their continuous help, guidance, support and encouragement throughout all of the difficult and enjoyable moments of my PhD endeavour. Special thanks and deep appreciation go to Dr Asad I. Khan for all of his advice and support throughout the duration of my study. You have been an inspiration and guide to me. I acknowledge and highly value your expertise and experience.

My deep gratitude goes to my family for being there with me throughout this journey. In particular, I want to thank my precious wife, Fatima, for standing beside me, and for her endurance, support and unwavering love that will always be in my heart. These few words are not enough to express my deepest appreciation for her efforts during the past few years. My never-ending thanks and love are conveyed to my kind dad, my lovely mom, my supportive brother and my gorgeous sister for their continuous love and support, and for always being there for me through easy and difficult times. Finally, I would like to thank all of my friends and colleagues who helped to make this possible. It has been an incredible journey of self-discovery. Thank you all for making my dreams come true.

Amir Hossein Basirat

September 2016

This page intentionally left blank.

Contents

Copyright Notice ..... **ii**
Abstract ..... **iii**
Declaration ..... **vi**
Acknowledgements ..... **ix**
Contents ..... **x**
List of Tables ..... **xiv**
List of Algorithms ..... **xv**
List of Figures ..... **xvi**
List of Abbreviations ..... **xxi**
List of Publications ..... **xxiii**
Chapter 1: Introduction **1**
1.1 Recognition at Large-scale and Big Data 2
1.2 Cloud Computing and Large-scale Data Processing 4
1.3 Big Data, Feature Extraction and Pattern Recognition 5
1.4 Pattern Recognition for Large-scale Data Processing 6
1.4.1 Common Barriers 7
1.4.2 Possible Solutions 8
1.5 Motivation and Research Objectives 10
1.6 Hypotheses and Methodologies 12
1.7 Research Contributions 15
1.8 Thesis Outline 16
Chapter 2: Distributed Pattern Recognition and Data Management at Internet-scale **19**
2.1 Definition and Characteristics of Big Data 22
2.1.1 Data Volume 24
2.1.2 Data Velocity 24
2.1.3 Data Variety 25
2.2 Neural Network Schemes for Big Data Processing 26
2.2.1 Feed-forward Neural Network 27
2.2.2 Recurrent Neural Networks 28
2.2.3 Hopfield Network 29
2.2.4 Self-organising Maps 30
2.2.5 Support Vector Machine 31
2.3 Neural Network/Machine Learning Requirements for Large-scale Pattern Recognition and Data Processing 33
2.4 Parallel Data-processing Frameworks 35
2.4.1 Hadoop and Hadoop Distributed File Systems 35
2.4.1.1 HDFS Features 35
2.4.1.2 HDFS Architecture 36

2.4.2 MapReduce 38
2.4.3 Hadoop YARN 40
2.4.4 Apache Mahout 42
2.4.5 Google Pregel 44
2.4.6 GraphLab 46
2.4.6.1 GraphLab 1.0 46
2.4.6.2 GraphLab 2.2 (PowerGraph Abstraction) 48
2.5 Machine Learning and Pattern Recognition 49
2.5.1 Pre-processing 50
2.5.2 Feature Selection 51
2.5.3 Model Selection 51
2.5.4 Training, Testing and Optimisation 51
2.6 Distributed Approach for Large-scale Pattern Recognition and Data Processing 52
2.6.1 Learning Approach 53
2.6.2 Processing Approach 54
2.6.3 Training Approach 55
2.7 Graph Neuron for Scalable Recognition 55
2.7.1 Graph Neuron Architecture 56
2.7.1.1 Single-cycle Learning Approach 58
2.7.1.2 GN Pattern Crosstalk Problem 61
2.7.2 Hierarchical Graph Neuron 61
2.7.2.1 HGN Communication Approach 63
2.7.3 Distributed Hierarchical Graph Neuron 65
2.8 Conclusion 66

Chapter 3: Edge Detecting Hierarchical Graph Neuron 69
3.1 Associative Memory Concept for Pattern Recognition 70
3.2 Pre-processing and Dimensionality/Content Reduction 72
3.2.1 Structural Reduction 73
3.2.2 Content Reduction 74
3.2.2.1 Drop-fall Algorithm 75
3.3 EdgeHGN Computational Architecture 78
3.3.1 Two-stage Recognition Procedure 80
3.3.1.1 Sub-pattern Recognition Level 80
3.3.1.2 Pattern Reconstruction and Recognition Level 81
3.3.2 Bias Array Design 84
3.4 EdgeHGN Communication Framework 85
3.4.1 Network Generation 85
3.4.2 EdgeHGN Communications 86
3.4.2.1 EdgeHGN Macro-communications 87
3.4.2.2 EdgeHGN Micro-communications 88
3.5 EdgeHGN Algorithms and Functions 89
3.6 EdgeHGN Time Complexity and Scalability Analysis 92
3.6.1 Time Complexity 92
3.6.1.1 Recall Time Comparative Study 96
3.6.2 Scalability Analysis 100
3.6.2.1 Storage Capacity Analysis 101

3.6.2.2 Communication Complexity Analysis 103
3.7 Pattern Recognition Simulation and Results 106
3.7.1 Binary Character Pattern Recognition 106
3.7.2 Recognition Test on Binary Images 113
3.7.3 Recognition Test on Noisy Binary Images 119
3.7.3.1 Global Binary Signature Scheme for Colour Recognition 120
3.7.3.2 Sobel's Edge Recognition for Structural Information 120
3.7.3.3 Recognition Accuracy Analysis 121
3.7.4 Handwritten Object Recognition Test with Multiple Features 127
3.7.4.1 Classification Procedures 127
3.7.4.2 Recognition Analysis 129
3.8 Conclusion 132

Chapter 4: EdgeHGN_MR: Edge Detecting Hierarchical Graph Neuron-based MapReduce 135
4.1 Neural Network based Classification Techniques 136
4.2 Associative Memory Concept for Implementing Large-scale Classification 137
4.2.1 EdgeHGN Approach for Cloud Data Access 139
4.3 EdgeHGN-based MapReduce 139
4.3.1 EdgeHGN_MRv1 141
4.3.2 EdgeHGN_MRv2 144
4.3.2.1 Bootstrapping 146
4.3.2.2 Algorithm Design 146
4.3.3 EdgeHGN_MRv3 148
4.4 Performance Evaluation 152
4.4.1 Classification Accuracy 153
4.4.2 Computational Efficiency 157
4.5 Comparative Performance Results 158
4.5.1 EdgeHGN-based MapReduce versus Distributed MapReduce 158
4.5.2 EdgeHGN-based MapReduce versus Pregel-like Graph Processing Systems (Giraph, GPS, Mizan and GraphLab) 160
4.5.2.1 System Setup and Datasets 161
4.5.2.2 PageRank Algorithm 162
4.6 Conclusion 166

Chapter 5: EdgeHGN Application in Fine-grained Wireless Sensor Networks 169
5.1 Distributed Data Processing Scheme for Wireless Sensor Networks 172
5.1.1 WSN Event Detection 173
5.1.1.1 Performance-specific Event Detection Schemes 173
5.1.1.2 Application-specific Event Detection Schemes 174
5.1.1.3 Distributed Pattern Recognition Scheme within WSN 175
5.2 Integrated EdgeHGN-WSN Processing Scheme 176
5.2.1 Dimensionality Reduction in Sensory Data 178
5.2.2 EdgeHGN Event Classification 179
5.2.2.1 Pattern Matching at Sensor Level 180
5.2.2.2 EdgeHGN Classification Approach 181
5.3 EdgeHGN-WSN Performance Evaluation 182
5.3.1 EdgeHGN-WSN Memory Utilization 187

5.4 Conclusions 188

Chapter 6: Case Study: Applying EdgeHGN based MapReduce Approach to Real World Big Data Processing Scenarios 191

6.1 EdgeHGN based MapReduce – High Level Framework 193

6.2 Case Study: Solarwinds and ITSM Big Data Processing using MapReduce and EdgeHGN based MapReduce 195

6.2.1 Solarwinds and ITSM Data Correlation Design Model 197

6.3 ITSM & Solarwinds Data Correlation Using EdgeHGN_MR 199

6.4 Comparing MR & EdgeHGN_MR for Data Correlation 204

6.5 Conclusions 205

Chapter 7: Conclusion 207

7.1 Research Summary 208

7.2 Research Contributions 213

7.3 Future Research 217

7.3.1 Algorithm-Specific Research 218

7.3.2 Application-Specific Research 218

Vita 221

References 223

List of Tables

Table 3.1: EdgeHGN total recall time complexity terms 92
Table 3.2: Big-O notations for Hopfield and EdgeHGN schemes in the network generation stage (Hopfield network, 2012) 97
Table 3.3: Big-O notations for the Hopfield and EdgeHGN networks in the recognition stage (Hopfield network, 2012) 98
Table 3.4: EdgeHGN storage and communication complexity terms 100
Table 3.5: Binary signatures for the image in Figure 3.17 114
Table 3.6: Discretisation of feature data values using variable-binning methods 128
Table 3.7: EdgeHGN networks setup details for processing four feature sets 128
Table 3.8: Recognition parameters with their respective definitions 129
Table 4.1: Hadoop Cluster Details 155
Table 4.2: Processing time comparison between EdgeHGN_MR and MapReduce 162
Table 4.3: Experiments Setup Details 164
Table 4.4: Dataset Details 165
Table 5.1: Temperature readings example with their respective binary signature 183
Table 5.2: Recognition parameters with their respective definitions 187
Table 5.3: Comparative analysis on recognition accuracy parameters between EdgeHGN and other classifiers for event recognition using three sensory data obtained from Catterall et al. (2003) (Smart-It 1, Smart-It 2, and Smart-It 3) 188
Table 6.1: ITSM and Solarwinds given data snapshots for data processing exercise 200
Table 6.2: Hadoop 4-node cluster setup for implementing MR and EdgeHGN_MR 201
Table 6.3: ITSM & Solarwinds data correlation processing time using MR & EdgeHGN_MR 208

List of Algorithms

Algorithm 3.1: SI Module Function 89
Algorithm 3.2: Voting Function 90
Algorithm 3.3: Adjacency Comparison Function (Base Layer) 91
Algorithm 3.4: Bias Calculation Function 91
Algorithm 4.1: EdgeHGN_MRv1 147
Algorithm 4.2: EdgeHGN_MRv2 151
Algorithm 4.3: EdgeHGN_MR_3 154
Algorithm 5.1: Pattern Matching Algorithm at the Sensor Level 185
Algorithm 6.1: EdgeHGN based MapReduce – High level framework 199
Algorithm 6.2: EdgeHGN_MR scheme for pattern matching between ITSM & Solarwinds 205

List of Figures

Figure 1.1: Pattern recognition through the characterisation of patterns in data 6
Figure 2.1: Feed-forward neural network model (Feed-forward neural network, 2011) 28
Figure 2.2: RNN with feedback link (Recurrent Neural Networks in Ruby, 2012) 28
Figure 2.3: A Hopfield network with four nodes (Hopfield network, 2012) 29
Figure 2.4: Schematic view of a self-organising map network (Schlegel, 2011) 31
Figure 2.5: SVM classification process (Introduction to SVM, 2012) 32
Figure 2.6: HDFS with multiple data nodes for storing data (Apache Hadoop, 2010) 37
Figure 2.7: MapReduce data flow structure (OpenSource Forum, 2011) 38
Figure 2.8: Apache Hadoop 2.0 (Apache Hadoop YARN, 2013) 40
Figure 2.9: YARN, Apache next-generation MapReduce (Apache Hadoop YARN, 2013) ... 41
Figure 2.10: Apache Mahout (Apache Mahout Software Foundation, 2012) 43
Figure 2.11: Pregel data model (Percolator, Dremel & Pregel, 2012) 45
Figure 2.12: PowerGraph solution to power-law graphs (GraphLab Open Source, 2009) 48
Figure 2.13: Gather-apply-scatter decomposition (GraphLab Open Source, 2009) 49
Figure 2.14: An input pattern BABBC is stored in a GN array where each row of the array represents a value and each column represents a position 57
Figure 2.15: Four arbitrarily chosen patterns – P1: ABBD, P2: ACCB, P3: BACA, P4: ABCD – have been stored in the GN array. The maximum bias size is three for storing four patterns, indicating that the storage requirement per node would not disproportionably increase with the increase in the stored patterns. 59
Figure 2.16: HGN with pattern size of seven and two possible values within the pattern (Nasution & Khan, 2008) 62
Figure 2.17: HGN compositions of (a) 2-D (7x5) and (b) 3-D (7x5x3) for pattern sizes 35 and 105, respectively (Nasution & Khan, 2008) 63
Figure 2.18: Transformation of the HGN structure (top) into an equivalent DHGN structure (bottom) (Khan & Muhamad Amin, 2007) 65
Figure 3.1: Auto-AM network to determine whether the input vector is ‘known’ or ‘unknown’ 71
Figure 3.2: Structural reduction on binary character images 74

Figure 3.3: EdgeHGN progressively removes unnecessary nodes from the two dimensional data using drop-fall for content reduction 75
Figure 3.4: Pixel from which to commence the drop-fall 76
Figure 3.5: Movement rules for the drop-fall algorithm 77
Figure 3.6: Hybrid drop-fall heuristic approach on character data patterns 77
Figure 3.7: EdgeHGN framework for distributed pattern recognition 79
Figure 3.8: EdgeHGN estimated and actual recall times for processing 10,000 stored patterns 96
Figure 3.9: Comparison of communication costs between the HGN, DHGN and EdgeHGN (Khan & Muhamad Amin, 2007) 105
Figure 3.10: EdgeHGN recall percentage for the three character patterns 'A' of different sizes 107
Figure 3.11: Seven different levels of random distortion applied to binary character patterns 108
Figure 3.12: EdgeHGN recall accuracy for various distortion rates 109
Figure 3.13: EdgeHGN node recall percentage for various distortion rates 110
Figure 3.14: Recall percentage rate for EdgeHGN *v*. DHGN 111
Figure 3.15: Response time for EdgeHGN *v*. DHGN 112
Figure 3.16: Recognition time for different sub-pattern sizes and different number of random sub-patterns 112
Figure 3.17: Block image with four different colours is divided into equally sized grids 113
Figure 3.18: Transformation of global colour histogram of image Lena from original image to various quantisation levels 115
Figure 3.19: Average recall and error rates for EdgeHGN greyscale image recognition on 40 16 KB binary images using various quantisation levels 116
Figure 3.20: Total recognition time for each EdgeHGN subnet in binary pattern recognition with different number of sub-patterns derived from 16 KB binary images 117
Figure 3.21: Recall error rates for binary image recognition of 100 facial image classes when tested against 1000 stored images using EdgeHGN, DHGN, SVM & BPNN schemes. 118
Figure 3.22. Edge map after applying Global Binary Signature and Sobel's edge detection 120
Figure 3.23. Fifty different individuals in the face image dataset obtained from the Face Recognition Data 121

Figure 3.24. Applying the Sobel operator on both the base image and the test image before pattern matching 122
Figure 3.25. Applying four possible drop-fall directions to the input pattern 123
Figure 3.26. EdgeHGN recognition times after applying four drop-fall schemes on a test image 123
Figure 3.27. Error values for EdgeHGN processing 50 facial image classes of 1000 test images 124
Figure 3.28. Error values for EdgeHGN and BPNN processing 50 facial image classes of 1000 test images 124
Figure 3.29: (Top) images contaminated by both Gaussian noise and salt-and-pepper noise with σ = 10 and s = 30% (bottom) recognition results using the EdgeHGN scheme ... 126
Figure 3.30: (Top) images contaminated by both Gaussian noise and random-valued noise with σ = 10 and s = 25% (bottom) recognition results using the EdgeHGN scheme ... 126
Figure 3.31:EdgeHGN classification results on four different features of numeral character objects 130
Figure 3.32: EdgeHGN classification best average results on four different features of numeral character objects 130
Figure 3.33: Comparative study on error rates between EdgeHGN and other classifiers for similar dataset with respective features 131
Figure 4.1: EdgeHGN_MRv1 Architecture 145
Figure 4.2: EdgeHGN_MRv2 architecture 148
Figure 4.3: EdgeHGN_MRv3 Architecture 152
Figure 4.4: Handwritten digits (MNIST Database) 156
Figure 4.5: Accuracy rate of EdgeHGN_MRv1 157
Figure 4.6: Accuracy rate of EdgeHGN_MRv2 157
Figure 4.7: Accuracy rate of EdgeHGN_MRv3 158
Figure 4.8: Accuracy rate comparison between EdgeHGN_MRv1, EdgeHGN_MRv2 and EdgeHGN_MRv3 158
Figure 4.9: Accuracy rate stability comparison between EdgeHGN_MRv1, EdgeHGN_MRv2 and EdgeHGN_MRv3 159

Figure 4.10: Computational efficiency comparison between EdgeHGN_MRv1, EdgeHGN_MRv2 and EdgeHGN_MRv3 160
Figure 4.11: MapReduce implementation of PageRank algorithm where the mapper emits initial PageRank values for every node. The reducer receives all PageRank contributions for a given node, adds them up, and emits its contribution to its own outgoing links 166
Figure 4.12: Computing time comparison between Giraph, GPS, Mizan, GraphLab and EdgeHGN_MR using PageRank algorithm for ego-Twitter Dataset 167
Figure 4.13: Computing time comparison between Giraph, GPS, Mizan, GraphLab and EdgeHGN_MR using PageRank algorithm for ego-Twitter Dataset 167
Figure 4.14: Maximum memory usage comparison between Giraph, GPS, Mizan, GraphLab and EdgeHGN_MR using PageRank algorithm for ego-Twitter Dataset 168
Figure 4.15: Maximum memory usage comparison between Giraph, GPS, Mizan, GraphLab and EdgeHGN_MR using PageRank algorithm for ego-Twitter Dataset 169
Figure 5.1: EdgeHGN distributed event detection framework 181
Figure 5.2: Sensor node placement in a Cartesian grid where each node is allocated to a specific grid area 182
Figure 5.3: EdgeHGN event detection result for a test using 1800 light sensor datasets (Smart-It 1) (x-axis) with a threshold of 100 (Basirat & Khan, 2013) 187
Figure 5.4: Comparative analysis on recognition parameters rates between EdgeHGN and other classifiers for event recognition using three sensory data obtained from Catterall et al. (2003) (Smart-It 1, Smart-It 2, and Smart-It 3) 189
Figure 5.5: EdgeHGN Recognition time for 1800 sensor data (x-axis) taken from Smart-It 1, Smart It 2 and Smart It 3 datasets 190
Figure 5.6: Maximum memory consumption for each EdgeHGN subnet for different pattern sizes. EdgeHGN uses minimum memory space with small pattern size 191
Figure 6.1: SPSS modelling process of linking ITSM and Solarwinds using EdgeHGN_MR scheme 202
Figure 6.2: Architectural overview of ITSM and Solarwinds data correlation project 203
Figure 6.3: EdgeHGN_MR architecture for pattern matching between ITSM and Solarwinds datasets 206

Figure 6.4: ITSM tickets raised due to Solarwinds alerts 206
Figure 6.5: Service field for ITSM tickets raised due to Solarwinds alerts 207
Figure 6.6: Main causes of Solarwinds alerts 207
Figure 6.7: Average distribution of Solarwinds alerts during day 208
Figure 6.8: Processing time of performing data correlation between ITSM and Solarwinds datasets using both MR and EdgeHGN_MR schemes 209

List of Abbreviations

AM	Associative Memory
ANN	Artificial Neural Network
API	Application Programming Interface
BAA	Bias Associative Array
BAM	Bidirectional Associative Memory
BMU	Best Matching Unit
BPNN	Back-Propagation Neural Network
BSP	Bulk Synchronous Parallel
CAM	Content-Addressable Memory
CBIR	Content-Based Image Retrieval
CF	Collaborative Filtering
CHN	Continuous Hopfield Network
CPU	Computer Processing Units
DBMS	Database Management Systems
DDOS	Distributed Denial of Service
DHGN	Distributed Hierarchical Graph Neuron
DHN	Discrete Hopfield Network
DPR	Distributed Pattern Recognition
EBI	European Bioinformatics Institute
EdgeHGN	Edge Detecting Hierarchical Graph Neuron
FAM	Fuzzy Associative Memory
GAS	Gather-Apply-Scatter
GCH	Global Colour Histogram
GIS	Geographical Information System
GN	Graph Neuron
GPS	Global Positioning System
GPU	Graphical Processing Unit
HDFS	Hadoop Distributed File System
HGN	Hierarchical Graph Neuron

HPC	High-Performance Computing
InSAR	Interferometric Synthetic Aperture Radar
ITSM	IT Service Management
IoT	Internet-of-Things
IP	Internet Protocol
JVM	Java Virtual Machine
KNN	K-Nearest Neighbour
LDA	Linear Discriminant Analysis
LLE	Local Linear Embedding
MAM	Morphological Associative Memories
MGI	McKinsey Global Institute
ML	Machine Learning
MNIST	Mixed National Institute of Standards and Technology
MPI	Message-Passing Interface
MRI	Magnetic Resonance Imaging
NM	NodeManager
OCR	Optical Character Recognition
PCA	Principal Components Analysis
RDBMS	Relational Database Management System
RM	ResourceManager
RNN	Recurrent Neural Network
SI	Stimulator/Interpreter
SOM	Self-Organising Map
SoS	System-of-Systems
SPIE	Society of Photo-optical Instrumentation Engineers
SPSS	Statistical Package for the Social Sciences
SV	Support Vector
SVM	Support Vector Machine
TSP	Travelling Salesman Problem
UAI	Uncertainty in Artificial Intelligence
WSN	Wireless Sensor Network

List of Publications

Publications arising from this thesis include:

Book Chapters

Basirat, A. H., & **Khan, A. I. (2010**). Building context aware network of wireless sensors using a scalable distributed estimation scheme for real-time data manipulation, *in Wireless Sensor Network*. Chapter 22, In-Tech Publication, DOI: 10.5772/13756.

Basirat, A. H., Khan, A. I., & **Schmidt**, **H. W. (2015**). Pattern recognition for large-scale data processing, *in Strategic Data-Based Wisdom in the Big Data Era*, IGI Global Publication, pp. 198–208.

Conference Proceedings

Basirat, A. H., & **Khan, A. I**. **(2009**). Building context aware network of wireless sensors using a novel pattern recognition scheme called Hierarchical Graph Neuron. *Proceedings of the 2009 IEEE International Conference on Semantic Computing (ICSC 2009)*, 14–16 September, IEEE Computer Society, San Francisco, CA, pp. 487–494.

Basirat, A. H., Muhamad Amin, A., & **Khan, A. I**. **(2010**). Under the cloud: A novel content addressable data framework for cloud parallelization to create and virtualise new breeds of cloud applications. *Proceedings of the Ninth IEEE International Symposium on Network Computing and Applications*, 15–17 July, IEEE Computer Society, Cambridge, MA, pp. 168–173.

Basirat, A. H., & **Khan, A. I**. **(2010**). Evolution of information retrieval in cloud computing by redesigning data management architecture from a scalable associative computing perspective. *Neural Information Processing. Models and Applications, Lecture Notes in Computer Science*, volume 6444, pp. 275–282.

Basirat, A., & **Khan, A**. **(2011**). Introducing a novel data management approach for distributed large-scale data processing in future computer clouds. *Neural Information Processing, Lecture Notes in Computer Science*, volume 7063, pp. 391–398.

Basirat, A. H., & **Khan, A. I**. **(2012**). A novel associative model of data: Toward a distributed large-scale data processing scheme for future computer clouds. *Proceedings of the IEEE 11th International Symposium on Network Computing and Applications*, 23–25 August, IEEE Computer Society, Cambridge, MA, pp. 163–166.

Basirat, A. H., & **Khan, A. I. (2013**). Scalable event detection in wireless sensor networks using a novel content-based pattern recognition scheme. *Proceedings of the 3rd International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering (PARENG 2013)*, Civil-Comp Press, Stirlingshire, UK, pp. 53–59.

Basirat, A. H., & **Khan, A. I. (2013**). Introducing an intelligent MapReduce framework for distributed data processing in clouds. *Proceedings of the 12th IEEE International Symposium on Network Computing and Applications (NCA 2013)*, Cambridge, MA, pp. 61–64.

Basirat, A. H., **Khan, A. I.,** & **Srinivasan**, B. **(2014**). Highly distributable associative memory based computational framework for parallel data processing in cloud. *Lecture Notes in Computer Science, Social Informatics and Telecommunications Engineering*, volume 131, pp. 66–77.

Basirat, A. H., & **Khan, A. I. (2015**). A highly distributable computational framework for fast cloud data retrieval. *Proceedings of the 14th IEEE International Conference on Machine Learning and Applications (ICMLA 2015*), Miami, FL, USA, pp. 246 – 250.

Chapter 1

Introduction

Recent advancements in computing technology and data analysis have resulted in the generation of massive volumes of highly complex data, leading to a call for a paradigm shift in computing architectures and large-scale data processing frameworks. Jim Gray, a distinguished database researcher and the manager of Microsoft Research's e-Science group, referred to this shift as the '*fourth paradigm*' (Hey. et. al., 2009), with the first three shifts representing experimental, theoretical and computational science. Gray suggested that the only solution to this outgrowth of big data, commonly known as '*data deluge*', is to develop a new set of computing, processing and analysing tools. He also argued that current computing frameworks are becoming more incapable of handling data-intensive tasks over time due to the constantly and rapidly growing latency gaps between multi-core computer processing units (CPUs) and mechanical hard disks (Gray. et. al., 2006). In fact, with emerging interest to leverage massive amounts of data that are available in open sources, such as the Web for solving long-standing information retrieval problems, the question of how to effectively process immense datasets is becoming increasingly relevant. The outgrowth of big data has significant implications for the development of computing

applications (Hey & Trefethen, 2003). According to Anderson (2008), the chief editor of Wired magazine:

> Sixty years ago, digital computers made information readable. Twenty years ago, the Internet made it reachable. Ten years ago, the first search engine crawlers made it a single database. Now Google and like-minded companies are sifting through the most measured age in history, treating this massive corpus as a laboratory of the human condition. They are the children of the Petabyte Age. Kilobytes were stored on floppy disks. Megabytes were stored on hard disks. Terabytes were stored in disk arrays. Petabytes are stored in the cloud. As we moved along that progression, we went from the folder analogy to the file cabinet analogy to the library analogy to – well, at petabytes we ran out of organizational analogies.

Thus, the world of big data is in need of high levels of scalability. As human beings, our brains could be represented as a large-scale distributed and interconnected network of sensory systems and memories. Observing, recognising and recalling what we have seen all form a considerable portion of the activities performed within these large-scale interconnected networks. Provided that an optimal solution is found for the scalability problem, the Internet can provide us with levels of interconnectivity and complexity that bear a resemblance to the human brain. Harnessing the massive potential embodied within these distributed networks of interconnected high-performance machines may provide recognition and processing capabilities for large-scale and highly complex data.

1.1 Recognition at Large Scale and Big Data

Transforming big data into valuable information is a significant challenge that real-world systems must deal with. In fact, more data translates into more effective and efficient algorithms; hence it is quite reasonable to take advantage of the tremendous amounts of data that surround us. In this regard, the development of powerful high-

resolution data-capture instruments and sensors in areas such as satellite and biomedical imaging has resulted in a massive production of voluminous and complex data. In satellite imaging applications, including the geographical information system (GIS) and the global positioning system (GPS), depending on the resolution of the images captured, the size of data generated can be enormous.

These large datasets need to be properly processed before they can be used in relevant applications. In biomedical imaging, intelligent processing schemes are usually deployed to extract critical information from high-dimensional images obtained through complex imaging approaches, such as Magnetic Resonance Imaging (MRI), to help medical experts with their diagnoses. With the advent of high-resolution imaging techniques and recent technological developments in high-speed networking and storage fields, medical experts can conduct a collaborative diagnosis by collecting data from various sensory and imaging equipments over large scattered networks and storing and accessing these data within distributed repositories. With all these capabilities available, the amount of data produced and processed can be at the Internet-scale. In addition, significant advancements in large-scale scientific analysis activities have resulted in the introduction of complex and state-of-the-art technologies. One example is the advent of next-generation DNA sequencing technology producing an excessive amount of sequence data. This enormous amount of data must be properly and efficiently stored, indexed and delivered to scientists for further processing. Given that, in the modern science of genetics, genotypes can explain phenotypes, the effects of this advanced technology are nothing but transformative (Elaine, 2008). The European Bioinformatics Institute (EBI), which holds a huge central repository of sequence data called EMBL-bank, increased its storage capacity from 2.5 petabytes in 2008 to 18 petabytes in 2013 (EBI, 2013). Medical experts believe that in not a very distant future, sequencing an individual's genome will be as easy as getting a simple blood test, thereby introducing a new era of personalised medicine, where prescriptions can be specifically developed and targeted for an individual. As mentioned in the work of (Fox. et. al., 2005), the development of sophisticated data-capture instruments and sensors, such as the Large

Hadron Collider and Interferometric Synthetic Aperture Radar (InSAR), in high-energy physics has resulted in the consistent generation of large volumes of highly complex multi-dimensional data.

In fact, Petabyte datasets are rapidly becoming the norm, and the trends are obvious; our ability to produce and store data is quickly overwhelming our ability to process what we generate and store. In this regard, the need for highly sophisticated computational schemes is somehow prevalent, as the volumes of generated data make it absolutely impractical for data analysts to conduct any form of data processing without having the right tools available. However, existing data mining schemes are mostly suffering from various shortcomings such as the algorithmic complexity of deployed methods. For example, depending on the form of pruning applied the order of complexity for the decision tree classification tool can range from $O(n\log n)$ to $O(n^2)$ or even worse (Kamath & Musick, 1998). This in turn makes it practically infeasible for use in large-scale data processing approaches. Moreover, the rapid expansion of integration between various computational devices and sensor networks with the Internet has created a pervasive computational framework known as the Internet-of-Things (IoT) (Kopetz, 2011). This development builds a bridge between the physical and information domains and creates a smart space where a large number of high-performance computational devices can interact in real time to provide various services – a model that is analogous to the human biological nervous system. The problem arises when an enormous amount of data has been captured from various computing systems and there is an urgent need to process this data load somewhat in real time.

1.2 Cloud Computing and Large-scale Data Processing

Cloud computing offers a pay-per-use paradigm for providing services over the Internet in a scalable and distributed manner. In this regard, supporting data-intensive applications is an essential requirement for computer clouds. However, the dynamic and distributed nature of cloud computing environments leads to complex and

cumbersome data management processes, especially in the presence of real-time data-processing/database-updating tasks. While the possibilities provided by the parallelisation and distribution of data in clouds have introduced some efficiency, existing relational and object-oriented data models in particular result in complicated storage and retrieval processes, especially when dealing with large parallel real-time data. Chaiken et al. (2008) observed that the challenge of processing large datasets in a scalable and cost-efficient manner has rendered traditional database solutions prohibitively expensive. At the other end of the spectrum, high-performance computing (HPC) has advanced rapidly but has generally focused on computational complexity and performance improvements. Virtual HPC in the cloud has significant limitations, especially when big data is involved. According to Shiers et al. (2009), 'it is hard to understand how data-intensive applications; such as those that exploit today's production grid infrastructures; could achieve adequate performance through the very high-level interfaces that are exposed in clouds'. Thus, the question of how to effectively process large-scale datasets is becoming increasingly relevant. Further, existing data management schemes do not work well when data is partitioned among numerous available nodes dynamically. Approaches towards parallel data processing in the cloud, which offer greater portability, manageability and compatibility of applications and data, are yet to be fully explored.

1.3 Big Data, Feature Extraction and Pattern Recognition

A practical solution to the challenge of voluminous datasets can be implemented through the use of pattern recognition/matching models where patterns represent a set of data captured over a certain period. To extract useful information from the captured data, feature extraction needs to be implemented in an efficient manner. Feature extraction can be viewed as a mapping from a typically high-dimensional data space to a reduced dimension space, while maintaining some key properties of the data. This approach for feature/pattern extraction is commonly referred to as data mining, which involves the process of uncovering patterns, determining associations

between data objects, detecting anomalies and even predicting future data trends. In this regard, pattern recognition is a common processing tool used in a wide range of applications, including medical diagnosis, environment and condition monitoring, decision-making, and various types of scientific explorations. However, when it comes to processing an enormous amount of data, common pattern recognition schemes that operate within a CPU-centric environment may not scale well to deal with data in the order of gigabyte or petabyte scales. Hence, a paradigm shift in data-processing approaches is essential to handle recognition at the Internet-scale.

1.4 Pattern Recognition for Large-Scale Data Processing

In recent years, interest in pattern recognition has been dramatically renewed mainly due to the data explosion phenomenon that is currently taking place. In simple terms, a pattern may be expressed through the use of a common denominator among multiple instances of an entity. In this regard, pattern recognition schemes aim to make the process of observing and detecting these common characteristics explicit in such a way that they can be employed in computational devices to facilitate data processing by learning and adapting to its characteristics (see Figure 1.1).

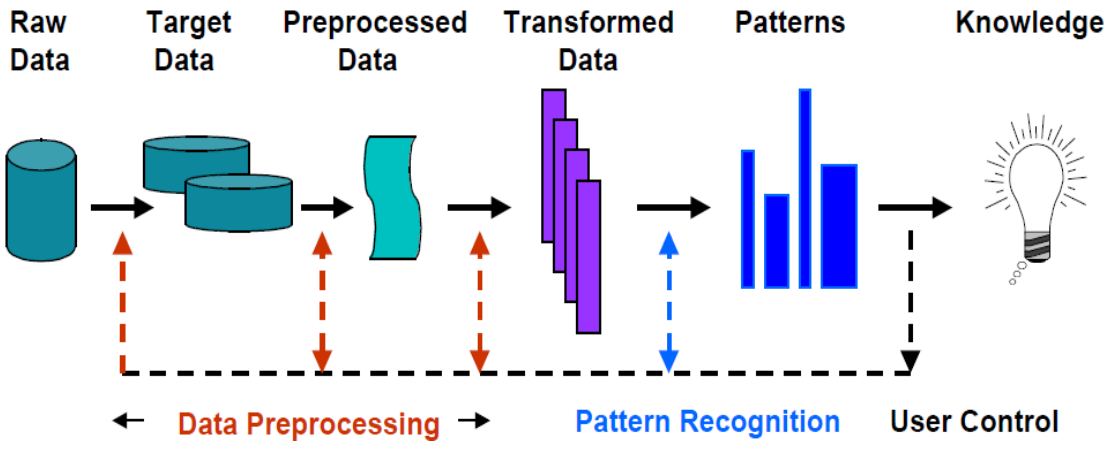

Figure 1.1: Pattern recognition through the characterisation of patterns in data

The data deluge, along with rapid advancements in data capture technologies, such as in sensor networks, has led to a call for a paradigm shift in recognition approaches and analytical schemes. In fact, current recognition schemes must be reconsidered from a larger perspective to scale with the rapid growth of the data (i.e. from an Internet-scale perspective). Scalability is one of the most important factors to consider when deploying an efficient pattern recognition model. To meet the requirements of existing Internet-scale data, the capability of pattern recognition schemes should continue to grow and scale to minimise the risk of becoming obsolete. In this regard, Pal and Mitra (2004) restated the question of scalability as follows:

> 'Can the pattern recognition algorithm process large data sets efficiently, while building from them the best possible models?'

The recent surge in interest for scalable pattern recognition schemes has been accompanied by exponential growth of data sizes generated by digital media (images/audio/video), web authoring, scientific instruments and physical simulations. Thus, the question of how to effectively process these immense datasets is becoming increasingly important. Nevertheless, most of existing models suffer from excessive computational complexity when dealing with highly complex datasets.

1.4.1 Common Barriers

To achieve an adequate level of efficiency, a numbers of barriers must be overcome when implementing pattern recognition. These include, but are not limited to:

i. **Large Data:** As the size of the data generated/stored increases over time, pattern recognition approaches should become more capable of coping with this outgrowth of the data in the most efficient and effective way. This requires taking into account all relevant data considerations from storage and transport perspectives.

ii. **Hi-dimensional Data:** With current advancements in data capture technologies, there are many application domains where data to be extracted from the

environment is of considerably higher dimensionality and is not basically spatial (e.g., biological data measuring gene features). In this context, pattern-modelling schemes should be able to incorporate higher dimensionalities of data in their processing/implementation.

iii. **Algorithmic Complexity**: To measure the performance of existing pattern recognition models, we need to consider two aspects of algorithm performance, namely time and space. First, how fast does the algorithm perform, and what affects its runtime? Second, what type of data structure can or should be used to maximise performance? Although existing pattern recognition models are very powerful and are capable of providing efficient solutions, they suffer from excessive complexity, mainly due to their iterative nature along, with complex mathematical foundations. A large portion of them are exponential and hence infeasible for implementation in large-scale data scenarios. Moreover, their high cost of implementation in terms of time and space makes them operationally costly for large-scale data.

Hence, any scheme for processing big data should be capable of addressing increasing size and dimensionality of data while minimising implementation complexity.

1.4.2 Possible Solutions

There are some major techniques available for scaling up pattern recognition in dealing with big data:

i. **Data Approach:** In this model, captured data are pre-processed and modified in preparation for the recognition process. A number of techniques have been proposed in the literature for this purpose, including data reduction (Chow & Huang, 2008), dimensionality reduction (Rueda & Herrera, 2008), and data partitioning (Kbir, et. al., 2000). The ultimate goal is to reduce/minimise the size and dimension of the data for faster and more efficient recognition; however, the approach is liable to overlook the importance of data integrity by reducing the size of the data domain.

ii. **Learning Approach:** A learning mechanism is a common component among pattern recognition schemes, and researchers have made many attempts to reduce the computational complexity of the learning phase in favour of achieving scalable models with a faster recognition speed. Examples include active learning (Cheng & Wang, 2007) and incremental learning (Schlimmer & Granger, 1986). A risk associated with this approach is that recognition accuracy will be compromised to reach faster recognition. Moreover, in many cases of learning approach, the issue of over-fitting is still present. This is mainly due to the fact that a model is more inclined to '*memorise*' training data while putting less effort into '*learning*' to generalise from data trends.

iii. **Distributed Computing Approach:** Advancements in parallel processing technologies and improved networking capabilities have resulted in shifting large-scale computations to be performed within the body of the network exploiting resource-sharing capabilities of distributed systems to cope with the incremental growth of resource demands. This approach benefits from guaranteed levels of reliability, availability and scalability due to its large-scale distributed nature of operations. In this regard, cloud computing may be viewed as a good example of a distributed computing system that is capable of providing scalable services using largely distributed resources to perform complex and computationally expensive tasks (e.g., recognition at the Internet-scale level).

Of the above three computing approaches, distributed processing is more promising for scaling up with today's outgrowth of data. This technique is fundamentally different in the sense that adaptation to bio-inspired modelling of the brain being readily possible under parallel distributed processing when dealing with an excessive amount of data. However, this is not the case for the data and learning approaches. In effect, major advancements in parallel computing technology from simple multi-threading computational models to multi-core and graphical processing unit (GPU) forms of distributed computing have enabled large-scale processing to be performed in more elegant and efficient ways. Nevertheless, some existing models are extremely complex and highly cumbersome to parallelise. Moreover, the

scalability of deployed methods for processing voluminous data is still an open problem that needs to be addressed. Further, existing data management schemes do not work well when data is partitioned among numerous available nodes dynamically. Thus, the question of how to effectively process large-scale datasets is becoming increasingly relevant.

1.5 Motivation and Research Objectives

The dynamic and distributed nature of cloud computing environments, as well as their exponential growth, makes real-time data management complicated and storage, updates and analytics costly (Szalay. et. Al., 2006). This thesis hypothesises that fundamental changes and improvements in data access and movements are possible and beneficial for cloud-based data processing. That is, transforming big data into valuable information requires a fundamental rethink of how future data management models will need to be developed on the Internet. As previously discussed, distributed pattern recognition approaches can be investigated as an alternative solution for large-scale data processing. Nevertheless, some major obstacles must be overcome before these approaches are considered suitable for cloud environments. In fact, existing distributed pattern recognition models have been mainly formed along a top-down approach – that is, from the interface design towards hardware and computing resources development and management. In this approach, relatively CPU-centric (or sequential-based) algorithms are instrumented and enhanced to function in a distributed manner. In addition to this limitation, most of current approaches implement distribution partially – that is, in the context of training and validation (e.g., feed-forward neural networks and self-organising maps).

In regards to distributed pattern recognition schemes dealing with large-scale data, the main motivation for the research work conducted in this thesis lies in the need for a bottom-up approach – that is, from existing heterogeneous resources to the development of abstraction layers for general usage. For this purpose, associative memory concepts open a new pathway for accessing data in a highly distributed environment that will facilitate a parallel-distributed computational model to

automatically adapt to the dynamic data environment for optimised performance. The problem lies in marrying such concepts with relevant advanced parallel processing patterns. With this in mind, this thesis targets a new type of data processing approach that will efficiently partition and distribute data for clouds and facilitate content-based access for a wide range of applications. Thus, a fully distributed pattern recognition scheme that can work with a parallel-distributed computational model such as MapReduce will provide a reusable cloud-based framework for a range of applications, from image search and sensor data analysis to the control of cyber-physical infrastructure, mobile equipment and devices.

The ability to partition data efficiently and automatically will allow elastic scaling of system resources and remove one of the main obstacles in provisioning data-centric software-as-a-service in clouds. Improved data management, where data are optimally and automatically distributed – stands to improve application performance through efficient data access. In a nutshell, the fundamental motivation for this research work is to establish a fully distributable and highly scalable pattern recognition scheme for Internet-scale data analysis that can enable fast large-data retrieval across voluminous datasets associatively, utilising a parallel approach. Doing so will yield a new form of database-like functionality that can scale up or down over the available infrastructure without interruption or degradation, dynamically and automatically. In summary, the research work conducted in this thesis aims to meet the following objectives:

i. Provide a distributed data access scheme that enables data storage and retrieval by association where data points are treated as patterns, thereby circumventing the scaling issue experienced within referential data access mechanisms. This will also enhance the overarching relationships among distributed datasets for a variety of pattern recognition and data-mining applications.
ii. Provide a distributed data management scheme that is beneficial for the operational requirements of big data processing, thereby enabling relevant data to be readily available for large-scale computations. This can be achieved by redesigning the data management architecture from a scalable associative

computing perspective to create a database-like functionality that can scale up or down dynamically over the available infrastructure without any interruption or degradation.

iii. Significantly reduce the number of processing messages and increase tolerance to failure by adapting higher data representation techniques.

iv. Validate results and find asymptotical limits of the technique through simulation environments and real-world examples to establish the usefulness of our approach.

1.6 Hypotheses and Methodologies

Scalability in the context of large-scale pattern recognition can be defined as 'the ability to either handle growing amounts of patterns in a graceful manner or to be readily enlarged' (Bonndi, 2000). In this context, this thesis examines fundamental research on scalability for pattern recognition in association with big data. A number of different large-scale data processing approaches will be extensively reviewed and examined, and an effective solution for the scalability problem will be proposed.

Associative Memory Hypothesis: The main hypothesis of this research is that loosely-coupled associative techniques, which have not widely considered to date, can provide an efficient framework for cloud data management. This approach will entail two-fold benefit. First, applications based on associative computing models can efficiently utilise the underlying hardware to scale up and down the system resources dynamically. Second, it will remove the main obstacle to providing scalable partitioning and data distribution in the cloud, thereby providing a superior solution for handling data-intensive applications and the system infrastructure to support a pay-per-use basis. This thesis aims to design a neural networking computational framework that will result in an architecture that can scale up to use many neural network nodes operating in parallel, capable of performing large-scale data processing using neural computations in real-time. As this thesis aims to perform

computation at a scale that greatly exceeds the capacity of any single device, it proposes the scalability hypothesis.

***Scalability Hypothesis*:** Neural networking computational systems must be able to be increased in real-time by scaling the system to multiple devices. The scalability hypothesis indicates that a neural computation system must be a parallel processing system. If a neural network is considered a parallel processing system, then neurons can be considered computing resources, while connections can be considered communication resources. This leads to the communication-centric Hypothesis.

Communication-Centric Hypothesis: The scalability of a neural computation system is communication-bound, not compute-bound. The communication-centric hypothesis means that the work involved in modelling communication in a neural computation system dominates the work involved in modelling the behaviour of neurons. It also means that the scale of the neural network that can be handled by a neural computation system in real-time is bounded by the availability of communication resources in this system rather than the availability of compute resources. This leads to the inter-node and intra-node bandwidth hypothesis.

Inter-node and Intra-node Bandwidth Hypothesis: The scale of the neural network that can be handled by a neural computation system in real-time is bounded by inter-node and intra-node communication bandwidths. These provide important considerations when designing a massively parallel neural computation system where it is necessary to employ an implementation platform that provides effective inter-node and intra-node communication to maximise the scale of the neural network.

This thesis will provide justification for these hypotheses. To address the broad aims targeted in this research and prove its hypotheses, a number of objectives have been formulated:

i. Conduct a comprehensive review of current cloud data management schemes and existing large-scale data processing models. The ultimate goal for this objective is to create a repository of knowledge playing the role as a foundation for the process of creating algorithms that provide demonstrably more efficient, robust

and scalable end-to-end data access for distributed real-time information processing in computer clouds through distributed pattern analysis.

ii. Conduct an extensive study of a distributable recognition scheme for big data and benchmark the proposed model against other established big data processing schemes such as Hadoop MapReduce, Pregel and GraphLab. To achieve this, comprehensive analysis will be undertaken to best represent the learning and distribution mechanisms. This will enable us to better evaluate the computational complexity, scalability and fault tolerance of the proposed scheme. Further studies will be carried out to test the applicability of this technique in performing Internet-scale data processing, provided we can compare its characteristics with state-of-the-art techniques in cloud data management. This will in turn enable us to create a proposal for large-scale distributed data processing for cloud environments.

iii. Scalability is an important factor when dealing with Internet-scale data. As a result, any scheme for large-scale data processing should have the ability to scale up for any given amount and dimension of data. Hence, a detailed study will be conducted of the scalability aspects of the proposed model to ensure that the scheme will be applicable to large-scale data scenarios. To achieve this, we will examine processing time and accuracy as two fundamental parameters for this study.

iv. Formulate a distributed intelligent cloud data management model that enables seamless data access and distribution using single-cycle learning associative memory-based algorithms. This is done by developing an associative MapReduce framework that allows complex data representations to be used as keys for map and reduce operations. This will allow content-association-based data retrieval and storage within the cloud.

V. Establish the veracity of the datasets through rigorous testing, and benchmark the results against state-of-the-art techniques used in the literature. This will enable us to identify novel scheme usages for big data processing in computer clouds.

1.7 Research Contributions

The contributions of this thesis can be summarised as follows:

i. A distributed data access scheme is proposed that enables data storage and retrieval by association where data records are treated as patterns; hence, finding overarching relationships among distributed datasets becomes easier for a variety of pattern recognition and data-mining applications. The proposed scheme will be suitable for the operational requirements of computer clouds and will enable relevant data to be readily available for large-scale computations. An extension towards data representation for distributed pattern recognition algorithms will be presented, which will significantly reduce the number of messages and increase tolerance to failure by adapting higher data representation techniques.

ii. This thesis will reconcile MapReduce with associative memory concepts, in particular for adaptive and fast data access, aggregation and movement will be a key contribution of this thesis. This will improve MapReduce-based parallel processing by uniformly formatting data in a standard two-dimensional representation. It will eliminate data imbalances and complete the transition to cloud by replacing referential data access mechanisms with more versatile and distributable associative functions that allow complex data relations such as images to be easily encoded into the keys as patterns. These patterns can be applied in a variety of applications that require content recognition, such as image databases, searches within large multimedia files and data mining. The algorithmic strengths of the MapReduce approach are investigated for the first time in regards to the effectiveness of one-shot learning-based parallelism provisioned via our distributed pattern recognition approach.

iii. This thesis will investigate the capabilities of the proposed scalable pattern recognition scheme in the context of distributed data processing in large-scale cloud of wireless sensor networks (WSNs). The research looks specifically into the applications of associative-memory based approaches in a resource-constrained WSN network to demonstrate the ability of the proposed technique to provide an effective front-end recognition scheme for event detection. This

approach outlines a new type of the WSN that detects macroscopic events by collating diverse sensor data, locally and in real-time, into meaningful patterns and eliminates the bottleneck problem by offering on-site computations through adoption of a completely distributed and decentralised technique.

1.8 Thesis Outline

The rest of this thesis is organised as follows.

Chapter 2 will undertake a comprehensive review of various pattern recognition approaches to determine their computational complexity and how effectively they can address scalability concerns. Moreover, different state-of-the-art techniques used in the literature for Internet-scale data will be discussed, along with their pros and cons. The chapter will also provide a detailed introduction to graph neuron (GN) and hierarchical graph neuron (HGN), which implement an effective scalable associative memory scheme through their parallel in-network processing frameworks. The chapter will also discuss their strengths and limitations. The primary goal of this chapter is to provide some background information and build a strong foundation for upcoming chapters, where we propose novel associative-memory-based schemes for large-scale data processing.

Chapter 3 will present a novel distributed single-cycle pattern matching scheme, referred to as edge detecting hierarchical graph neuron (EdgeHGN). The chapter will detail all of the features and characteristics of the scheme that we use throughout this thesis. The study will include discussions on required pre-processing steps, scheme architecture, data representation, communication and learning mechanisms, accuracy, and speed of implementation. In addition to scalability and computational complexity evaluations, various experiments are performed to examine the applicability of the scheme to large-scale datasets.

Chapter 4 will present a detailed discussion of a novel distributed pattern recognition model for Internet-scale data processing. The chapter will examine the capabilities of our evolved EdgeHGN based MapReduce scheme in different real-life testing scenarios. The applicability and efficiency of the proposed model for multiple

pattern recognition domains will be analysed in further detail. Further, this chapter will investigate an extension of our proposed distributed data management scheme for different data-intensive scenarios. In particular, three data-intensive scenarios are considered: dealing with large datasets, handling large training volumes and a neural network with an excessive number of processing neurons. In addition, a comprehensive evaluation of the proposed model will be conducted, to benchmark the proposed scheme's performance against some of the state-of-the-art techniques used in the literature. This will include scalability tests and performance tests for large-scale datasets, and recognition accuracy tests using distorted and noisy patterns.

Chapter 5 will investigate the capabilities of the proposed scheme in the context of distributed data processing in large-scale cloud of wireless sensor networks (WSNs). The chapter will examine WSNs as a platform of operation for EdgeHGN distributed pattern matching, and it will provide an extensive set of performance benchmarks. The aim of this chapter is to demonstrate the ability of the proposed recognition technique to learn and recognise complex patterns using minimal information and resources to effectively perform classification tasks.

Chapter 6 will present the results of a 6-month AMSI internship project conducted at a major pharmaceutical company to showcase a study on the adoption of our proposed distributed data processing scheme for analysing real-world large-scale environmental monitoring data and IT service management data. The results presented in this chapter will validate the research findings by providing access to large-scale commercial data to test the effectiveness of our approach as a crucial element in the cross-validation of research contributions.

Finally, Chapter 7 will conclude this thesis by summarising its contributions and discussing potential future works.

This Page Intentionally Left Blank

Chapter 2

Distributed Pattern Recognition and Data Management at Internet-scale

To deal with the voluminous data gathered from the entire Internet in an efficient form and at a reasonable cost, search engines utilise a customised distributed data-processing framework that is deployed on large clusters of computing nodes rather than relying on traditional database management systems (DBMSs). Relying on his previous experience as Inktomi (now part of Yahoo!) co-founder, Eric Brewer claimed that novel data-intensive frameworks (e.g., search engines) should 'apply the principles of databases, rather than the artifacts' (Brewer, 2005), as typical DBMSs are mostly overly generalised with some redundant features that, in the case of search engines, could introduce redundancy with costly overheads. As a result, search engines work better with simplified distributed and parallel data-processing schemes when dealing with large-scale data. Moreover, due to changes in the data access patterns of applications and the necessity to use thousands of compute nodes, the

main cloud computing providers have integrated specific frameworks for parallel-distributed data processing in their product offerings, making them more suitable for end-users to access their services and deploy their applications. Thus, efficiencies through the widespread use of multi-core CPUs, cost reductions for commodity hardware, enhanced performance and higher reliability in use are derived from an architectural paradigm that favours a massively distributed data-processing platform running on an extensive number of cheap computing nodes. Large data operations, such as processing crawled documents or reproducing a web index, are divided into several independent jobs that are distributed across the network among the available processing nodes and computed in parallel within the network. To simplify the development of distributed applications on top of such highly distributed architectures, customised data-processing frameworks are developed and deployed. Well-known examples are Google's MapReduce (Dean & Ghemawat, 2004), Hadoop YARN (Apache Hadoop YARN, 2013), Apache Mahout (Apache Mahout Software Foundation, 2012), Apache Spark (Apache Spark, 2013), Microsoft Dryad (Isard. et. al., 2007), Google Pregel (Grzegorz, et. al., 2010) and GraphLab (Low, et. al., 2010).

While these approaches are different in structure, their design principles share similar objectives – mainly reducing the task complexity of implementing parallel processing, fault tolerance and execution optimisation for the developer. In most cases, developers can write sequential programs without worrying about parallelising their code, and it is the compute framework's responsibility to take care of distributing the program among the available processing nodes and executing each instance of the program on the proper segment of the dataset. Hence, the emergence of successful cloud computing projects can mainly be attributed to commoditising parallelism for solving the data management problem. However, the dynamic and distributed nature of cloud computing environments makes data management processes complicated, especially in the case of real-time data processing/database updating (Szalay. et. al., 2006). To cope with today's intensive data workloads, Scalable Database Management Systems (DBMSs) are a critical component of the cloud infrastructure and play an important role in ensuring the smooth transition of

applications from traditional enterprise frameworks to the next generation of cloud computing services. Although distributed data management has been the vision of the database research community for a long time, much of the research has been focused on designing scalable schemes for intensive workloads in traditional large-scale data-processing settings, and there has been little impetus on redesigning the processing architecture to keep up with big data.

The efficiency of the cloud system in dealing with data-intensive applications through parallel processing essentially lies in how data is partitioned and how processing is divided among nodes. As a result, data access schemes are sought to efficiently handle this partitioning automatically and to support the collaboration of nodes in a reliable manner. The majority of current data-parallel frameworks have achieved greater scalability than parallel databases. However, this comes at a cost, as time-consuming analysis and code customisation are required when dealing with complex data interdependencies. Moreover, real-time reliability guarantees remain elusive. Further, existing data management schemes do not work well when data is partitioned among numerous available nodes dynamically. Thus, the question of how to effectively process large-scale datasets is becoming increasingly relevant. Neural network approaches can provide effective tools needed for cloud-based data management. One of the main problems within artificial neural networks (ANN) is that the computational complexity increases substantially with increases in the problem size, and these algorithms often fail to scale up for large and complex datasets (Jain et al., 2000). Further, there is no clear solution to optimally segmenting multidimensional datasets such as images. Addressing these shortcomings for large-scale data analysis will transform the way big data processing is done at present, and it will create a new path for fast data classification.

In this regard, pattern recognition will be an important element in addressing afore-noted shortcomings. However, a number of problems have prevented use of pattern recognition so far, mainly being CPU centric algorithms. To overcome this, Graph Neuron (GN), a scheme that was primarily developed for event detection in WSN (Khan, 2002), is identified as a potential candidate to benefit distributed

frameworks in cloud to process big data. The GN has been tested in pattern recognition applications within different types of distributed environments (Muhammad Amin & Khan, 2008). GN uses a graph-based model for pattern learning and recognition. One of the strengths of this technique is the employment of parallel in-network processing to address scalability issues effectively, which is a primary concern in distributed approaches. This research intends to further extend this scheme to establish an efficient scalable model for Internet-scale pattern recognition and data processing.

The aim of this chapter is to conduct a comprehensive review of current state-of-the-art techniques for large-scale data processing. For this purpose, a number of data-parallel frameworks are discussed in detail, along with their pros and cons. Further, the algorithmic strengths of various neural network approaches for scalable data processing are investigated in regards to the effectiveness of one-shot learning-based parallelism provisioned via graph neuron scheme. The objectives of this chapter are as follows:

i. Conduct a comprehensive review of current data-parallel frameworks and machine learning schemes that deal with Internet-scale data and discuss how neural network approaches can open a new pathway for accessing data in highly distributed environments.
ii. Conduct a detailed review of scalable data-processing requirements and the shortcomings of existing techniques in the literature.
iii. Discuss single-cycle learning and in-network processing for removing the main hurdles towards providing the scalable partitioning and distribution of cloud data.

2.1 Definition and Characteristics of Big Data

There are many discussions on the topic of large-scale data processing – both within industry and academia – as well as the definition of '*big data*' and how the term should be used. In a well-executed commercial study entitled '*Big data: The next frontier for innovation, competition, and productivity*', the McKinsey Global Institute (MGI) defined the term '*big data*' as follows (Manyika, et. al., 2011):

> Big data refers to datasets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyse. This definition is intentionally subjective and incorporates a moving definition of how big a dataset needs to be in order to be considered big data.

In this definition, the MGI claims that there is no certain volume threshold to classify data as '*big*'; rather, it depends on the context. However, the definition uses size or volume of data as the only criterion. In fact, the usage of the term '*big data*' can be misleading in the sense that it mainly highlights the volume criterion. The question of how to deal with large datasets is an ongoing topic of discussion in the database community, and it led to the invention of parallel database systems with '*shared-nothing*' architectures (DeWitt & Gray, 1992). As a result, there should be more to the definition than just size and volume, and most publications elaborate further on this definition. One of these definitions is offered in IDC's '*The Digital Universe*' (Gantz & Reinsel, 2012):

> IDC defines Big Data technologies as a new generation of technologies and architectures, designed to economically extract value from very large volumes of a wide variety of data by enabling high-velocity capture, discovery, and/or analysis. There are three main characteristics of Big Data: the data itself, the analytics of the data, and the presentation of the results of the analytics.

This definition relies on the 3Vs model suggested by Doug Laney in 2001 (Laney, 2001). Instead of using the term '*big data*', Laney predicted that data management would become more important and difficult over time. This led him to identify 3Vs – *data volume, data velocity* and *data variety* – as the major challenges facing data management. Data volume refers to the size of the data, data velocity defines the speed at which new data is introduced to the system, and variety confirms that data can be extracted from different sources and can be unstructured or semi-structured. Later, a 5Vs model was suggested in the literature to capture variability (inconsistency of the data set) and veracity (the quality of captured data) as well

(Hilbert, 2015). However, when the discussion about big data started, different sources decided to adopt the 3Vs definition of big data to highlight the fact that any solution offered should effectively tackle all three to be successful (Russom, 2011). Overall, the 3Vs model of big data seems to be well accepted within both the industry and academia. Moreover, it is helpful when describing characteristics that can be exploited to derive the requirements for the relevant technologies. Therefore, the 3V model is used as a guiding definition for this thesis. The following section details the characteristics which are of fundamental nature and thus more relevant to our approach.

2.1.1 Data Volume

Dealing with voluminous data is the first and most important challenge. However, there is no obvious or concrete point at which data should be considered '*big*'. This is similar to a moving target that moves with a rapid pace over time as computing power increases. While a few hundred terabytes were considered big data almost 10 years ago, petabyte datasets are now considered big, and the trend is shifting towards exabyte and even zettabyte data volumes (Gantz & Reinsel, 2012). In addition, more data results in better outcomes, especially in the case of complex analytic tasks. Halevy et al. (Halevy, et. al., 2009) stated that for big data challenges that involve machine learning and statistical approaches, generating larger datasets is the preferred method over designing sophisticated models and schemes. The authors referred to this as '*the unreasonable effectiveness of data'*, which means that, for machine-learning-specific tasks, larger training sets of freely available, even noisy data typically provide a better result than smaller training sets of carefully cleaned data with complicated schemes.

2.1.2 Data Velocity

Velocity refers to the speed of data, which can be either the rate of new data coming into the system or the existing data being updated (Chen, et. al., 2013). Agrawal et al.

(2012) name this the '*acquisition rate challenge*'. Data velocity can also refer to the time it takes to conduct analysis while data is still receiving new feeds and updates, called the '*timeliness challenge*'. These two challenges are two separate issues in nature; they typically occur at the same time, but they do not necessarily need to (Agrawal, et. al., 2012). Tim Kraska referred to the first challenge as '*big throughput*' (Kraska, 2013). In most cases, the workload is transactional, and the main task is to receive, filter, store and process fast and continuously incoming data in an effective way. Stonebraker et. al., (2013) also claimed that traditional relational database management approaches are not suitable for these types of processing tasks because they involve significant overhead due to excessive locking, logging and buffer pool control for multi-threaded operations. The main challenge here is to maintain a somewhat consistent and persistent state while handling a large number of typically small write operations. A possible solution to this problem is to conduct some pre-processing and filter redundant data to make the process more manageable. However, this filtering stage requires an intelligent mechanism to dismiss unnecessary information without losing important data, and it comes at a cost because pre-processing consumes time and resources to perform the task. Further, it is not always feasible to filter data due to its nature and properties. Another requirement is to extract and keep metadata along with the streaming data so that data lineage can be managed, thereby enabling us to track which data should be kept and how they should be measured (Agrawal, et. al., 2012).

2.1.3 Data Variety

One reason why big data has gained so much attention is that data from diverse sources can provide significant value when properly aggregated and integrated for analytics. Data variety refers to a general diversity of data sources, including both excessive data gathered from various sources and considerable structural differences among those sources. On a higher level, this leads to the requirement of processing structured data, semi-structured data and unstructured data (Kaisler, et. al., 2013). However, on a lower level, while data sources can be structured or semi-structured,

they can still be heterogeneous and their schema may not be compatible, resulting in inconsistent semantics (Helland, 2011). Integrating and processing this collection of structurally different data can introduce several challenges. One of these challenges relates to the actual mechanism that should be used to store and manage this type of data in a database-type platform. For this purpose, relational database management systems (RDBMSs) may not be a good fit to cater for all types and formats of data. For example, Stonebraker et. al., (2013) claimed that RDBMSs offer a poor choice for array or graph data, where array data is typically important in scientific research and graph data is of high importance for social networks analytics (Stonebraker, et. al., 2013). Another important challenge when dealing with semi-structured or unstructured data is that before these data types can be of much use for analysis, some type of structure should be imposed on them to enable the researchers to extract entities, relationships and other relevant information. There are some existing machine learning, information retrieval, natural language processing and data-mining techniques available for this purpose; however, due to the variety of unstructured data sources (e.g., images and videos), new data extraction schemes should be developed to provide more feasible and effective ways of processing, as many of the existing processing tools fail when facing unstructured data (Agrawal, et. al., 2012).

2.2 Neural Network Schemes for Big Data Processing

Neural networks can be defined as interconnected parallel-computing networks of a massive number of processing nodes known as neurons (Jain, et. al., 2000). One of the main benefits of using neural network techniques for data processing is that they allow the system to learn from the data and progressively adapt to the nature of the data. This adaptive feature provides a promising tool for scalable Internet-scale recognition. However, a number of issues need to be overcome in relation to their implementation and deployment. This section provides an overview of some of the most well-known neural network schemes for pattern recognition. The techniques discussed here are widely used across many applications of pattern recognition; however, the focus here is on the scalability and adaptability of these approaches for

large-scale pattern matching tasks. An evaluation of the neural network and machine learning algorithms discussed in the literature suggests that these schemes can offer promising tools for deterministic pattern recognition (Vivanco, et. al., 2005). Moreover, they can let the system to learn from data and adopt itself to its conditions. However, the relative computational complexity of current schemes places a heavy burden on their widespread use for large-scale pattern processing. This is mainly due to their excessive and highly iterative training procedures.

2.2.1 Feed-forward Neural Network

The feed-forward neural network scheme provides a well-defined approach for building auto-associations between an input layer and an output layer in a large domain of computational problems such as pattern recognition (Nadal, 1989) (see Figure 2.1). However, its classification process faces some limitations relating to its implementation. These implantation barriers include: its relative sensitivity towards the training parameters, training speed, non-linear classification function, overtraining sensitivity and regularisation requirements (Jain et al., 2000). As described by Kalos (2005), it is difficult to incorporate feed-forward neural networks for mainstream applications because they are highly specialised. Moreover, he highlighted a few implementation-related issues for such networks, including the difficult process of interpreting the results, as well as the trial-and-error approach needed to build their architecture. The cumbersome nature of predicting the results is due to the fact that the intermediate results obtained from the network should first be cross-validated before they can produce the best output. In addition, the algorithm design is a complex process because of the instability that is present in the number of hidden layers required for a given dataset. Given their intensive computational operations, they require excessive training procedures to achieve optimum recognition accuracy for a given dataset. As a result, feed-forward neural networks suffer from scalability and adaptability issues. Although their design permits parallel processing to improve upon scalability, the complex computational operations prevent them from being suitable candidates for scalable pattern recognition schemes.

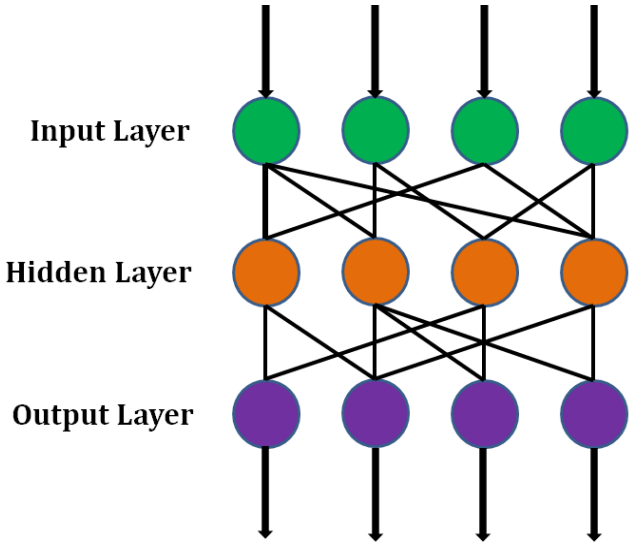

Figure 2.1: Feed-forward neural network model
(Feed-forward neural network, 2011)

2.2.2 Recurrent Neural Networks (RNN)

A recurrent neural network (RNN), also referred to as a feedback neural network, is a multi-layered network structure where the input receives feedback from an RNN output to increase recognition accuracy (Duda, et. al., 2001) (see Figure 2.2).

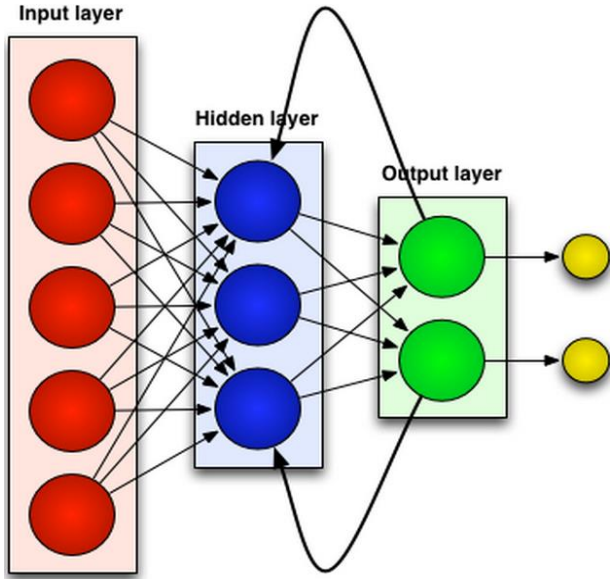

Figure 2.2: RNN with feedback link (Recurrent Neural Networks in Ruby, 2012)

RNNs can be categorised into two main categories, namely standard and relaxation RNNs (Connor, et. al., 1994). Standard RNNs follow the same principles as standard neural networks and incorporate feedback links within their design. Relaxation RNNs continuously implement learning and recognition stages until feedback inputs reach a predefined threshold. While RNNs are simple and powerful schemes, they may get stuck in calculating local minima during gradient descent, achieving sub-optimal results (Bengio, et. al., 1994). This limits their applicability to large-scale pattern recognition problems requiring optimal results in minimal time.

2.2.3 Hopfield Network

Hopfield and Tank (1985) introduced the Hopfield network as a type of supervised neural network, offering an alternative solution to complex computational tasks such as combinatorial optimisation. The Hopfield network has also provided an efficient approach for solving the Travelling Salesman Problem (TSP) and other pattern recognition tasks (see Figure 2.3).

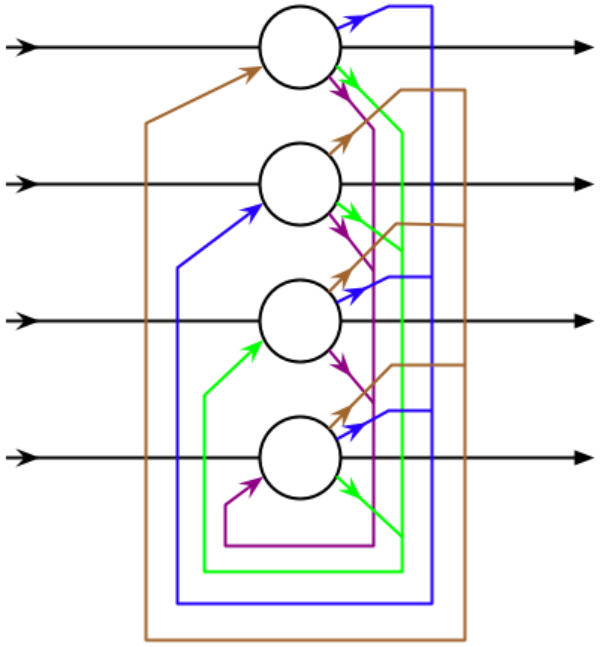

Figure 2.3: A Hopfield network with four nodes (Hopfield network, 2012)

Hopfield networks can be divided into two main categories: Discrete Hopfield Network (DHN) and Continuous Hopfield Network (CHN) (Kim, et. al., 1992). DHN is a fast-processing stochastic approach that is simple to implement. As the scheme uses binary values to represent the states of neurons, it does not produce accurate results. Hence, DHN is not capable of providing sufficient levels of accuracy for pattern recognition applications. Conversely, CHN uses a differential equation scheme to achieve a near-optimal result. This approach requires more time to produce acceptable results, which in turn places a practical burden on its implementation. As a result, CHN cannot offer an efficient solution to pattern recognition applications that require fast processing. Moreover, the Hopfield network suffers from a convergence problem, which results in producing less-than-optimal solutions (Li et. al., 2005). It also does not scale well due to scalability issues associated with the storage of biased patterns (Lowe, 1999). These prevent the Hopfield approach from being a suitable candidate for solving large-scale pattern recognition problems, regardless of its ability to perform parallel recognition (Wilson, 2009).

2.2.4 Self-Organizing Maps

A self-organising map (SOM), also referred to as a Kohonen map, is an unsupervised neural network approach used for performing pattern clustering and classification (Kohonen, 2000). Kohonen maps are formed based on the competitive learning algorithm (Rumelhart & Zipser, 1988). As part of the classification process, SOM offers a mapping of high-dimensional data space to lower-dimensional data space using dimension reduction (see Figure 2.4). A peculiarity of this technique is that the neurons are well-placed and well-represented in the form of a geometrical dimension (Giorgetti, et. al, 2007). The SOM scheme also exhibits a high level of adaptability, as it can represent various types of data using a single form of representation. Kohonen networks have been extensively used in a wide range of applications including, geo-informatics (Zaremba, et. al., 2000), bioinformatics (Wang, et. al., 2007), finance (Blazejewski & Coggins, 2004), information retrieval (Lin, et. al., 1991) and wireless technology (Giorgetti et. al., 2007).

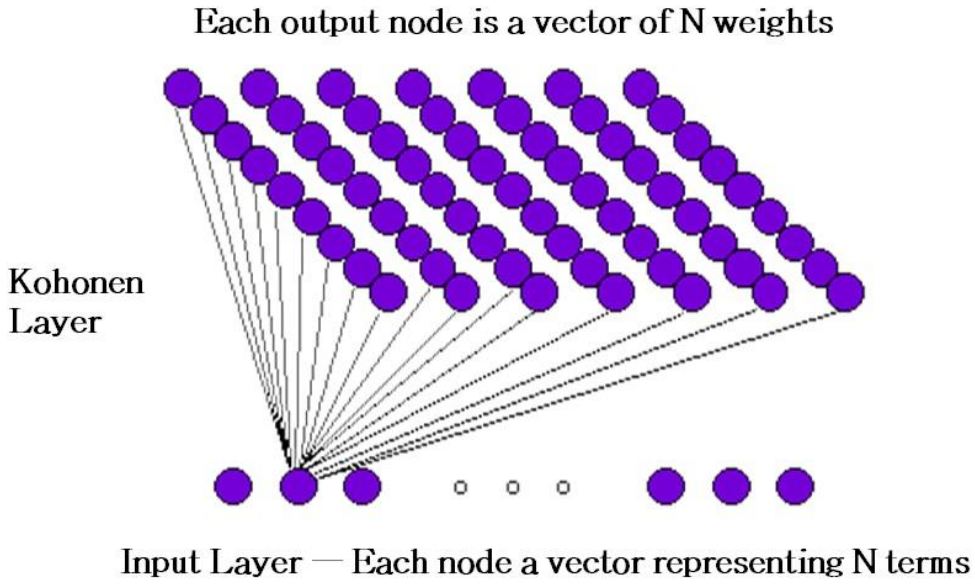

Figure 2.4: Schematic view of SOM Network (Schlegel, 2011)

One of the main drawbacks of the standard SOM approach is its high computational complexity and its inability to scale efficiently with increases in the map size. While SOM is a well-established method for clustering high-dimensional data, its training and classification process requires numerous iterations to be performed on each neuron, which makes the scheme computationally costly when dealing with high-dimensional data. To address this shortcoming, various dimension reduction techniques are adopted at the cost of increased processing time. Moreover, determining the learning rate of SOM-based schemes to estimate the efficiency of the approach is a non-trivial task (Cheung & Law, 2007). These limitations restrict the applicability of SOM-based techniques in solving large-scale pattern recognition problems.

2.2.5 Support Vector Machine (SVM)

Support vector machines (SVMs) have gained a lot of attention recently due to their ability to perform effective data classification and regression tasks (Casali, et. al., 2006). The scheme implements classification by creating a mapping of input vectors to a high-dimensional feature space in a non-linear shape and form, and then forming

a linear decision surface by building one or more hyper-planes to enable class separation (Cortes & Vapnik, 1995). The classification process will be performed by determining the optimal separating hyper-plane. This approach is illustrated in Figure 2.5, where the problem is simply defined as finding an optimal solution to a problem. In this example, the optimal solution is to find the best line passing as far as possible from all of the points. As a result, the SVM target here is to determine the hyper-plane that provides the largest minimum distance to the training examples. This optimal separating hyper-plane maximises the margin of the training data. SVM was originally deployed and tested for binary problems because it offers unique solutions along with good generalisation properties of the solution (Mavroforakis & Theodoridis, 2006). However, the technique suffers from a slow test phase compared to other learning schemes presented in the literature (Huang, et. al., 2005). Moreover, the computational costs can become excessive with an increased number of support vectors (SVs) (Dong, et. al., 2005). In addition to that, SVM does not scale effectively, as the SVM training kernel matrix size increases quadratically with the size of the dataset; this results in a dramatic increase for big datasets, making it impractical for large-scale pattern processing problems (Nguyen & Ho, 2006).

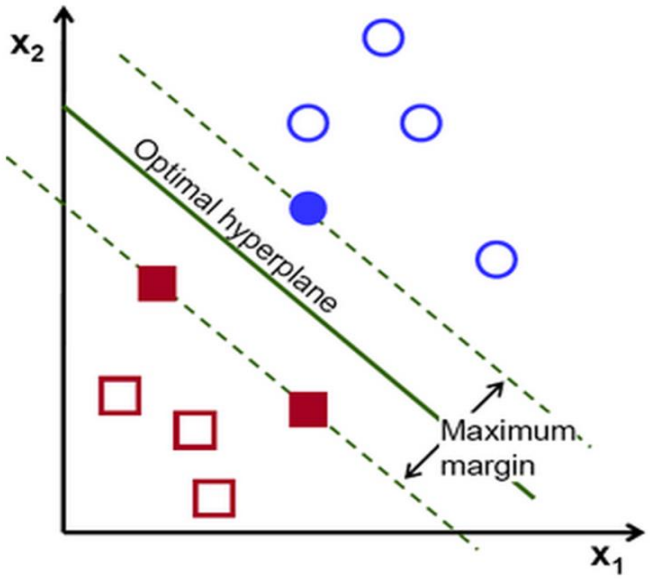

Figure 2.5: SVM classification process (Introduction to SVM, 2012)

2.3 Neural Network/Machine Learning Requirements for Large-scale Pattern Recognition and Data Processing

Artificial neural network techniques have been extensively used for different pattern matching and classification tasks. For instance, Jiang et. al. (2010), introduced a back propagation neural network scheme to perform the processing of high-resolution sensory images to identify roads. In another example, Khoa et. al. (2006) presented a stock price forecasting scheme utilising the BPNN approach. Previously, ANN schemes were mainly used to deal with small-size datasets. However, with the emergence of large-scale data-processing applications, their potential use for big data-processing tasks can be restricted in their current shape and form as they become computationally intensive, with large memory requirements when applied to large-scale datasets. One of the main benefits of using neural network techniques for data processing is that they enable the system to learn from data and progressively adapt to that nature of data. This adaptive feature provides a promising tool for scalable Internet-scale recognition (Vivanco, et. al., 2005). However, a number of issues need to be overcome in relation to their implementation and deployment. Wang et al. (2014) claimed that ANN schemes can be considered one of the major tools for large-scale data analysis if the fundamental challenges of dealing with big data can be faced effectively within the two phases, namely the *training phase* and *operation phase*. As an example, back-propagation neural network (BPNN) is one of the most widely used ANN techniques that can potentially approximate any sort of continuous non-linear function by arbitrary precision if enough neurons are available for computation (Hagan, et.al., 1996). In most cases, BPNN uses the back-propagation algorithm as part of the training stage, which can be time-consuming when dealing with large volume of training data (Gu, et. al., 2013). To take advantage of the potentials of neural networks for big data processing, an alternative is to use parallel processing approaches to speed up computational work – for example, the use of Message Passing Interface (MPI) (Kumar, et. al., 2002). Long and Gupta (2008) introduced a parallel ANN with an MPI technique for providing computational parallelism. While

MPI was originally designed for data-intensive applications, it does not offer great deal of support for fault tolerance. In fact, in many fault-occurring cases, MPI processes should be re-initiated, which makes them unsuitable for big data processing scenarios where failures can occur at any time in the system (Long & Gupta, 2008). Furthermore, many of the existing neural network techniques suffer from the over-fitting problem, where a small-sized training dataset cannot effectively represent the actual characteristics of the large dataset.

A comprehensive study of the existing pattern recognition techniques in the literature shows that they are capable of offering acceptable levels of scalability and adaptability, but at the cost of introducing excessive computational costs with increased complexity. In this regard, some attempts have been made in the literature to offer faster neural network computations by either trying to better select the initial weights (Nguyen & Widrow, 2010) or better control the learning parameters (Kanan & Khanian, 2012). In recent years, many researchers have started focusing on utilising the potential embedded in parallel processing techniques and distributed computing approaches to come up with scalable methods to work around the computational bottlenecks of existing large neural network schemes (Ikram, et. al., 2013) (Huqqani, et. al., 2014). Gu et. al., (2013) introduced a computationally fast parallel neural network approach utilising some in-memory data processing at the cost of providing less accurate results. They achieved a faster scheme through parallelism by splitting the training data into data chunks and processing them in parallel. In another work, Liu. et. al., (2010) demonstrated the application of a MapReduce-based BPNN in classifying an excessive amount of mobile data. In their proposed parallel neural network classification approach, they utilised AdaBoosting to compensate for the accuracy shortfall. Despite achieving higher accuracy rates, their approach suffers from computationally expensive operations within both training and classification phases. In addition, the AdaBoosting approach has the risk of increasing the weight of potential poor classified instances, resulting in less accurate results (Freund, et. al., 1998).

2.4 Parallel Data-processing Frameworks

To simplify the development of distributed applications on top of such highly distributed architectures, customised data-processing frameworks are developed and deployed.

2.4.1 Hadoop and Hadoop Distributed File System

Hadoop has been extensively used for large-scale data processing in the clouds (Chen, et. al., 2008), and it is widely utilised by the industry's major web players – Google, Yahoo, Microsoft and Facebook – as the platform to enable the cloud. As in the cloud, the computing unit is mostly VM (virtual machine) based, such that Amazon Elastic Cloud Computing (Amazon Elastic Cloud Computing, 2011) and GoGrid (GoGrid Cloud Hosting, 2011) offer VM-based computing infrastructure as a service. It is therefore possible to use cloud data-processing schemes in a virtualised data centre. Although poor functionality and significant load imbalances exist, VMs can still be employed to assist with utilising the system resources and provide better management and control while improving reliability (Figueiredo, et. al., 2003). Existing cloud systems often rely on Hadoop Distributed File Systems (HDFS) and the parallel scanning procedure as their underlying platform to manage data. HDFS is a kind of distributed file system that offers high throughput access to application data, and it is built and developed to function on commodity hardware (Apache Hadoop, 2010).

2.4.1.1 HDFS Features

HDFS has many common characteristics with other distributed file systems, but its high level of fault tolerance makes it an efficient approach to support the development of Hadoop on large clusters of machines, providing high throughput access to deal with Internet-scale datasets (Apache Hadoop, 2010). In summary, the features of HDFS can be categorised as follows:

- *Very large datasets*: HDFS enables the processing of massive amounts of data in the order of petabyte scales on distributed file systems.

- *Streaming data access*: The HDFS process follows the model of write once and then read many times. This approach not only minimises data coherency issues, but it also over-simplifies high throughput data access, making it efficient and suitable for MapReduce applications, which is discussed later in this chapter.
- *Commodity hardware*: Hadoop can run on inexpensive machines with noticeably low power consumption. This brings the power of fault tolerance to the scheme, where basic failures can be recovered and compensated with minimum or no effects on the total functionality.
- *Data reliability*: As data is stored on multiple nodes and racks in HDFS architecture, data will be easily accessible in case of sudden failures. In fact, the placement of replicas is one of the key differentiators between HDFS and other distributed file system mechanisms.
- *Portability*: HDFS is designed so that it can be easily moved from one platform to another, thereby facilitating its use as a platform of choice for many applications.

2.4.1.2 HDFS Architecture

HDFS consists of one *NameNode* and a number of *DataNodes*, and it has *master/slave* architecture. The master server, referred to as the *NameNode*, divides files into blocks and distributes them among the cluster members with replication to cater for fault tolerance. It also keeps all metadata about stored files and arranges the system namespace. The slaves, referred to as *DataNodes*, are the actual repository of the data blocks; they respond to read/write requests from clients and distribute replication tasks as instructed by the *NameNode*. In response to a request, the relevant data is retrieved from the HDFS and sent to a set of pre-allocated compute nodes to implement parallel scanning. To achieve its goal, HDFS uses the *JobTracker* and *TaskTracker* functions. *JobTracker* is responsible for scheduling and assigning the relevant tasks to *TaskTrackers*, while *TaskTrackers* are only responsible for accomplishing the jobs they are assigned to. Upon completion of the job, *TaskTrackers* notifies *JobTracker* about the result of the work (success/failure), and in case of failure, the *JobTracker* reschedules the failed operations (see Figure 2.6).

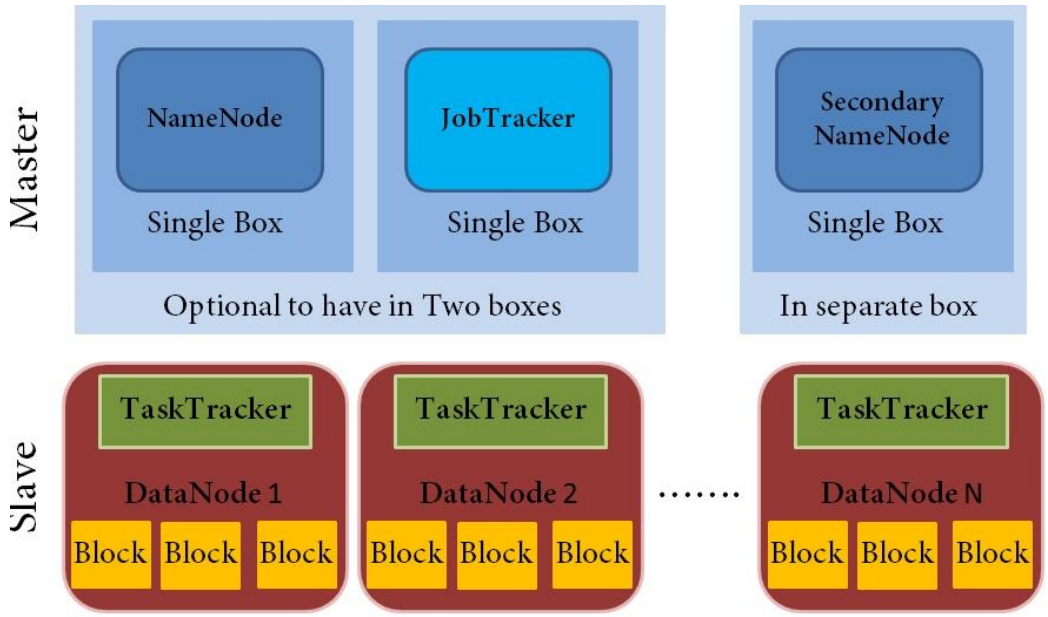

Figure 2.6: HDFS with multiple data nodes for storing data
(Apache Hadoop, 2010)

With this HDFS architecture, we may face some technical challenges, including:

- Adapting ourselves to writing application codes in a new programming paradigm.
- *NameNodes* do not scale effectively. In fact, HDFS runs a secondary *NameNode* for faster recovery, but the secondary *NameNode* plays the role of a log server rather than a failover server. In case of failure, the primary *NameNode* will eventually need to be restarted, and some of the actions should be replayed.
- Similar to *TaskTrackers* with regards to their immediate effect on job performance, the poor performance of the *NameNode* or particular set of *DataNodes* can have a significant deteriorating effect on the overall functionality of the whole data cluster.
- Based on experimental results, *NameNode* can be a bottleneck for linear scaling. In fact, a 10,000-node HDFS cluster with a single *NameNode* is expected to handle a workload of 100,000 readers (memory-only operation); however, practical experiments have shown that even 10,000 writers (bounded by the local hard drive performance) can produce a sufficient workload to saturate the *NameNode* and degrade the overall performance (NameNode Performance, 2008).

2.4.2 MapReduce

MapReduce is a parallel processing framework for the distributed processing of large datasets among compute cluster nodes (Dean & Ghemawat, 2004), and the Hadoop version of it relies on HDFS as its underlying platform (Apache Hadoop MapReduce, 2011). The MapReduce and HDFS frameworks both perform processing on the same set of nodes. That is, the majority of computational tasks are performed where the data already resides (data locality). This approach significantly speeds up processing because it is computationally much cheaper to move the computations and not the data. In the MapReduce data-processing approach, all processing operations are expressed using two main primitives – (a) a map function to accept a key-value pair, perform some computations and generate a set of intermediate key-value pairs as output, and (b) a reduce function to aggregate all intermediate results associated with the same intermediate key, perform some computations and emit the final output (see Figure 2.7). These simplified functions permit users to build and deploy parallel data-processing jobs without the need to explicitly coordinate parallel sub-tasks with distributed file storage. As a result, the MapReduce abstraction can vastly improve user productivity and experience (Apache Hadoop MapReduce, 2011).

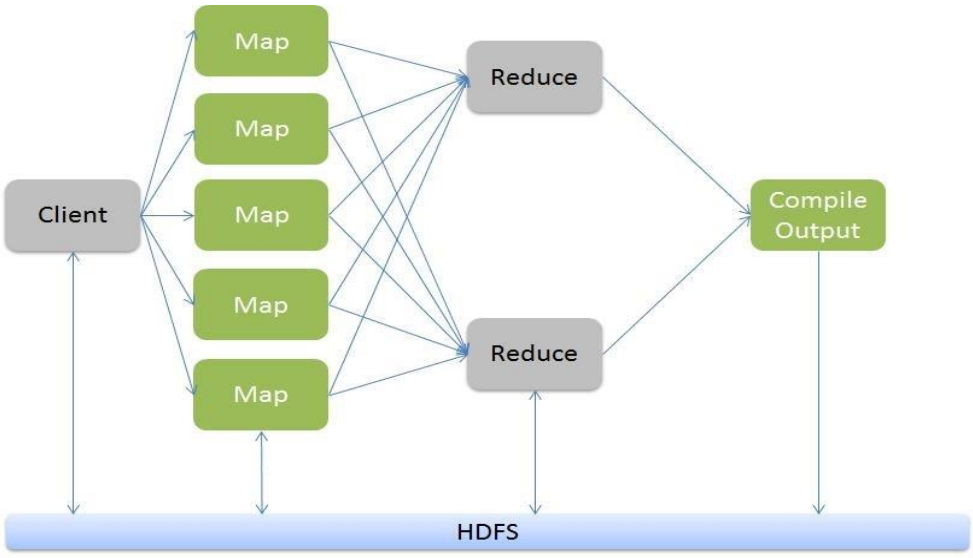

Figure 2.7: MapReduce data flow structure (OpenSource Forum, 2011)

MapReduce applications usually process large volumes of cloud data. This requires two critical data movements: gathering the input data for the map phase, and reorganising and redistributing the output of the map tasks as input for the reduce phase. The MapReduce data movement approach heavily depends on the parallelisation strategy used. It should be noted that maximum parallelism of the (parallel) map phase is bounded by the number of input pairs while the parallelism in the reduction phase is also restricted by the number of various output keys of the map phase, which in turn highly depends on the deployed algorithms and the nature of the input data. Moreover, the approach suffers from certain key limitations:

- In the MapReduce type of query processing, the map tasks are assumed to be fully independent. However, when applied to massive relational or object-oriented data, large records or objects resulting from aggregation and analytics are often broken into parts and distributed, thereby creating dependencies and requiring trade-offs between redundancy (for speed), coherence (for integrity under frequent updates) and compromises to parallel schedulability, as they break assumptions of mutual independence.
- In practice, MapReduce functions are implemented imperatively and produce an excessive number of intermediary entities between the map and reduce stages (e.g., in the form of intermediate files). In many cases, these intermediate files must be first sorted and indexed before they can be entered as input to the reduce function. This system's extensive sort and redistribution tasks incur significantly high processing and communication costs, and the system is either fundamentally non-scalable or requires fine-tuned architecture-aware access mechanisms.
- While assisting designers and developers with few predefined architectural patterns for many applications (Gamma, et. al., 1995), the MapReduce data flow model is also rigid, limits variation and hence increases the complexities of dealing with errors, fault tolerance, performance and other end-to-end non-functional issues. Some exploratory research implementations use key-value pairs with distributed '*spaces'*, such as JavaSpaces or other derivatives of Linda tuple spaces (Gelernter & Carriero, 1985), to simplify data sharing and conceptually

separate shared data from computational tasks. However, this simplification comes with significant efficiency loss and exacerbates the uncertainty of predicting reliability and real-time behaviour.

Hence, in practice, MapReduce cannot automatically scale up for many applications and datasets.

2.4.3 Hadoop YARN

In its original design, MapReduce suffers from various issues, including scalability concerns, as the maximum practical cluster size could achieve about 4000 nodes with coarse synchronisation processes for the *JobTracker* (Yahoo Developer Network, 2008). In addition, the partitioning of resources into map and reduce slots results in inefficient utilisation. Moreover, the rigid design of the MapReduce framework, along with its lack of support for alternate paradigms, makes it impractical for some applications, in particular those that are iterative in nature. As a result, the MapReduce scheme has been significantly reinvented in Hadoop-0.23, and the current version is referred to as MapReduce 2.0 (MRv2) or YARN. YARN was originally introduced by Apache as a newly designed resource manager, but in its current form it is considered the next generation compute and resource management framework for big data applications (Apache Hadoop YARN, 2013) (see Figure 2.8).

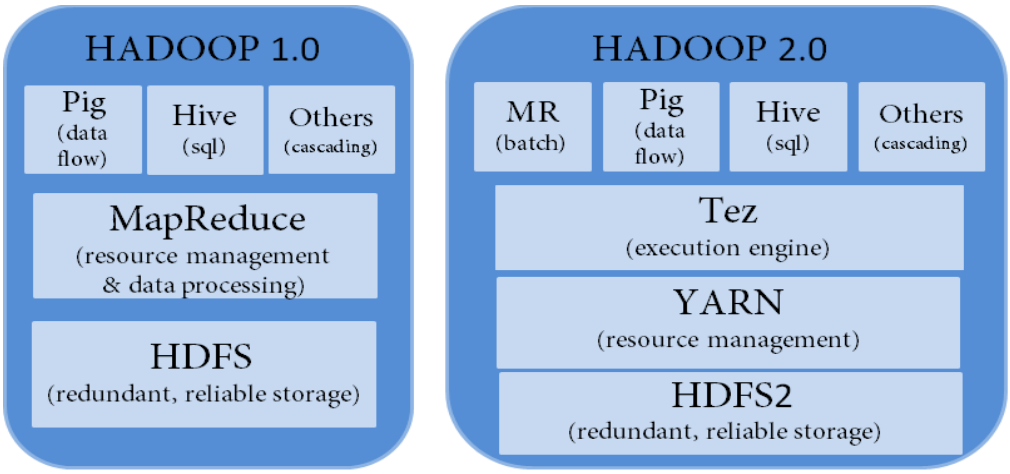

Figure 2.8: Apache Hadoop 2.0 (Apache Hadoop YARN, 2013)

YARN has a master/slave architecture in which the *JobTracker*'s two main functionalities are executed within separate daemons, namely resource management and job scheduling/monitoring. YARN introduces a global *ResourceManager* (RM) and an *ApplicationMaster* for each application. In simple terms, the YARN data framework consists of the global *ResourceManager* and a per node slave called *NodeManager* (NM). For each application, the *ApplicationMaster* component looks after the implementation and monitoring of the dedicated tasks after negotiating required resources with the *ResourceManager* (see Figure 2.9).

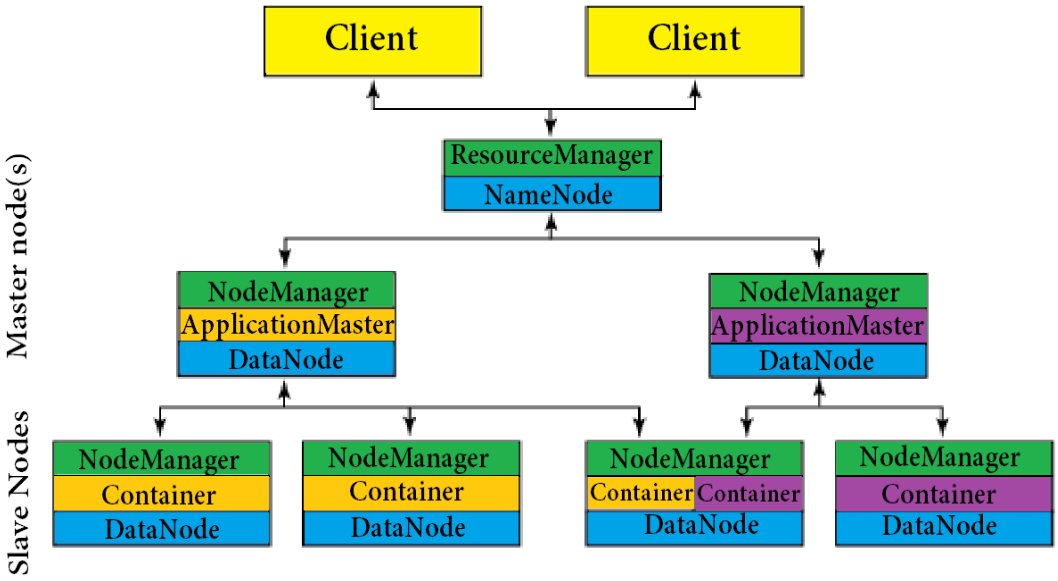

Figure 2.9: YARN, Apache next generation MapReduce
(Apache Hadoop YARN, 2013)

This new computational framework design in YARN will eliminate MapReduce bottlenecks where pre-YARN jobs were forced to be executed through a single *JobTracker* daemon (as batch processes), thereby limiting MapReduce scalability while reducing processing speed (Apache Hadoop YARN, 2013). The *ResourceManager* in YARN plays the role of global coordination, which in turn creates a single point of failure in the system. Any planned (upgrades) or unplanned (node crashes) outages for *ResourceManager* can render YARN unavailable.

2.4.4 Apache Mahout

Apache Mahout was initially started as a sub-component of the Apache Lucene project in 2008 (Apache Lucene Core, 2008) and became a big project by itself in 2010. The goal of Apache Mahout is to build Apache-licensed machine learning libraries that can scale up with big data. Machine learning in the context of big data is simply to create computational intelligence using example data or by observing past experiences (Alpaydin, 2004). Building applications that intelligently learn from input data is of high interest, and such applications require machine learning schemes to achieve this. Apache Mahout is designed to address this requirement by enabling users to employ highly scalable machine learning approaches on multicore, such as collaborative filtering (CF) (Resnick & Varian, 1997) and random forest decision-tree-based classifiers (Breiman, 2001). The goal of Apache Mahout for big data processing is to deliver a scheme that is as fast and efficient as possible given the intrinsic design of the algorithm. It must be capable of addressing the major shortcomings of many machine learning approaches, which do not scale effectively to massive machine clusters. Given its multicore implementation model, Mahout uses Hadoop's HDFS and MapReduce frameworks. Mahout introduces a number of techniques – many of which are still in the development phase. However, Mahout's three core themes are recommender engines, clustering and classification (Apache Mahout Software Foundation, 2012) (see Figure 2.10). The recommender system is probably the most noticeable machine learning scheme in use today. It recommends services and products based on historical actions and preferences. For this purpose, Mahout uses an extensive framework for collaborative filtering where top-level implementation packages define the Mahout interfaces to the following key abstractions: data model, user similarity, item similarity, user neighbourhood (nearest N-user neighbourhood or threshold-user neighbourhood) and recommender abstraction (Ricci, et. al., 2011). A clustering algorithm is an unsupervised machine learning technique that facilitates grouping a large number of entities together that share a similarity. This clustering approach helps in discovering patterns within complex unstructured datasets and assists in forming a hierarchy of datasets to

discover relationships. To perform clustering, Mahout calculates small intra-cluster distances by determining local and global minima and then calculates large inter-cluster distances using Mahout's Canopy clustering MapReduce algorithm to compute initial cluster centroids (Apache Mahout Canopy Clustering, 2012).

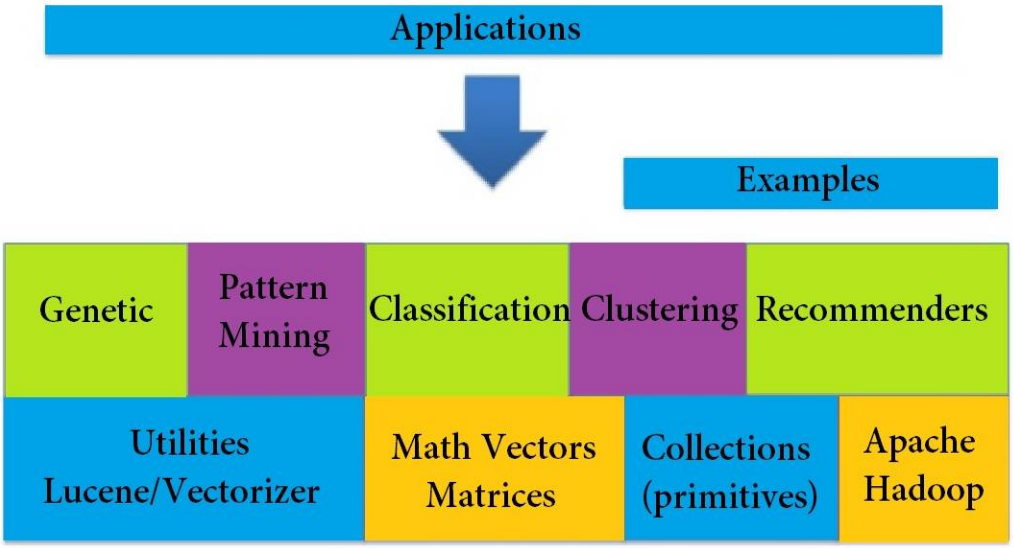

Figure 2.10: Apache Mahout (Apache Mahout Software Foundation, 2012)

In contrast, classification algorithms are supervised machine learning techniques that aim to determine whether a certain entity belongs to a group or category, or whether it does or does not have some attribute. To achieve this, Mahout initially performs some training on labelled data and then runs a classifier algorithm against unlabelled data to implement the classification. In this regard, Mahout can choose from a large number of classification algorithms, such as the maximum entropy classifier (Ratnaparkhi, 1997), the naïve Bayes classifier (Friedman, et. al., 1997), decision trees (Kamath & Musick, 1998), SVMs (Cortes & Vapnik, 1995), KNN algorithms (Dasarathy, 2002) and the perceptron scheme (Freund, 1999). While Mahout provides an effective approach for implementing machine learning techniques in a scalable fashion over multicore architectures, it suffers from certain key limitations (Apache Mahout Software Foundation, 2012):

- Mahout does not always scale as well as expected. Experimental results show that different data sizes with different algorithms can occasionally produce inefficient results, sometimes minus percentage speed-ups.
- Experiments show that using Mahout for recommender systems can be inefficient in terms of high memory and resource consumption.
- Its performance may not be comparable with specifically optimised schemes for certain datasets, but it still offers a great speed-up.

Moreover, in its current form, Mahout only works with a limited range of standard machine learning approaches, and it should be further optimised for specific algorithms.

2.4.5 Google Pregel

Pregel is an efficient, scalable and fault-tolerant framework that enables large-scale graph processing using a simple code. It is capable of performing computations over large graphs in a very fast fashion while hiding relevant distribution details behind an abstract application programming interface (API) (Malewicz, et. al., 2010). Its architecture is inspired and developed by the Bulk Synchronous Parallel (BSP) model (Valiant, 1990) that empowers the programmer to find parallel-computing solutions for a specific problem without having to know how communication and memory allocations are performed in a distributed setup. To minimise communication overhead, Pregel also tries to preserve data locality by moving the computations to where the data reside. The input-directed graph is loaded once during the start-up phase, and all following computations are performed in-memory. Pregel takes a directed graph as an input in which each vertex is uniquely identified with a string vertex identifier. Pregel has a master/slave model architecture where each worker receives a subset of the directed graph's vertices to work with. Each vertex has an arbitrary value that can be get/set, and it also maintains a list of its outgoing edges. The directed edges are linked with their source vertices, and each edge is lined with a user-defined modifiable value and a target vertex identifier. A typical Pregel

implementation starts with the graph input, where the graph is first initialised. A sequence of processing iterations (super steps) is then performed, separated by a global synchronisation checkpoint, and this process continues until the algorithm terminates with an output (see Figure 2.11). As discussed previously, Pregel implements the BSP model when the master initiates an iteration, referred to as a super step. At every super step, workers execute a user-defined function on all of its vertices. Vertices can send/receive messages to other vertices, and those messages will be delivered in the next super step. Within each of the super step iterations, vertices can update their value, make changes to the value of edges or even modify the topology of the graph by adding or removing vertices/edges. Vertices also have the power of '*vote to halt'*. The execution is terminated when all vertices vote to halt and there are no more messages to be delivered. A problem arising from this type of processing is that a vertex program should keep running in the background to send out update messages, but there is a chance that some vertices converge earlier than others, resulting in a waste of CPU resources.

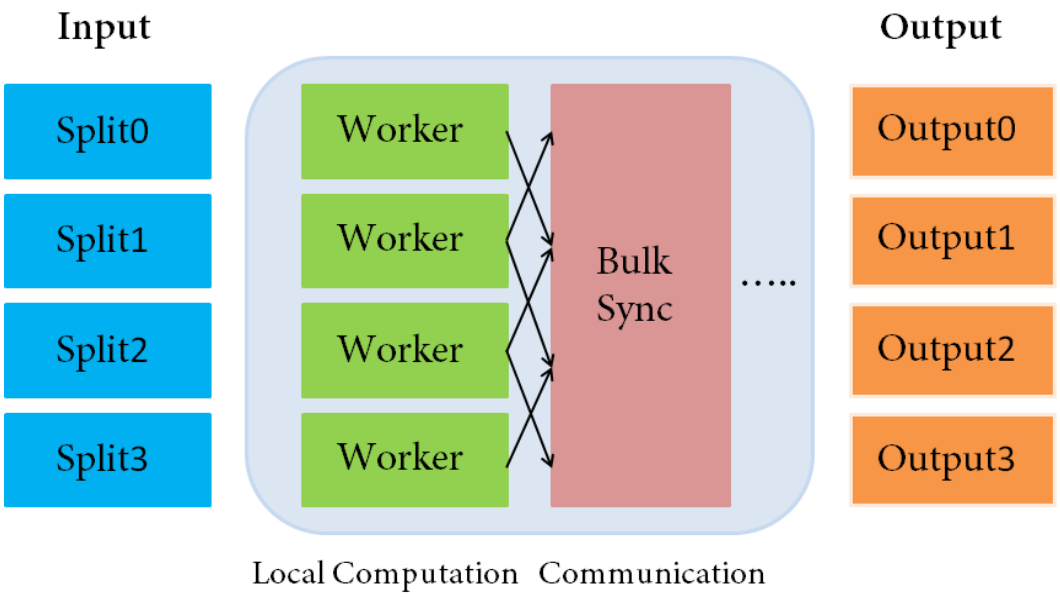

Figure 2.11: Pregel data model (Percolator, Dremel & Pregel, 2012)

In addition, algorithms like Pregel, which follow the BSP computational model, can suffer from the straggler problem, where the transient slowdown of any processing thread can slow down the whole system. In fact, in the BSP approach, tight synchronisation requirements in each iteration can have a significant adverse effect on the overall performance of the scheme. Pregel is a simple parallel processing framework with many opportunities for improvement, and it has led to the invention of several other graph processing approaches, including Apache Giraph (Apache Giraph, 2013), GPS (Salihoglu & Widom, 2013) and Mizan (Khayyat, et. al., 2013).

2.4.6 GraphLab

A team from Carnegie Mellon University started the GraphLab project in 2009 to create a scalable computing framework for large-scale data processing, incorporating the requirements of parallel abstractions tailor-made for machine learning (ML) applications (GraphLab Open Source, 2009). They initially suggested a shared-memory approach referred to as GraphLab 1.0, and they later introduced GraphLab 2.1 for distributed environments. The latest release, GraphLab PowerGraph version 2.2, introduces a new set of capabilities through the use of a new API to increase usability. The GraphLab algorithm was initially developed based on the common computational characteristics of machine learning approaches to provide an effective and scalable framework for building machine learning schemes. GraphLab achieves its goal by targeting common patterns in ML, such as sparse computational dependencies, asynchronous iterative computation along with sequential consistency and prioritised ordering (Low, et. al., 2010).

2.4.6.1 GraphLab 1.0

The GraphLab data structure consists of a directed data graph and a shared data table. The data graph represents both problem-specific sparse computational dependencies and the program state, which can be modified during execution. Each vertex and directed edge in the graph can be allocated an arbitrary block of data by the user.

Further, and to support a globally shared state, a shared data table is established by implementing an associative map between some given keys and arbitrary blocks of data. Computations in GraphLab are conducted using either an update function or through the sync mechanism. The update function is analogous to the map function in MapReduce, where local computations are performed as instructed by the input data graph. A difference here is that the update function in GraphLab can read and update overlapping sets of data (program states) in a controlled manner, and this process can be permitted to occur in a recursive fashion, thereby enabling dynamic iterative computations. GraphLab maintains the prioritised ordering of update functions by using powerful scheduling primitives. The sync mechanism is analogous to the reduce function in MapReduce, and it enables reductions to occur while update functions are in progress. A difference here is that the sync operation in GraphLab, unlike Reduce function in MapReduce, can run concurrently with the Update function (Low, et. al., 2010). GraphLab 1.0 exhibits very effective '*scatter*' capability, where an update to a single vertex can be efficiently communicated across the network to all target machines, thereby minimising the amount of data replication. This is a significantly powerful advantage over Pregel, where excessive data replication is needed to transmit vertex data modifications. In contrast, GraphLab 1.0 suffers from an inefficient '*gather*' capability that does not allow each vertex to effectively '*pre-combine*' its target data. In fact, the update function in GraphLab 1.0 needs full access to the entire scope rather than the aggregated value, which in turn results in excessive unnecessary communications that waste network bandwidth (Low, et. al, 2012). Further, the majority of today's natural graphs have an uneven power-law degree distribution (Faloutsos, et. al., 1999), where most vertices have a limited number of neighbours and a few vertices have many neighbours (e.g., LinkedIn, Facebook, Twitter). GraphLab 1.0 works well on low-degree vertices within small neighbourhoods; however, when dealing with power-law graphs in real-life scenarios, the job of graph partitioning in GraphLab 1.0 becomes very complex, if not impossible, to implement (Gonzalez, et. al., 2012).

2.4.6.2 GraphLab 2.2 (PowerGraph Abstraction)

To address the aforementioned inefficiencies in the design of GraphLab 1.0, the team at Carnegie Mellon University introduced GraphLab 2.2, known as PowerGraph (Gonzalez, et. al., 2012). PowerGraph brings together the best offerings of both GraphLab 1.0 and Pregel (see Figure 2.12). From GraphLab 1.0, PowerGraph inherited the *data graph concept* and *the shared-memory approach* of computation. From Pregel, PowerGraph received *commutative*, *associative* and *gather* concepts as part of its implementation strategy. PowerGraph supports both Pregel's highly distributable bulk synchronous form of computation and GraphLab 1.0's efficient asynchronous computational model. PowerGraph achieves better efficiency by introducing a new way of partitioning power-law graphs. In the new approach, edges in the graph are tied to machines, and high-degree vertices can span machines.

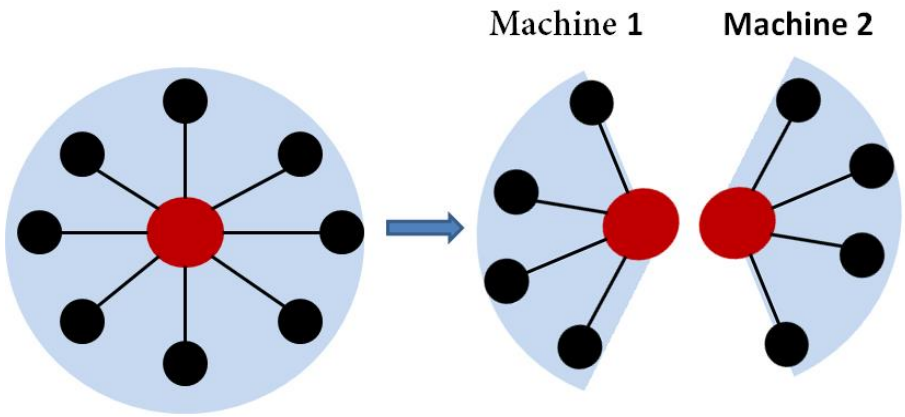

Figure 2.12: PowerGraph solution to power-law graphs
(GraphLab Open Source, 2009)

PowerGraph positions '*gather*' and '*scatter*' phases into the abstraction and directly exploits the '*gather-apply-scatter*' (GAS)' decomposition to factor vertex programs over edges (see Figure 2.13). This approach distributes the computation of a single vertex program over the entire cluster and eliminates the degree dependence of the vertex program (Gonzalez, et. al., 2012). Both GraphLab and PowerGraph suffer from varying communication volumes due to power-law fan-in and fan-out because they do not take into account edge direction when synchronising data.

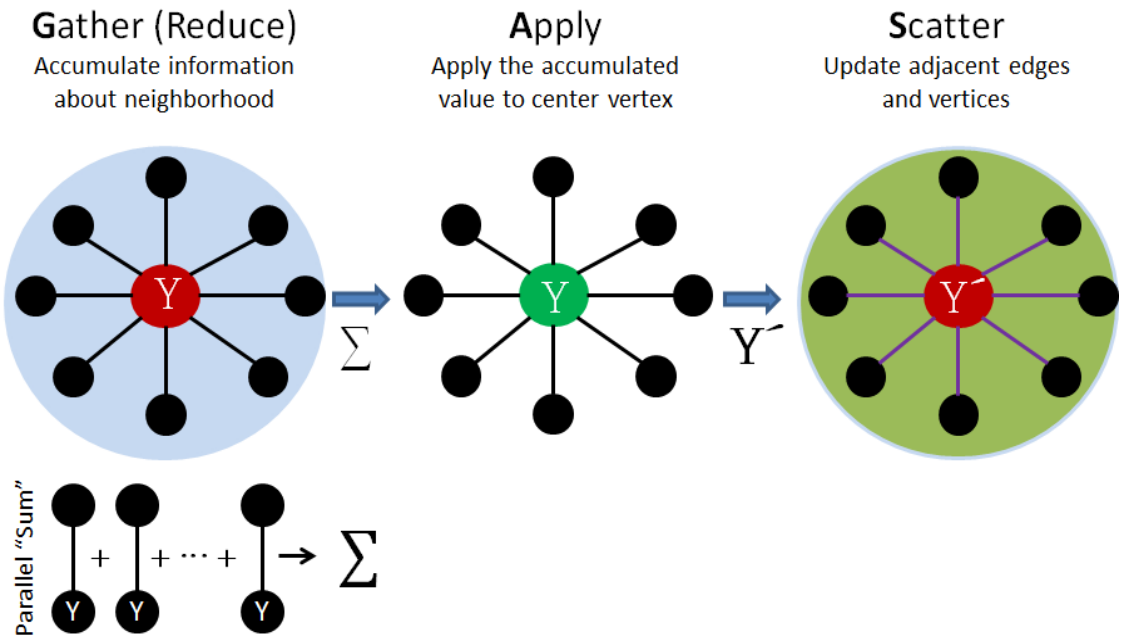

Figure 2.13: Gather-apply-scatter (GAS) decomposition (GraphLab Open Source, 2009)

However, PowerGraph performs significantly better and communicates remarkably less, due to the efficacy of the vertex cuts. While the above discussed parallel processing frameworks provide efficient tools for processing large data, adding higher and complex data representations within the model will vastly improve its usability and provide an important pattern recognition based data analysis option. For this purpose, in the following two sections, a detailed discussion on machine learning and pattern recognition is provided. This discussion will form the basis for the proposals later presented in chapters 3 & 4 to explore new methods for partitioning and distributing data in the clouds and propose detailed recommendations, with supportive data and prototypes, for optimally developing future data management models in the Internet.

2.5 Machine Learning and Pattern Recognition

Broadly speaking, ML is '*a branch of artificial intelligence, concerned with the construction of programs that learn from experience*' (Daintith & Wright, 2008). A

concept closely related to ML in general and classification in particular is '*pattern recognition'*. Pattern recognition is the automatic detection of regularities in data using computer algorithms and upon detection of such regularities, taking appropriate actions (Bishop, 2006). In most cases, the discovered regularities – also referred to as patterns – are further utilised to build models to represent real-world scenarios. Based on these developed models, which can be viewed as approximations of the real world, possible actions include categorising data into different groups and making predictions about future data trends based on the observed data (Alpaydin, 2004). Past computational experiences have shown that one of the most promising methods for building models and designing classifiers involves learning from example patterns (Duda, et. al., 2001). The corresponding approaches, which are typically referred to as ML, often include large volumes of training data or past experiences in order to accomplish their task. Moreover, the schemes utilised for pattern recognition and ML often tend to be complicated both in terms of training and computation time. A big burden for the adoption and integration of pattern recognition and ML techniques into real-world processing systems is the mathematical complexity and sophistication involved in adapting them for specific problem domains. Such schemes often have many parameters representing concepts that are usually not very straightforward for developers, and their performance behaviour is remarkably sensitive to the way the underlying pattern recognition modules are deployed and interconnected.

2.5.1 Pre-processing

Pre-processing is the foundation of a pattern recognition system for which some given raw data must be pre-processed and prepared by applying application–specific pre-processing algorithms before they can be fed into the system. For instance, document images often have to be binarised before conducting any sort of optical character recognition (OCR) or layout analysis. In other cases, the input data may be incomplete because of missing values, while the intended ML scheme does not process incomplete datasets. In this regard, an appropriately designed pre-processing stage would help with filling these gaps in the dataset – for example, by averaging or

using other proper statistical approaches. The pre-processing phase may also include steps to remove outliers or noise from the dataset. Therefore, choosing and applying the right pre-processing method will have a significant effect on further steps taken by the pattern recognition system.

2.5.2 Feature Selection

Generally, the patterns to be processed and categorised are represented by various measurement metrics referred to as features. To recognise patterns, a proper set of features must be chosen. These features must satisfy certain aspects in order to be most effective. They must be distinguishing enough for the patterns in the domain, invariant to irrelevant modifications of the input data, and sufficiently compact in size to minimise memory and resource consumption and computation time. Moreover, they should be easily extractable while being insensitive to noise. The selection of features may require prior knowledge of the problem domain, but choosing appropriate features is often a tricky task, and it sometimes involves lengthy evaluations.

2.5.3 Model Selection

The performance of a classifier also relies on the scheme that is utilised to approximate the real-world situations. More accurate approximations result in better classification rates and improved predictions. The challenge here is that various classes of models offer different real-world approximations for different problem domains. It is worth noting that the volume and quality of available sample data are important determining factors, as different classes of models exhibit different levels of robustness against noisy data. In addition, performance requirements may come into play when choosing a model.

2.5.4 Training, Testing and Optimisation

After selecting a classification scheme, it should be evaluated against example data. As a result, it must be applied on a data subset, also referred to as training data. The

output of this training phase forms a model that has to be further tested and examined in a subsequent testing phase by using a data subset called test data. One issue here is that the computation time for each phase may become quite excessive depending on the schemes used. In particular, the large number of iterations occurring within the training and testing phases during parameter optimisation may require significant computational requirements. A solution to this problem for the more effective deployment of ML techniques can be achieved by utilising parallelism within the system. As a result, the system can be empowered to use many processors, and even an excessive number of processing machines, at the same time. *This is where the concept of distributed computing comes into the picture.*

2.6 Distributed Approach for Large-scale Pattern Recognition

Distributed pattern recognition (DPR) can be an effective alternative for large-scale data-processing-related problems, where the recognition process is performed across a distributed system. Many past DPR approaches looked into creating a distributed architecture for pattern matching and placed less emphasis on the algorithmic approach. The focus on distributed architecture can create a strong dependency on the hardware implementation, resulting in inflexible schemes that cannot scale well to the size of today's data. An algorithmic-based DPR approach, which is scalable and independent of any hardware framework implementation, has yet to be fully developed. In this regard, the applicability of using a pattern recognition scheme for Internet-scale data processing is an open issue that needs to be addressed. To overcome this, several techniques are suggested in the literature, including data reduction, active learning and distributed models for large-scale recognition. Nevertheless, a common denominator of such schemes is buried in the algorithmic complexity of the employed recognition models. In fact, any effective DPR scheme for large-scale data analysis should be able to extract relevant information in the most efficient manner.

DPR for big data processing is a relatively unexplored area because pattern recognition is mainly considered problem-specific; until recently, there had been little focus on developing scalable recognition solutions for large-scale data analysis and processing. In addition, the complexity of existing schemes restricts their application for big data domains. A number of initiatives are proposed in the literature to target distributing recognition processes across a distributed environment, but they mostly suffer from achieving an effective parallelism strategy. In this regard, the neural network approach is believed to offer a promising mechanism for providing large-scale recognition. The scheme can achieve this efficiency by optimally distributing parallel computation tasks across the network to be implemented by a large number of interconnected neurons. Nevertheless, there are some major issues that need to be addressed including, the convergence problem, complicated iterative learning processes and low scalability due to training data. Hence, any scalable scheme for DPR should handle the following three recognition stages in the most effective manner possible: *the learning stage, the processing stage and the training stage*.

2.6.1 Learning Approach

The learning approach for pattern recognition schemes is a critical component in determining how efficient and effective the approach is when implementing pattern matching. Some of the prominent approaches discussed in the literature are Hebbian learning (Hebb, 1988), incremental learning (Schlimmer & Granger, 1986) and one-shot learning. Learning stage for pattern recognition schemes is a critical component in determining how efficient and effective the approach is when implementing pattern matching. Hebbian learning is a well-recognised learning approach based on the synaptic plasticity model. In this technique, the output of a neuron can have a significant effect on the input to other neurons. The Hebbian learning scheme is a classical approach when dealing with spatio-temporal recognition problems in auto-associative neural networks. Most of the existing neural network techniques use Hebbian learning in their learning stage, including Hopfield (Hopfield & Tank, 1985) and feed-forward neural networks (Nadal, 1989). Nevertheless, the technique suffers

from *potential saturation* and *catastrophic forgetting*, which make the approach not scalable for big data processing.

The incremental learning technique was developed as a solution to scalability issues encountered in pattern recognition applications (Song, Liu, Zhang & Yang, 2008). Incremental learning works by distributing the training data into a number of subsets, with each subset going through the training phase independently. Following this, the outputs from each training stage will be combined to yield the final result. This approach can simplify the issue of large training sets considerably, specifically for ML applications such as SVMs (Mavroforakis & Theodoridis, 2006). Moreover, the division of large training sets into smaller chunks helps the scheme to scale better. Nevertheless, the incremental learning technique does not function well enough when handling large-scale patterns. The reason is that more computational resources are needed to deal with larger patterns. In addition, the algorithm is tightly coupled and requires the implementation of kernel functions, which makes it computationally costly (Schlimmer & Granger, 1986). One-shot learning was developed based on the concept of a system using minimal initial data to learn information. Existing implantations of this technique use the probabilistic approach; a well-known example is the Bayesian classifier (Fei-Fei, Fergus & Perona, 2006; Miller, Matsakis & Viola, 2000). This one-shot learning method is similar to incremental learning in the sense that the learning stage continues with the introduction of new patterns. Graph Neuron (GN) (Khan & Mihailescu, 2004) also implements one-shot learning, but from a different perspective. The GN executes its learning algorithm using a neuron-adjacency comparison model that will be discussed later in this chapter.

2.6.2 Processing Approach

To achieve higher-speed processing for pattern recognition, one alternative is to distribute the input space within the scheme by enabling the parallel processing of recognition processes. Most of the existing neural network recognition approaches are iterative in nature, including the Hopfield network (Hopfield & Tank, 1985), the back-propagation neural network (BPNN) (Wythoff, 1993), the convolutional neural

network (LeCun & Bengio, 1995) and the fuzzy neural network (Kasabov, 1996). This iterative mode of processing makes these techniques time- and resource-intensive. As a result, they cannot efficiently scale with an increase in the size of the pattern domain. Moreover, current neural network approaches have mostly been developed for single–processor environments, while they are also tightly coupled. To overcome these issues, new approaches, which offer higher distribution and parallelism of data and processing, are yet to be realised.

2.6.3 Training Approach

Training in the recognition context refers to an approach where the scheme starts learning from a sample dataset to perform an actual recognition task. This training phase can be performed within a single cycle or multi-cycle, and depending on the nature of the application and its specific requirements, the training dataset size can be small or large. To meet generalisation purposes, current deterministic pattern matching approaches usually require very large training datasets to maintain all of the necessary characteristics of the actual data. Further, existing schemes usually perform multi-cycle training. Single-cycle training can be performed using the GN technique introduced by Khan (2002). The learning phase involves recognising adjacency values between the GNs rather than calculating weight values between the nodes, as is the case in the Hebbian and incremental learning schemes. The training phase in the GN is implemented using a single cycle, allowing faster pattern matching and recognition.

2.7 Graph Neuron for Scalable Recognition

Lazy learners mainly pass the computational cost to the recognition phase. Graph Neuron (GN) provides a theoretical limit on the number of steps required in the recognition stage (Nasution & Khan, 2008), and thus those of interest for this thesis. The GN is based on a special type of associative memory (AM) model that is readily implemented within distributed architectures. AM is a subset of ANNs that utilises the benefits of content-addressable memory (CAM) in microcomputers (Chisvin &

Duckworth, 1989). It is also one of the important concepts in associative computing. In this regard, the development of AM has been largely influenced by the evolution of neural networks. As discussed previously, existing AMs generally apply the Hebbian learning rule or kernel-based learning approach. Thus, these AMs remain susceptible to the well-known limits of these learning approaches in terms of scalability, accuracy and computational complexity. It has been suggested in the literature that graph-based algorithms provide various tools for graph-matching pattern recognition (Muhamad Amin, Khan, & Mahmood, 2009), while introducing universal representation formalism (Baqer & Khan, 2007). The main issue with most graph-based approaches lies in the significant increase in the computational expenses of the deployed methods as a result of the increase in the size of the pattern database (Khan, 2007). This increase places a heavy practical burden on the deployment of those algorithms in clouds for data-intensive applications and real-time data processing/database updating. However, as will be shown in the next few sections, none of the GN-based approaches are very sensitive to an increase in the size of pattern databases. This insensitivity is one reason why the GN exists.

2.7.1 Graph Neuron Architecture

The GN is a finely distributed parallel pattern recognition scheme that maintains data relationships in a graph-like memory structure. The GN framework and its data representations are analogous to a directed graph, where the processing nodes of the GN array are mapped as the vertex set $\mathbf{V}$ of the graph, and the inter-node connections (i.e., the communication channels) belong to the set of edges, $\mathbf{E}$. The communications are restricted to the adjacent nodes (of the array); hence, there is no increase in the communication overheads with corresponding increases in the number of nodes in the network (Khan et al., 2004). The information presented to each of the nodes is in the form of a (*value, position*) pair. Each of these pairs represents a data point in a two-dimensional pattern space. Hence, the GN array converts spatial/temporal patterns into a graph-like structural representation and then compares the edges of the graph with subsequent inputs for memorisation or recall (see Figure 2.14).

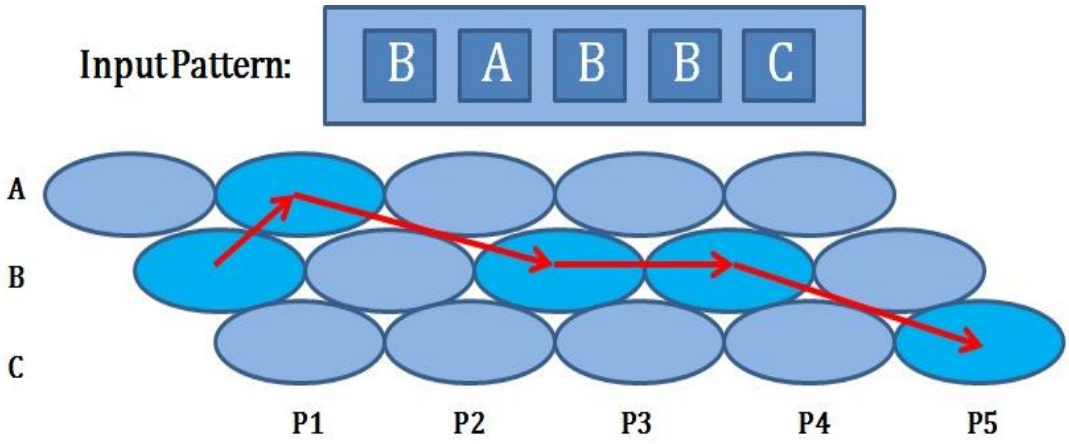

Figure 2.14: An input pattern *BABBC* is stored in a GN array where each row of the array represents a value and each column represents a position

The graph-based techniques may be applied for pattern recognition at the cost of computational cycles exponentially increasing with the increase in the number of stored patterns (Trajan & Trojanowski, 1984). The GN avoids these increases by performing the computations over a fine-grained parallel processing network, and it eases network bottlenecks through adjacent node communications. As far as we are aware, there is no other method in the literature that truly implements a scalable and accurate single-cycle AM model within neural networks. Morphological associative memories (MAM) apparently provide one-shot learning; however, the length of the learning cycle cannot be fully estimated a priori, and it depends on the size and number of stored patterns (Sussner & Valle, 2006). Single-cycle learning within the GN is achieved by sidestepping the commonly used error/energy minimisation approaches within ANNs. The GN array is designed to hold all possible solutions for the problem domain. The array can thus find the solution in a single cycle (i.e., a fixed number of steps). This is in contrast with the error/energy minimisation approaches, which generally start from a random point in the problem space and progressively close in on the solution. Hence, error/energy minimisation approaches tend to become costlier and inaccurate for larger problem sizes. We have successfully implemented the GN array in the simplest form for pattern recognition and aim to make it readily developed into a fully featured AM system. The resultant single-cycle

AM will transform the way complex data are currently processed by replacing iterative processes with a fixed number of steps. An abbreviated explanation of the approach is provided in the following subsection.

2.7.1.1 Single-cycle Learning Approach

The GN stores new patterns and recalls previously encountered patterns by executing a fixed number of steps. An input pattern **Ƥ** is represented as a set of **ρ**(*value, position*) pairs. These inputs are mapped onto a real or virtual processor array by using the adjacency characteristics of the input; for example, alphabets and numbers would have their inherent adjacency characteristics, and images would have frequency bands, intensity and spatial coordinates as the adjacency characteristics per pixel. For a reference pattern domain **Ʀ**, the GN array must represent all possible combinations of **Ƥ** in **Ʀ**. Hence, each GN node is initialised with a distinct pair **ρ** from the input domain **Ʀ**. Further, each GN node executes an instance of the full code associated with the GN algorithm. Hence, the computation overhead imposed on all nodes is the same. Each GN node maintains an updated list, called the '*bias array'*, which records the position(s) of its adjacent GN node(s) activated during a pattern input. The bias array is populated during the learning phase of the GN. Figure 2.15 depicts learning of four simple patterns, wherein pattern P1 comprising a string 'ABBD' is learnt by nodes GN(A,1)→ GN(B,2) → GN(B,3) → GN(D,4) of the array. The learning by these GN nodes occurs in the following stages:

- *Mapping of input patterns*: Input patterns in the form of **ρ**(*value, position*) pairs are sequentially broadcast through the network. Based on its predefined position and value setting, each node responds to the relevant input pair, disregarding the remainder of the pattern. From Figure 2.15, a node with a predefined value = 'A' and position = 1 will respond to the first sub-pattern/pair of pattern P1 (ABBD) – that is, value = 'A' occurring at position = 1. It will ignore the rest of the message and this will process continues for other patterns.
- *Synchronisation phase*: A broadcast signal is sent out to all of the nodes to mark the end of the incoming pattern.

- *Bias array update*: During this phase, each activated node contacts all of the adjacent nodes to find out which ones responded to the input. Figure 2.15 shows that for the input pattern P1(ABBD), Node GN(A,1) will update its local bias array with the entry {GN(B,2)}. Similarly, Node GN(B,2) will update its bias array with the entry {GN(A,1), GN(B,3)}. Thus, each bias array entry records the adjacent nodes being activated within a particular pattern input phase. That is, each row of the bias array comprises a list of the adjacent GN nodes activated during the input. A new pair is defined as the one that has a different set of adjacent GN nodes to all existing rows of the bias array. A new pattern is found when at least one GN within the list of activated GNs cannot find a matching entry in its bias array. In this stage, new patterns are stored and previously encountered patterns are recalled. *Our tests show that the memory requirement per GN node to maintain the bias array does not increase disproportionately with the increase in the number of stored patterns.* As shown in Figure 2.15, the maximum bias array size (three) occurs only in GN(C,3) after storing all patterns.

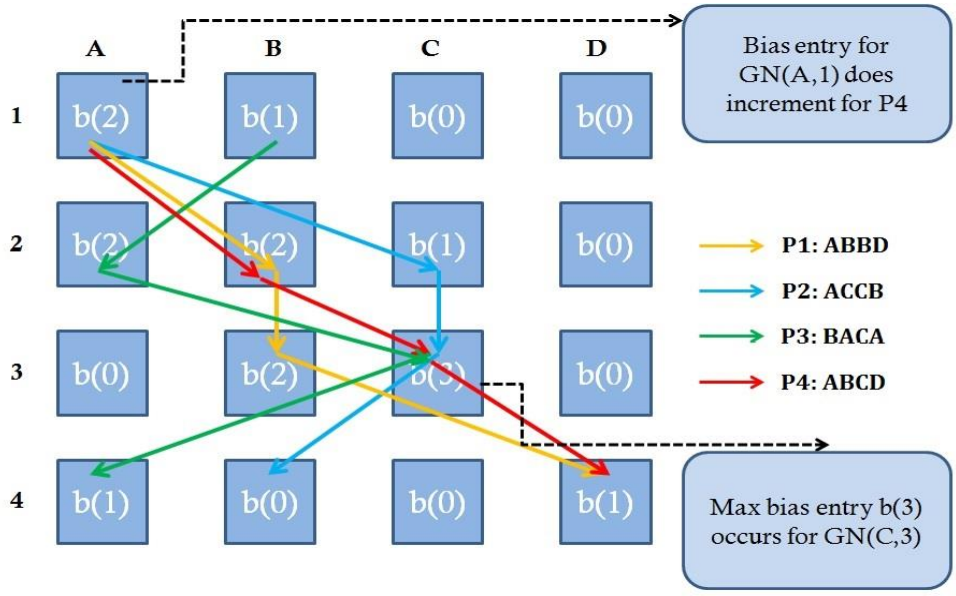

Figure 2.15: Four arbitrarily chosen patterns – P1: ABBD, P2: ACCB, P3: BACA, P4: ABCD – have been stored in the GN array. The maximum bias size is three for storing four patterns, indicating that the storage requirement per node would not disproportionably increase with the increase in the stored patterns.

The first and second stages of the GN learning phase take place in a parallel and decentralised manner. The results must be gathered from all activated GNs in third stage to make a memorisation/recall decision. Computational complexity for all three stages of the GN is mainly dependent on: (1) inter-node communication delays, (2) hardware/software latency per node and (3) delay in updating the local bias arrays. Hence, the total time needed to store or recall a pattern generally remains independent of the number of nodes present within a GN array. The scalability tests, with up to 16,384 nodes, show that computational complexity only increases nominally with increases in the size of the network (Baqer, Khan, & Baig, 2005). The distributed graph-based representation of patterns within the GN bias arrays would be valuable in segmenting spatio-temporal databases for classification. Spatio-temporal databases are multidimensional in nature. They capture both spatial and temporal relationships among stored objects, and they support efficient data retrieval (Theodoridis, et. al., 1996). These databases may be used to retrieve past information; as well as for future predictions based on the current dataset. In contrast with traditional database systems, spatio-temporal databases usually represent spatial and temporal data – for example, object trajectory – as motion functions and spatial relationships such as distances and adjacencies (Tao, et. al., 2003). With its graphical pattern recognition characteristics, the GN can act as an ideal platform for the storage (and subsequent retrieval) of complex features within the stored patterns. The pattern recognition process itself will be efficient, and the storage of large pattern databases will be facilitated by the distributed nature of the GN. The order of event occurrence would be preserved by the GN owing to its directed graph-based representation of the stored patterns. Hence, the temporal aspects of the dataset can be readily analysed alongside the spatial ones using the universal graph-based representation of patterns with the GN. Among various applications of the GN-based AM, an input pattern in GN pattern recognition may represent bit elements of an image (Khan & Muhamad Amin, 2007) or a stimulus/signal spike produced within a network intrusion detection application (Baig, et. al., 2006).

2.7.1.2 GN Pattern Crosstalk Problem

It may be seen in the bias array update process that each GN only requires its own input pair value and the values of its adjacent GNs to analyse a pattern. Hence, the limited perspective of GNs can lead to inaccurate results. Several GNs could recall their sub-patterns simultaneously, leading to a full recall. However, the recalled pattern may not have been presented before. For example, assume that a GN array can receive patterns comprising six possible values, *a, b, c, d, e* and *f,* and five possible positions. After memorising input patterns *abcdf* and *fbcde*, the array is presented with the pattern *abcde*. The array would raise a false recall, indicating the pattern *abcde* has already been memorised by the array. To understand this occurrence, we need to examine the GN scheme. Under this scheme, five of the GNs would recall sub-patterns: *ab, abc, bcd, cde* and *de*. Among these five, two of the GNs are at edge columns and thus only report shorter sub-patterns (i.e., *ab* and *de*). The recalls of all five sub-patterns are correct from each active GN's perspective, but the overall recall of the pattern by the array is incorrect. The GN communications will need to be extended to a layered communication schema to solve the crosstalk problem. In such a schema, each of the GN layers would propagate its local decision (i.e., memorisation or recall) to the layer above it. The layers can preserve the original GN functionality. In such a scenario, the topmost GN layer will be responsible for the fusion of all localised decisions into a final memorisation or recall decision. The pattern overlap problem can also be solved by increasing the length of the mapped bits to minimise the pattern overlaps. In doing so, we can lower the overall computational complexity within our technique without adding to the communication cost. The layered approach will handle noisy and distorted patterns by progressively adjusting the GN recognition criteria in the layers.

2.7.2 Hierarchical Graph Neuron (HGN)

The HGN extends the original GN algorithm to form a highly resilient distorted/noisy pattern recogniser with an average distortion tolerance within 15% to 20% (Nasution & Khan, 2008). The HGN comprises three layers of processing nodes, namely base,

middle and top layers (see Figure 2.16). The base layer acts as the input layer for the pattern recognition process, with all processing nodes at this layer receiving input patterns from the test dataset. The activated HGN nodes in this layer send an index value to the upper layers. A unique index value is generated for every new input pattern. The middle layers receive indices from the base layer after comparing them with the adjacent nodes, and the selected indices are then sent to the upper layer. The top layer decides whether the input pattern should be considered a new pattern for memorisation, or whether it is classed as an existing pattern for recall.

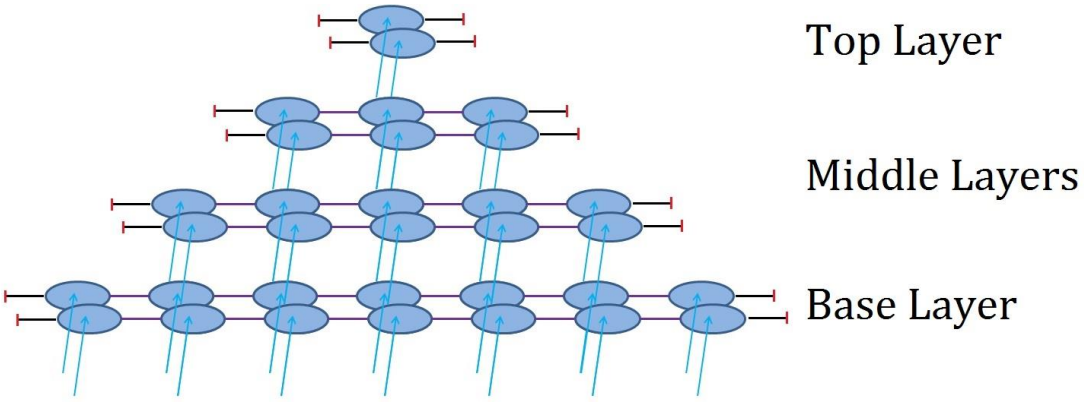

Figure 2.16: HGN with pattern size of seven and two possible values within the pattern (Nasution & Khan, 2008)

The HGN expands the capability of '*perceiving neighbours*' within the network by adding higher layers of GNs that see all of the pattern information and provide a bird's eye view of the overall pattern. Figure 2.16 illustrates a HGN structure that is only used in pattern recognition applications involving one-dimensional patterns. However, for applications that involve complex patterns with higher dimensionality, the HGN can easily be expanded to two, three or even multidimensional hierarchies. Figure 2.17 depicts examples of a HGN composition for a two-dimensional pattern of size 35 (7 × 5) and a three-dimensional pattern of size 105 (7 × 5 × 3).

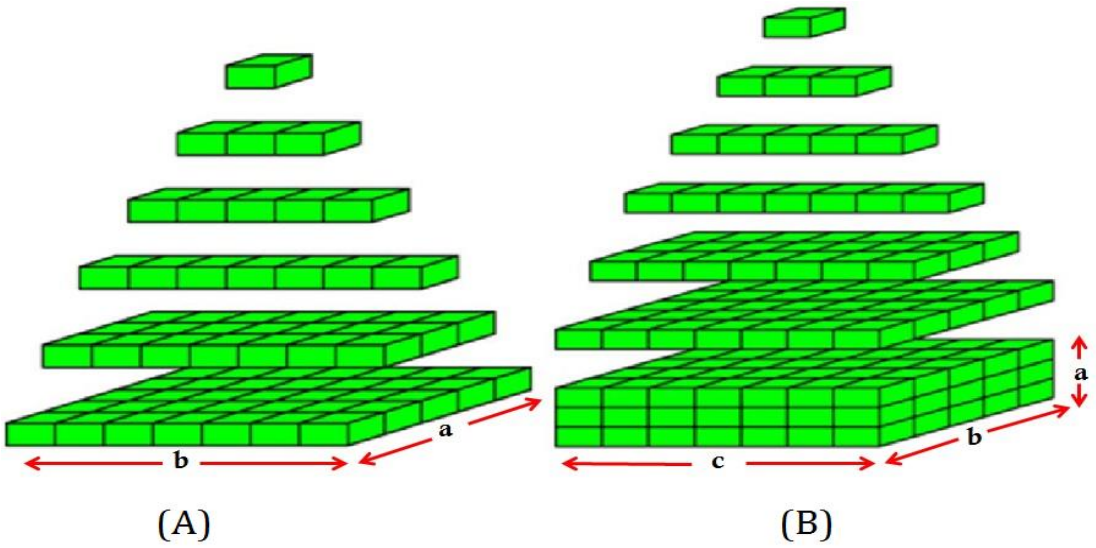

Figure 2.17: HGN compositions of (A) 2-D (7x5) and (B) 3-D (7x5x3) for pattern sizes 35 and 105, respectively (Nasution & Khan, 2008)

In an interesting side effect, increasing the dimensions of a HGN network topology results in fewer processing neurons in the hierarchy. This will significantly improve the performance of the system when dealing with large-scale patterns. For example, given a one-dimensional pattern of size 105, the total number of GNs required is: 105 + 103 + 101 + . . . + 3 + 1= 2809. A two-dimensional (15 × 7 = 105) GN composition requires: 15 × 7 + 15 × 5 + 15 × 3 + 15 + 13 + . . . + 3 + 1 = 279 GNs. In this example, increasing the dimensionality by one led to an approximate 90% reduction in the number of GNs in the composition (Nasution & Khan, 2008).

2.7.2.1 HGN Communication Approach

HGN implementation follows a graph-based pattern representation approach similar to the original GN. However, to end up with a hierarchical structure with a top neuron overseeing the overall pattern, there is a requirement to have an odd pattern size. As result, for every even-length pattern, a '*dummy*' value will be inserted at the end of the pattern to ensure that the HGN hierarchy can be formed as per requirements. There are a number of steps in the HGN pattern-matching model, including a recognition process occurring at every layer within the hierarchical structure. The

HGN communications procedure is as follows: the base layer of the HGN first processes the incoming pattern, where each neuron that receives an input will become an active neuron. Each active neuron in the base layer will then send its **ρ**(*value, position*) pair to all of the adjacent neurons to inform them that it is active. For each of the active neurons in the base layer, the **ρ**(*value, position*) pairs received from the adjacent nodes will form the bias array entry for the current input pattern. At the end of this process, each of the non-edge neurons should have received two pairs from its adjacent nodes, while the ones on the edges should have received one. At this stage, each active neuron must calculate its bias array index. If the incoming pair combination is matched with an existing entry in its bias array, then the index of the entry will be noted. Otherwise, a new index will be generated and stored to reference the new combination. Each of the active non-edge neurons then sends its index to the corresponding neuron in the same column but in the higher layer. This process will continue until the top-layer neuron is reached. The top-layer neuron will be in the position of deciding whether the input pattern should be treated as a new pattern and stored, or whether it should be treated as a previously visited pattern and recalled. Upon making its decision, a new index value will be sent downward for a stored pattern, and an existing index value will be sent downward for a recalled pattern. The bias array structure in the HGN follows the same principles as the bias array formation in the original GN. Nevertheless, it has been altered to cater for the recognition processes of higher-layer neurons based on adjacency comparison information provided by lower-layer neurons. The process of forming the bias array is as follows:

- For neurons in the base layer, their bias array entry takes the form {*left, right*}, where left and right represent the row number of left-adjacent and right-adjacent neurons, respectively.
- For neurons in the middle layer, their bias array entry takes the form of {*left_index, lower_index, right_index*}, where *left_index, lower_index* and *right_index* represent indices obtained from its left, lower (within the same column) and right neurons, respectively.

- The bias entry structure of the top-layer neuron is in the form of {*lower_index*}, which is obtained from its lower-layer neuron (within the same column).

The HGN is highly accurate when dealing with noisy patterns, however, the HGN implementation requires large computational resources (large number of processing nodes).

2.7.3 Distributed Hierarchical Graph Neuron

The functionality of the HGN is further extended by dividing and distributing the recognition processes over the network. This distributed scheme minimises the number of processing nodes by reducing the number of levels within the HGN. *This transformation of the HGN into an equivalent distributed hierarchical graph neuron (DHGN) composition allows, on average, an 80% reduction in the number of processing nodes required for the recognition process* (Khan & Muhamad Amin, 2007). Therefore, the DHGN can substantially reduce the computational resource requirement, from 648 processing nodes to 126 for the case shown in Figure 2.18.

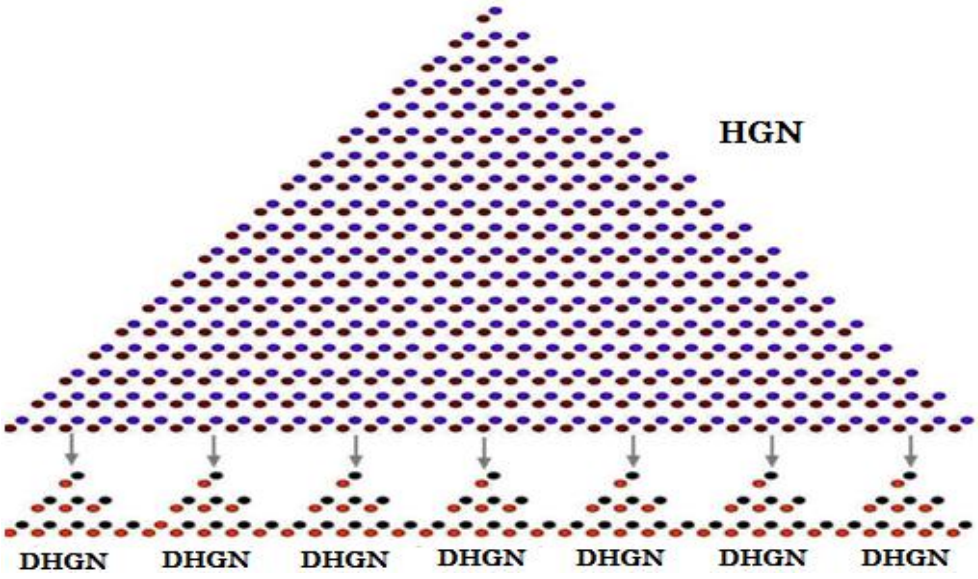

Figure 2.18: Transformation of the HGN structure (top) into an equivalent DHGN structure (bottom) (Khan & Muhamad Amin, 2007)

This figure shows the divide-and-distribute transformation from a monolithic HGN composition (top) to a DHGN configuration for processing the same 35-bit patterns (bottom). The base of the HGN structure in the figure represents the size of the pattern. Note that the base of the HGN structure is equivalent to the cumulative base of all DHGN subnets/clusters. The DHGN allows the recognition process to be conducted in a smaller sub-pattern domain, hence minimising the number of processing nodes, which in turn reduces the complexity of the pattern analysis. In addition, each subnet is only responsible for memorising a portion of the pattern (rather than the entire pattern). A collection of these subnets can form a distributed memory structure for the entire pattern. This feature enables recognition to be performed in parallel and independently. The HGN and DHGN provide higher levels of accuracy by introducing a hierarchal network topology. Moreover, the communication overhead in both schemes is minimal considering the adopted parallel and distributed mechanisms. However, the scalability of the HGN and DHGN techniques is not well suited for large-scale data-processing problems, as the number of required nodes can increase significantly with the increase in the size of the pattern space, and none of them can effectively fulfil scalability requirements for Internet-scale pattern recognition; consequently, new schemes need to be proposed.

2.8 Conclusion

The efficiency of the cloud system in dealing with data-intensive applications through parallel processing essentially lies in how data are partitioned among nodes and how collaboration among nodes is handled to accomplish a specific task. As a result, and to address the aforementioned concerns in relation to data storage and retrieval in the cloud, any data access schemes should aim to handle partitioning between processing nodes, as well as node collaborations, in a robust manner. These two features are still lacking in current data access mechanisms. Hence, new data management approaches need to be investigated for cloud computing environments. This chapter presented a comprehensive study of the current data-parallel frameworks for cloud data

processing, and it explored different approaches to large-scale data processing. The pros and cons of each approach were examined in relation to the scalability and adaptability requirements of big data processing. This chapter also presented a detailed analysis of how neural network approaches can open a new pathway for accessing data in highly distributed environments by discussing some of the major schemes presented in the literature. The investigation revealed that existing neural network techniques are far from providing a suitable scalable framework for large-scale recognition purposes. However, initiatives are currently being undertaken to establish more effective approaches using existing neural network algorithms.

One of these initiatives is the development of the GN – a single-cycle AM algorithm – to implement a scalable AM device through its parallel in-network processing framework. The GN uses a graph-based model for pattern learning and recognition. One of the peculiarities of this technique is the employment of parallel in-network processing capabilities to address scalability issues effectively, which is a major concern in graph-based approaches. The limited perspective of GNs, owing to purely adjacency-based computations, was widened through the HGN approach for distorted pattern recognition, which is the first distributed pattern recognition scheme that specifically targets WSN as the platform. The HGN provides a bird's eye view of the overall pattern structure and hence eliminates the crosstalk issue in pattern recognition. The HGN is highly accurate regarding noisy patterns. However, HGN implementation requires relatively larger computational resources in terms of the number of processing nodes. In addressing this limitation, the DHGN has been developed as an extension of the HGN. DHGN implementation involves the decomposition and distribution of patterns into sub-patterns, and recognition processes occur at the sub-pattern level. In doing so, the number of processing nodes required is significantly reduced. Nevertheless, the DHGN still suffers from scalability issues, as the size of the network increases significantly with the increase in the size of input pattern space.

The main contribution of this chapter was to conduct a detailed review of scalable pattern recognition requirements and the shortcomings of existing techniques in the

literature. We hypothesise that fundamental changes and improvements in data access and movements are possible and beneficial for cloud-based processing. In this regard, AM concepts open a new pathway for accessing data in a highly distributed environment that will facilitate a parallel-distributed computational model to automatically adapt to the dynamic data environment for optimised performance. The problem is to marry such concepts with relevant advanced parallel processing patterns. With this in mind, the remaining chapters will target a new type of data-processing approach that will efficiently partition and distribute data for clouds and facilitate content-based access for a wide range of applications. Thus, a fully DPR scheme that can work with a parallel-distributed computational model such as MapReduce will provide a reusable cloud-based framework for a range of applications, from image search and sensor data analysis to planetary monitoring and the control of cyber-physical infrastructure, mobile equipment and devices. The ability to partition data optimally and automatically will allow elastic scaling of the system. Moreover, improved data management, where data are optimally and automatically distributed, will improve application performance through efficient data access.

Chapter 3

Edge Detecting Hierarchical Graph Neuron (EdgeHGN)

The performance of associative learning schemes can be substantially improved by dividing patterns into sub-patterns and then distributing them across multiple computational networks. This improvement is due to two main reasons: (1) the scalability of the recognition scheme will be reasonably improved due to the distributed nature of the process, and (2) the distribution of patterns into sub-pattern domains enables better-controlled error encapsulation in a specific subnet, resulting in a more accurate approach. In this regard, the GN-based schemes have been designed and structured based upon two fundamental concepts of *graph-matching* and *associative memory*, providing them with acceptable levels of scalability for implementation. Moreover, their simple one-shot (single-cycle) learning mechanism, along with their lightweight algorithm, make them suitable for performing pattern recognition on distributed systems while incurring low computational and communication costs. In the previous chapter, Graph Neuron (GN), Hierarchical Graph Neuron (HGN) and Distributed Hierarchical Graph Neuron (DHGN) approaches are discussed.

This chapter will discuss the algorithmic design and architecture of the newly proposed scheme, referred to as Edge Detecting Hierarchical Graph Neuron (EdgeHGN), and its performance for distributed large-scale data-processing schemes will be analysed in detail. To achieve better scalability and higher effectiveness when performing pattern matching, EdgeHGN performs its recognition functions using fewer processing neurons, resulting in reduced computational complexity and minimised processing requirements. It also provides a high level of parallelism by enabling recognition processes to be executed as a composition of sub-processes that are being handled in parallel across a distributed processing environment. This sub-process functionality is conducted in a purely independent manner, making the scheme less cohesive compared to many other pattern matching algorithms.

3.1 Associative Memory Concept for Pattern Recognition

Associative Memory (AM) for pattern recognition refers to a set of learning networks or functions that provide an association between input and output. According to Roman-Godinez, et. al., (2009), Associative Memory **M** is a system that offers an input-output association relationship as follows: $\alpha \rightarrow \mathbf{M} \rightarrow \beta$, where α and β are input and output respectively. From this perspective, each input vector will be associated with an output vector, which will be well represented in the form of a fundamental set of associations: $\{(\alpha^{\mu}, \beta^{\mu}) \mid \mu = 1, 2, ..., \rho\}$. This set is a priori knowledge that needs to be known to the AM system. For pattern recognition purposes, there will be two types of AM, namely auto-associative memory and hetero-associative memory. Auto-associative memories are content-based memories that can recall a stored sequence when they are presented with a fragment or a noisy version of it. In auto-AM, the system performs a recognition task on an input pattern that is presented to the AM system and generates its associated output pattern. As a result, for a given set of associations $(\alpha^{\mu}, \beta^{\mu})$, the auto-AM rule is true with the following condition*:*

$$\alpha^{\mu} = \beta^{\mu} \quad , \quad \forall \mu \in \{1, 2, ..., \rho\} \tag{3.1}$$

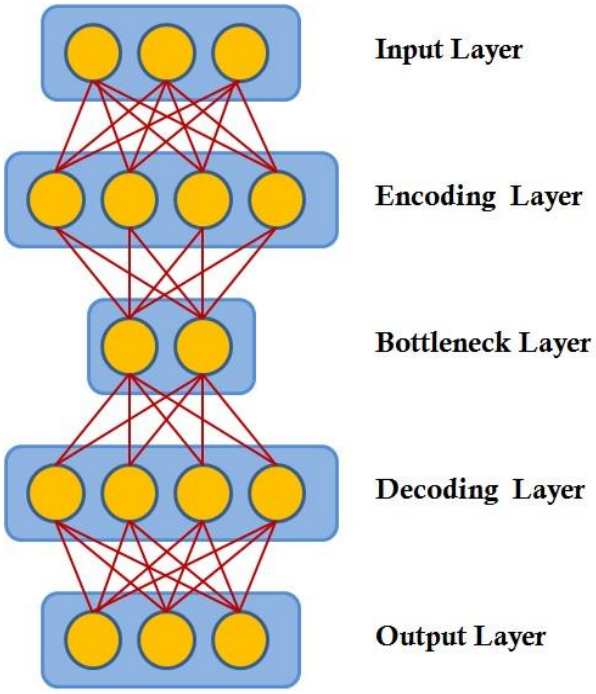

Figure 3.1: Auto-associative memory network to determine whether the input vector is '*known'* or '*unknown'*

Figure 3.1 represents an auto-associative memory network to determine whether the input vector is '*known'* or '*unknown'* to the system. In this figure, the training input and the target output are the same. In this setup, the stored vector can be retrieved from the distorted input if the input is sufficiently similar to it. The network performance is judged based on its ability to reproduce stored patterns from the noisy input. The network recognises the known vector by producing a pattern of activation on the output units of the network, which is the same as one of the vectors stored in it. The auto-associative framework enables the AM system to pass through input patterns towards output patterns without any changes because the input/output patterns have similar characteristics. The Hopfield network is an example of an auto-AM system. Conversely, hetero-AM system follows the rule of association in a way that incomplete input patterns can also result in complete output patterns. Here, the training input and target output are different. The weights are determined in such a way that the network stores a set of pattern associations. Hence, in terms of the association set (α^{μ}, β^{μ}), the following rule applies where:

$$\alpha^{\mu} \neq \beta^{\mu} \quad , \quad \text{for } \exists \mu \in \{1, 2, ..., \rho\} \tag{3.2}$$

In this scenario, given a distorted pattern $\dot{\mathbf{P}}$ of original pattern $\mathbf{P}$ as input, the AM system is capable of recalling pattern $\mathbf{P}$. Bidirectional associative memory (BAM) is one of the neural network schemes that adopts this hetero-AM approach. Hetero-AM also enables a recognition task to be performed on patterns with different sizes, such as the work presented by Kosko (1988). AM approaches such as the Hopfield network and fuzzy associative memory (FAM) (Kosko, 1992) are computationally intensive and iterative in nature. Conversely, Morphological Associative Memory (MAM) (Ritter, et. al., 1998) offers a solution within a single iteration, and hence provides single-cycle learning. However, MAM is a tightly coupled algorithm and relies on global maximum/minimum computations; thus, it is not readily distributed. GN-based schemes including HGN and EdgeHGN implement an auto-AM approach in their recognition process. In fact, the GN algorithm can recall patterns that have been memorised by the network. The memorisation phase could be executed either in the pre-execution stage (supervised recognition) or instantaneously as part of the recognition process (unsupervised recognition). The scalability features of EdgeHGN and other GN-based algorithms have been further contributed to by the adaptation of the auto-AM approach. EdgeHGN features will be further investigated in detail in the following sections in this chapter.

3.2 Pre-processing and Dimensionality/Content Reduction

Pre-processing is an important task that needs to be carried out before any recognition procedure. The main objectives of pre-processing are to reduce the quantity of data being analysed, while simultaneously enhancing its quality. It is considered a pre-requisite for most pattern recognition systems due to its critical influence in ensuring that pattern data are in the specific form that suits the algorithm or implementation. Moreover, raw pattern data might need to be normalised beforehand to ensure that the data are well distributed and do not contain any outlier values. When dealing with complex data such as images, environmental sensory readings, biomedical and

biochemical structural data, the dimensions of the data involved are usually at higher dimensions (more than one). The dimension of the data is the number of variables that are measured on each observation. High-dimensional datasets present many mathematical challenges. One of the problems with high-dimensional datasets is that, in many cases, not all of the measured variables are *important* for understanding the underlying phenomena of interest (Donoho, 2000). While certain computationally expensive novel methods can construct predictive models with high accuracy from high-dimensional data, it is still of interest in many applications to reduce the dimension and complexity of the original data prior to any data modeling (Breiman, 2001). From this perspective, two different approaches could be carried out to reduce the data complexity in terms of its dimensionality:

- *Structural reduction*: In this approach, the structure of the data will be reduced into a lower dimension. This can be achieved by projecting the data by linear transformations into lower-dimensional sub-spaces.
- *Content reduction*: For high-dimensional data, if no data reduction is carried out before inputting the patterns to classifiers, the computation required may be too heavy. Hence, by using specific data reduction techniques, we can obtain a reduced representation of the dataset that is much smaller in volume, but that also produces the same (or almost the same) analytical results.

In the following section, these two approaches are discussed in relation to EdgeHGN implementation.

3.2.1 Structural Reduction

Structural reduction in EdgeHGN pre-processing involves the reduction of the structural composition of patterns from a high-dimensional structure into its corresponding low-dimensional representation. In this approach, pattern data undergoes structural deformation, while the contents or elements within the pattern remain intact. Further, structural reduction works on the basis that the structure of the data is unlikely to be significant in determining the characteristics of the pattern. Consider two-dimensional binary images with the size of 7 x 5 bits (i.e., 35-bit

image), as shown in Figure 3.2. In the structural reduction approach, this image will be re-arranged in the form of one-dimensional bit-string. This rearrangement enables the algorithm to work on patterns in a low structural dimension.

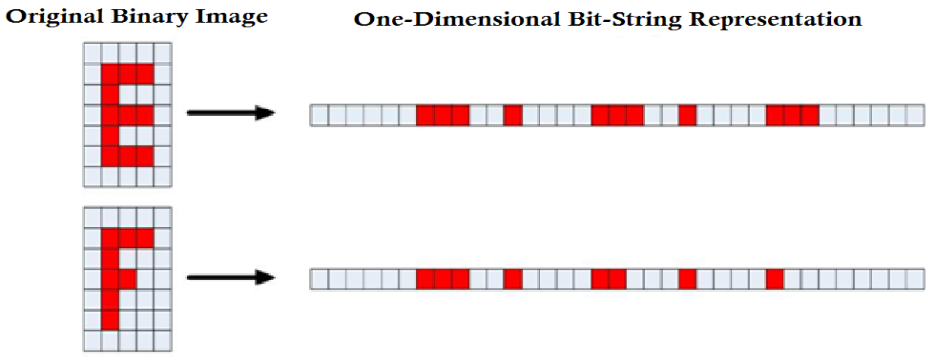

Figure 3.2: Structural reduction on binary character images

Hence, from the perspective of the EdgeHGN's implementation, this approach enables each subnet to conduct a recognition process using a simple one-dimensional EdgeHGN subnet structure. Therefore, it reduces the structural complexity of EdgeHGN subnets within the network. An advantage of using this structural reduction approach is that it reduces the structural complexity of patterns while maintaining the integrity of the contents or elements within these patterns. Hence, the content information in each pattern is preserved. A limitation of this approach is that it loses the structural information related to the pattern. In this context, the structure of the pattern or data is unknown to the system. Consider the same images as in Figure 3.2. The EdgeHGN pattern recogniser does not have the knowledge that the image represents the character **E**. Rather, it acknowledges the bit information and its association between neighbouring pixels in a one-dimensional formation.

3.2.2 Content Reduction

Content reduction involves the process of the selection or extraction of features from data, to be used in a pattern recognition system. It also transforms the data from high-dimensional space into its equivalent low-dimensional format. Some examples of

dimensionality reduction techniques include principal components analysis (PCA) (Abdi & Williams, 2010), linear discriminant analysis (LDA) (Martinez & Kak, 2001) and local linear embedding (LLE) (Gashler & Martinez, 2011). The dimensionality reduction approach allows the recognition system to obtain the best and most cost-efficient data representation that has been extracted from the original raw data obtained from sensory devices or from surroundings. For this purpose, in our proposed EdgeHGN model, and as part of the pre-processing phase, we reduce redundant data content for recognition by applying a lightweight hybrid drop-fall algorithm on the input pattern. This results in fewer processing neurons, which in turn results in a lower communication overhead within the scheme (see Figure 3.3).

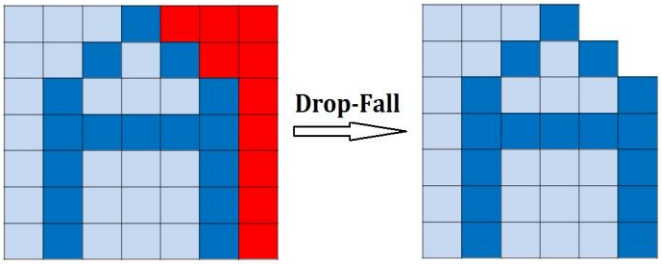

Figure 3.3: EdgeHGN progressively removes unnecessary nodes from the two dimensional data using drop-fall for content reduction

In Figure 3.3, a descending-left drop-fall algorithm is applied on the input pattern, reducing the number of processing nodes for each EdgeHGN subnet from 49 to 39. This reduction will minimise communication costs, and having an edge detection feature within the scheme can improve recognition accuracy to a high degree. Further, fewer neurons will result in a lower response time, which is of high interest for real-time pattern matching problems.

3.2.2.1 Drop-fall Algorithm

A drop-fall algorithm is often used for dividing touching pairs of digits into isolated characters (Congedo, et. al., 1995). A drop-fall algorithm simulates the path produced by a drop of water falling from above the character and sliding downwards along the contour under the action of gravity. The dividing path produced by the drop-fall

algorithm depends on three aspects: *a start point, movement rules and direction*. Based on this simple description of the method, the main issue that needs to be addressed in its implementation is the starting point. There are several methods available to decide where to start the drop-falling process. Dimauro, et. al., (2009) outlined a method that does this quite robustly. In this method, the pixels are scanned row-by-row until a black boundary pixel with another black boundary pixel to the right of it is detected, where the two pixels are separated by only white space. This pixel is then used as the point from which to start the drop-fall (see Figure 3.4).

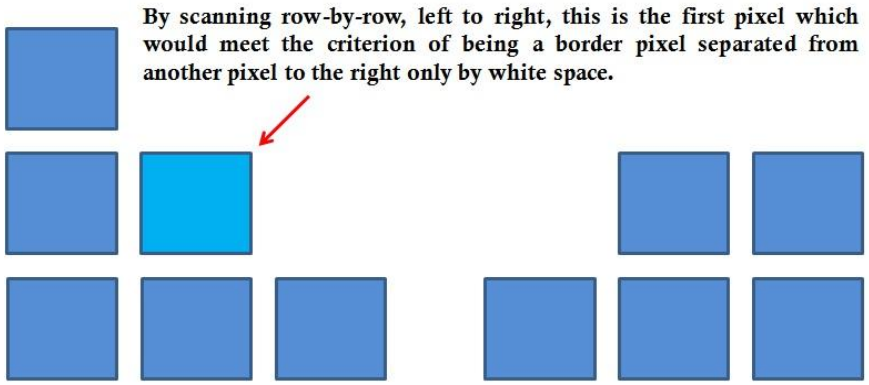

Figure 3.4: Pixel from which to commence the drop-fall

After the initial pixel is found, the next step is to begin the actual drop fall. The drop-falling algorithm is designed to mimic falling, so it will always move downwards, diagonally downwards, to the right or to the left. Figure 3.5 shows the directions that the algorithm will *move* in according to the current pixel position and its surroundings. The standard version of the drop-fall algorithm described above falls down and to the left/right of the pattern character. In other variations of the algorithm, they do not necessarily initiate from the top or fall down. Bottom-left or bottom-right drop-falls are identical in principle to the original drop-fall algorithm, except that they initiate from a pixel at the bottom of the image and *fall* up and to the left or right of the pattern. Despite the apparent similarity between the variations of the drop-fall heuristics outlined above, they often provide very different segmentation paths for various test examples.

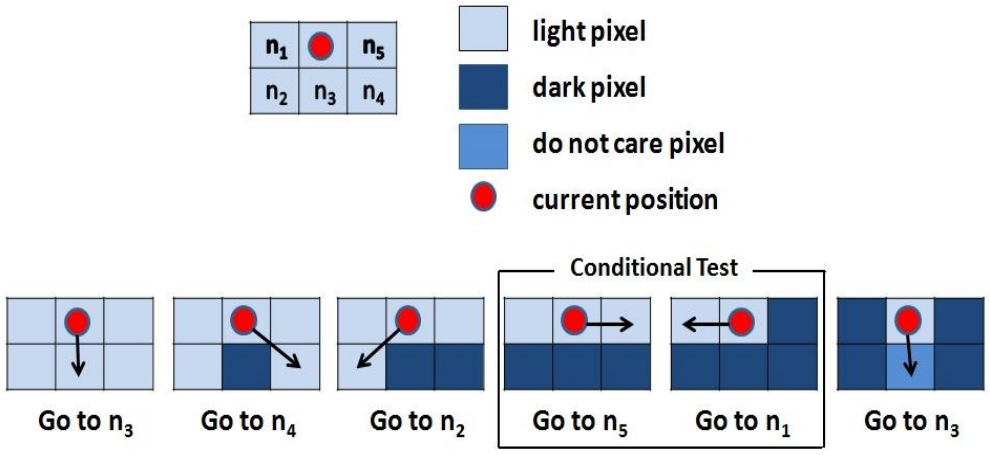

Figure 3.5: Movement rules for Drop-fall algorithm

For any of the drop-fall heuristics, it is easy to find cases where they work well and where they do not work well. This principle will provide the basis for the construction of a hybrid heuristic method that makes use of all of the aforementioned drop-fall variations. In the case of EdgeHGN, a hybrid drop-fall heuristic approach will be applied to the pattern to ensure it is producing the least number of processing neurons while maintaining all required character data bits for recognition (see Figure 3.6).

Figure 3.6: Hybrid drop-fall heuristic approach on character data patterns

3.3 EdgeHGN Computational Architecture

An important aspect in the development of the pattern recognition scheme is in algorithmic design. A proper design will lead to high efficiency and will have the ability to generate a more accurate classification strategy. GN-based algorithms have been developed based on two different concepts, namely *graph-matching* and *associative memory*. These two concepts provide an added advantage in terms of scalability for GN-based algorithm implementations. GN can perform pattern recognition processes on distributed systems due to its simple recognition procedure and lightweight algorithm. Further, the GN incurs low computational and communication costs when deployed in a distributed system. This section presents the algorithmic design of a newly proposed Edge Detecting Hierarchical Graph Neuron (EdgeHGN) algorithm for distributed pattern recognition scheme for large-scale datasets. The proposed approach extends the scalability of the existing GN implementations (HGN and DHGN) by reducing computational requirements in terms of the number of neurons for recognition processes, while providing comparable recognition accuracy. EdgeHGN provides a capability for the recognition process to be deployed as a composition of sub-processes that are being executed in parallel across a distributed network. Each sub-process is conducted independently, making it less cohesive compared to other pattern recognition approaches.

As mentioned above, the EdgeHGN scheme formalises a distributed version of the HGN by dividing and distributing patterns into sub-patterns and hence utilising a clustering approach for pattern recognition. Each of the sub-patterns undergoes a one-shot recognition procedure, and the results of the sub-recognition tasks will cumulatively add up to obtain the final recognition result. The EdgeHGN network constitutes a number of EdgeHGN subnets and a stimulator/interpreter module (SI module) node. Figure 3.7 shows the complete architecture of the EdgeHGN network. It illustrates a decomposition of the binary image pattern **A** into sub-patterns. The SI module node performs this composition process after the hybrid drop-fall pre-processing scheme is applied to reduce redundant content.

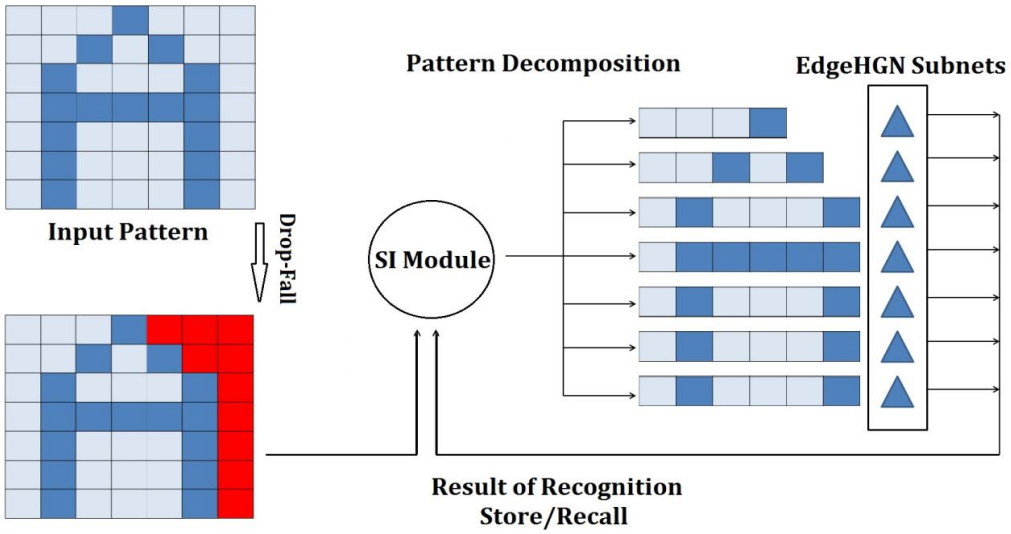

Figure 3.7: EdgeHGN framework for distributed pattern recognition

The input activates the GN nodes corresponding to the bits of the input pattern. In doing so, each pattern element within a sub-pattern is mapped to the relevant GNs in the respective subnet. Each subnet integrates its responses and sends the results back to the SI module to form an overall response. Figure 3.7 also illustrates the fact that communication within the EdgeHGN network occurs in a single-cycle fashion, where each pattern is passed through the network only once. This recognition process results in either a '*recall'* (pattern is known) or '*store'* (pattern is memorised). Moreover, within each EdgeHGN subnet, the recognition process involving communication between GNs occurs once for each sub-pattern, eliminating the need for any iterative mechanism while offering a fast recognition approach. The EdgeHGN disseminates the recognition processes at sub-pattern levels, allowing the processes to be conducted in a smaller sub-pattern domain. Further, the recognition processes within each sub-pattern are independent from other sub-patterns, thereby enabling them to be distributed across the network. The EdgeHGN implementation has been already conducted in a cloud environment. A further discussion of this will be presented in Chapter 4. The cloud environment provides the facilities for the EdgeHGN to distribute the recognition functions effectively within multiple networks or domains and enables an efficient resource management scheme for its implementation.

3.3.1 Two-stage Recognition Procedure

The EdgeHGN framework introduced in the previous subsection consists of two important entities, namely *SI module* and *EdgeHGN subnets*. The recognition of patterns mainly occurs within each EdgeHGN subnet after the hybrid drop-fall pre-processing scheme is applied; however, in this instance, all that is known to each subnet is only a sub-composition of the overall pattern. This means that there is a need for the EdgeHGN system to restructure the overall information of the pattern and produce the result for the entire pattern – that is, regardless of whether the input pattern is known to the system. In this regard, there is a need for another phase of recognition involving the results of the recognition process executed within each of the subnets. Thus, the EdgeHGN distributed pattern recognition performs pattern analysis upon completion of two different phases: (1) *sub-pattern recognition* and (2) *overall pattern reconstruction and recognition*. It should be noted that these two phases occur consequently and within a single cycle of the recognition process.

3.3.1.1 Sub-pattern Recognition Level

In EdgeHGN implementation, the core recognition process is conducted at the sub-pattern level after the hybrid drop-falling scheme is applied for the purpose of dimensionality/content reduction. There are four stages involved in this sub-pattern recognition level:

- *Stage 1*: After receiving an input from the SI module, each of the activated GNs at the base layer will send a signal message to other nodes in the adjacent columns containing the row number/address of the activated node. The activated nodes at the edge of the layer will only send the activation signal messages to the GNs in the penultimate columns.
- *Stage 2*: All active GNs at the base layer will then update their bias arrays. If the bias entry value, $\mathbf{bias_{ent}(left\ ,\ right)}$, received from both the activated nodes in the preceding and succeeding columns, have been recorded, the index of the entry will be sent to the respective GN in the same position at the higher layer. If the $\mathbf{bias_{ent}(left\ ,\ right)}$ value is not found within the bias array, then a new index will

be created and sent to the GN node in the higher layer. Note that active nodes at the edges of the base layer will not be communicating with higher-layer nodes because there is no node present at the edges of the higher layer due to the pyramid-like structure of the EdgeHGN subnets.

- *Stage 3:* The GN nodes at the layer above the base that receive a signal message containing the index of the bias entry that has been created or recalled from Stage 2 will be activated. A similar process as in Stages 1 and 2 will occur. However, the contents of the signal messages from the preceding and succeeding columns will be in the form of $\mathbf{bias_{ent}}$**(left , middle , right**) for non-edge nodes and either $\mathbf{bias_{ent}}$**(left , middle**) or $\mathbf{bias_{ent}}$**(middle , right**) for edge nodes. The values for *left, middle* and *right* are derived from the indices retrieved from the lower-layer nodes. For instance, left is for the preceding GN node's index received from its lower-layer counterpart. After the message communication between adjacent nodes has been completed, the active GNs will update their bias arrays and send the store/recall index/indices to the node at the same position in the higher layer (except for the GNs at the edges). This stage will be repeated for each layer above the base layer until it reaches the top-layer GN nodes.
- *Stage 4:* One of the top-layer GNs will receive a bias index from a GN in the layer underneath it. This top-layer activated node will search its bias array for the index. If the index is found, this node will trigger a recall flag with the recalled index. Otherwise, it will trigger a store flag and store the new index in its bias array. It will then send a signal message to the SI module with the message format **{subnet_id , status , index}**, where the status is either recall or store. The signal message sent by the top-layer active GN marks the completion of the recognition at the sub-pattern level. In an EdgeHGN implementation, lower bias arrays are updated whenever a new entry is found.

3.3.1.2 Pattern Reconstruction and Recognition Level

Recognition results obtained by the SI module from all subnets within an EdgeHGN network require further analysis to derive an overall recall of respective input sub-

patterns. In accommodating this analysis, two different methods have been considered, namely recall percentage and voting methods. These two methods differ in terms of the mechanism being adopted. The following section compares and contrasts these two approaches from an accuracy perspective.

***Recall percentage method**:* The recall percentage method underlines the use of bias indices obtained from all GNs within each subnet. The main principle of this approach is that the recall/store decision is mainly based on the cumulative decisions of all GNs within the network. This method requires an additional procedure to be conducted by each EdgeHGN subnet for index collection before the final recognition result is submitted to the SI module. For each sub-pattern introduced into the subnet, and after all recognition processes have been completed, the activated top GN will collect all index information from all GNs underneath it. These indices will then be compiled and structured with the format (**index:count**). The outputs will then be sent to the SI module using the message format {**subnet_id ,** (**$index_1$:$count_1$**) **,** (**$index_2$:$count_2$**) **, ... ,** (**$index_n$:$count_n$**)} for all n indices recalled or stored. Some of the advantages of recall percentage implementation for recognition at the pattern level include its high recall value precision in terms of the percentages of pattern indices being recalled. In this context, for a given input pattern, the EdgeHGN can present its precise recall value. The EdgeHGN also has the capability to analyse pattern composition based on the previous input patterns that have been stored within the network. However, the recall percentage method comes with a number of limitations. These include its effect on EdgeHGN recognition accuracy, where a slight change in the structure of the sub-pattern, due to distortion or noise, will affect the index calculation of the entire subnet. The recall percentage method also raises an issue of the level of confidence of the outputs of the system.

***Voting method*:** In the domain of pattern recognition, there has been a recent movement towards combining the decisions of several classifiers to arrive at improved recognition results. Most of the existing pattern recognition schemes apply the rejection technique to remove highly distorted patterns in the classification

procedure. This technique adopts the rejection/accuracy rate as a parameter to indicate levels of similarity of patterns. The technique offers a precise mean to obtain a good classification measurement. However, it is mostly suitable for deployment within a single-decision system in which the classification is conducted using a single classifier/recogniser. With a move towards distributed pattern recognition and/or classification, an important decision-making mechanism is needed to combine all decisions (in terms of accuracy/rejection) made by each classifier. A possible method for combining decisions on classification is the voting method. There are several forms of voting available in the literature, including majority, common-consent, unison and unanimity voting (Battiti & Colla, 1994; Kuncheva, 2004). Among all of the voting combination methods, majority voting is by far the simplest for implementation. It does not assume prior knowledge of the behaviour of the individual classifiers, and it does not require training on large quantities of representative recognition results from the classifiers. In EdgeHGN implementation, majority voting is used to obtain a combined decision on the recalls made by each of the subnets within a recognition network. The majority voting concept that has been adopted for EdgeHGN implementation follows the work by (Cruz et. al., 2007). For each recognition process, the decision of whether the input pattern has been recognised (i.e., recall) or is new to the network (i.e., store) is determined by obtaining majority consent from all EdgeHGN subnets. From this perspective, for a pattern to be recalled, the network should confirm that most of the sub-patterns belong to the respective input pattern. In this pattern reconstruction and recognition process, the SI module will initially receive all of the results of the recognition at the sub-pattern level from all of the EdgeHGN subnets. After all of these messages have been received, the actual recognition process is conducted. There are two stages involved at this level:

- *Stage 1*: All of the indices received from the EdgeHGN subnets for original patterns are stored in a two-dimensional vector matrix $\boldsymbol{M} = \{\mathbf{m}_{11}, \mathbf{m}_{12}, \ldots, \mathbf{m}_{sn}\}$. The width of the matrix is equivalent to the size of the pattern, $\boldsymbol{s}$, while the height corresponds to the number of stored patterns, $\boldsymbol{n}$.

- *Stage 2:* Calculate the frequency of the indices for each test pattern. All of the indices for the test pattern are stored in a vector $\mathbf{V} = \{\boldsymbol{\nu_1, \nu_2, ..., \nu_s}\}$. The width of the matrix is also equivalent to the size of the pattern. If an entry in vector **V** gives the list of indices as {1, 2, 2, 2, 1}, this indicates that three subnets have given a recall result of pattern 2 while two subnets have given a recall result of pattern 1. Therefore, by using the voting approach, the pattern will be recalled as pattern 2.

Consider that $\boldsymbol{\mathcal{P}}$ is an array of stored patterns $\boldsymbol{\mathcal{P}} = \{\boldsymbol{\rho_1, \rho_2, ..., \rho_n}\}$, where $\boldsymbol{n}$ represents the number of patterns being stored. For any pattern $\boldsymbol{\rho_x}$ to be recalled, maximum vote $\mathbf{V}_{\mathbf{max}}^{\boldsymbol{\rho x}}$, should be obtained using the following equation:

$$\mathbf{V}_{\mathbf{max}}^{\boldsymbol{\rho x}} = \mathbf{arg\,max}\,(\boldsymbol{\nu_x}) \quad , \quad \mathbf{x} \in \boldsymbol{n} \tag{3.3}$$

Where $\boldsymbol{\nu_x}$ represents the voting element of pattern $\boldsymbol{\rho_x}$ in voting vector $\mathbf{V}_{\boldsymbol{\rho}}$. It is worth noting that in the EdgeHGN approach, the recognition process for each pattern occurs in a single-cycle containing a fixed number of steps.

3.3.2 Bias Array Design

In EdgeHGN implementation, patterns are stored in the form of associations between its elements. This is somewhat different from other neural network approaches, in which patterns are stored as the composition of values. The pattern storage mechanism adopted by the EdgeHGN is in the form of a bias array, similar to the techniques used by the GN and HGN approaches as described in Chapter 2. However, the bias array capacity for the EdgeHGN minimises the storage requirement for input patterns because the bias array design limits the growth of storage elements within each GN through the use of the **index**(**left, right**) format of bias entry for one-dimensional input patterns. Moreover, the application of the drop-fall pre-processing stage reduces the number of GN nodes within each subnet significantly, resulting in minimised bias array size for pattern storage. For example, consider a comparison between an EdgeHGN bias entry and feed-forward neural network storage requirement capacities for each neuron, given different binary pattern sizes used in

the networks. In the feed-forward network, each neuron requires input from all of the elements within a particular pattern. Given a pattern $\boldsymbol{\rho}$ with $\boldsymbol{n}$ input elements (i.e., size) and $\boldsymbol{d}$ dimension, each neuron must be able to memorise $\boldsymbol{d^n}$ combinations of patterns. Conversely, the EdgeHGN only requires a maximum $\boldsymbol{2d}$ storage capacity for each neuron for memorisation. In this perspective, the EdgeHGN offers significantly higher storage efficiency compared to the feed-forward neural network. A further evaluation of the storage capacity of the EdgeHGN will be discussed in later sections.

3.4 EdgeHGN Communication Framework

Each EdgeHGN subnet is derived from a composition of interconnected GNs. The size of the subnet depends on the number of different elements in the sub-pattern and the size of the sub-pattern after applying the drop-fall scheme. Therefore, to define the size of each subnet, we consider the number of GNs $\boldsymbol{n}$ required for a sub-pattern of size $\boldsymbol{s}$, composed of $\boldsymbol{e}$ different elements given by the following equation:

$$\boldsymbol{n} = \boldsymbol{e}\left(\frac{\mathbf{s}+\mathbf{1}}{\mathbf{2}}\right)^{\mathbf{2}} \tag{3.4}$$

3.4.1 Network Generation

For the EdgeHGN scheme to perform the recognition of patterns, it must first be generated. Network generation involves the construction of the SI module node and a collection of EdgeHGN subnets. The SI module node is a control node that is responsible for managing the inputs and outputs among the EdgeHGN subnets. The distribution of EdgeHGN subnets within the network depends on the pattern decomposition by the SI module. Given a pattern vector $\mathbf{P} = \{\boldsymbol{\rho_1}, \boldsymbol{\rho_2}, \boldsymbol{\rho_3}, \ldots, \boldsymbol{\rho_n}\}$ of size $\boldsymbol{n}$, and sub-pattern length $\boldsymbol{l_{sub}}$ (after applying the drop-fall scheme), the number of EdgeHGN subnets $\boldsymbol{\mathcal{N}_{sub}}$ that needs to be generated is determined by the ceiling function in Equation (3.5):

$$\boldsymbol{\mathcal{N}_{sub}} = \left\lceil \frac{\mathbf{n}}{\mathbf{l_{sub}}} \right\rceil, \ \mathbf{sub} \leq \boldsymbol{n} \tag{3.5}$$

The GN nodes within an EdgeHGN subnet are structured hierarchically, where the number of GN layers, $\boldsymbol{l}_{\mathbf{GN}}$, required within an EdgeHGN subnet is given by:

$$\boldsymbol{l}_{\mathbf{GN}} = \frac{\mathbf{l}_{\mathbf{sub}} + \mathbf{1}}{\mathbf{2}} \tag{3.6}$$

At base layer $\boldsymbol{l}_{\mathbf{base}}$, the number of GNs generated, $\boldsymbol{\mathcal{N}}_{\mathbf{base}}$, is equal to the size of the sub-pattern multiplied by the number of different elements $\boldsymbol{e}$, $\boldsymbol{\mathcal{N}}_{\mathbf{base}} = \boldsymbol{l}_{\mathbf{sub}} \times \boldsymbol{e}$. At middle layer $\boldsymbol{l}_{\boldsymbol{i}}$, the number of nodes $\boldsymbol{\mathcal{N}}_{\boldsymbol{i}}$ varies according to the level of the layer $\boldsymbol{i}$ in the hierarchy, except for the top layer. Therefore, $\boldsymbol{\mathcal{N}}_{\boldsymbol{i}} = \boldsymbol{e}\ (\boldsymbol{l}_{\mathbf{sub}} - \mathbf{2}\boldsymbol{i})$. At the top layer $\boldsymbol{l}_{\mathbf{top}}$, the number of processing nodes required is equivalent to the number of different elements $\boldsymbol{e}$. Hence, $\boldsymbol{\mathcal{N}}_{\mathbf{top}} = \boldsymbol{e}$. In the network generation stage, the SI module is also responsible for initialising the EdgeHGN subnets. The initialisation involves the communication of possible input values to the base-layer GN nodes before the actual store/recall operations can start. The message communication between the SI module and base-layer GN nodes (within each EdgeHGN subnet) is conducted using a specific message communication protocol that has been developed for bitmap patterns. The SI module sends the possible input values to each EdgeHGN subnet using the instruction, message format. For example, if binary values are to be communicated, then the message would be **initialise(0,1)**. Each initialisation message received by the base layer GN nodes is used to coordinate the GN nodes within the base-layer, where each node represents a specific position.

3.4.2 EdgeHGN Communications

The EdgeHGN communications involve a message-passing mechanism, in which a single processing node communicates with other nodes to exchange messages. It is composed of two different types, namely macro- and micro-communication. In macro-communication, communication costs at the system level are taken into account (i.e., communications incurred between the SI module and the EdgeHGN subnets). Conversely, micro-communication deals with GN communications within a particular subnet for each pattern introduced into the system.

3.4.2.1 EdgeHGN Macro-communications

Macro-communication in the EdgeHGN occurs between the SI module node and either base- or top-layer GNs in each subnet. It occurs at three different phases:

Network generation phase: The SI module is responsible for communicating possible input values of the patterns to all base-layer GNs within EdgeHGN subnets. Equation (3.7) shows the number of messages that need to be communicated by the SI module to these GN nodes, $\mathcal{N}_{SI \rightarrow sub}$:

$$\mathcal{N}_{SI \rightarrow sub} = \mathcal{N}_{sub} \times l_{sub} \times e \tag{3.7}$$

In this equation, $\mathcal{N}_{sub}$ represents the number of available subnets. This equation is based on the assumption that all EdgeHGN subnets are of the same size. The messages communicated from SI module to each GN are in the form of (*instruction, message format*) as described earlier.

Pattern input phase: Upon the generation of all EdgeHGN subnets, the SI module starts performing a divide-and-distribute process on the input pattern, decomposing it into a number of sub-patterns according to the number of subnets available. Consequently, these sub-patterns will be sent to each subnet within the network. However, in the actual format, the SI module will communicate directly with each GN at the base layer of each EdgeHGN subnet. Hence, the number of messages communicated is similar to the number of messages in the network generation phase, as in Equation (3.7).

Result communication phase: After the recognition process in each EdgeHGN subnet has been completed, the results (in terms of recall or store) will be communicated back to the SI module for further analysis. In this communication, messages in the form of (**subnet_id , status , index**) will be sent to the SI module by all top-layer GNs of each subnet. In regards to the communication cost, the total number of messages communicated from the subnets to the SI module, $\mathcal{N}_{Sub \rightarrow SI}$ is equivalent to the number of subnets available, $\mathcal{N}_{sub}$. Hence, $\mathcal{N}_{Sub \rightarrow SI} = \mathcal{N}_{sub}$.

3.4.2.2 EdgeHGN Micro-communications

The following relations describe the micro-communications involved between GNs within each EdgeHGN subnet (***e***: number of different elements, ***s*****:** sub-pattern size):

Base layer: For each GN in the base layer, the number of message communications incurred could be derived from the number of messages communicated between adjacent neurons for each input sub-pattern. For GNs at the edge of the base layer, the number of communication exchanges is equivalent to the number of different elements within the sub-pattern. For non-edge GNs, the communication is required between adjacent neurons in both the preceding and succeeding columns, as well as the communication of bias indices to the GNs at the next higher layer. In this context, the number of message exchanges is $\boldsymbol{e^2+1}$. The cumulative communication costs involved for each input recognition process for all GNs in the base layer of an EdgeHGN subnet is derived from the following equation:

$$\mathcal{N}_{\text{base}} = ((\boldsymbol{e}^2+1)\ (\boldsymbol{s}-2) + 2\boldsymbol{e}) \tag{3.8}$$

Middle layers: While the communication costs for GNs in the middle layers are similar to those for the base layer, the difference is in the number of nodes available within each layer. For each middle layer $\boldsymbol{i}$, where $1 \leq \boldsymbol{i} \leq \text{top}-1$, the number of message exchanges that occurs for sub-pattern recognition could be derived as:

$$\mathcal{N}_{\boldsymbol{i}} = ((\boldsymbol{e}^2+1)\ (\boldsymbol{s}-(2\boldsymbol{i}+2)) + 2\boldsymbol{e}) \tag{3.9}$$

Equation (3.8) presents the cumulative communication costs for all GNs in the middle layers:

$$\mathcal{N}_{\text{i}}^{\text{total}} = \sum_{\text{i}=1}^{\text{top}-1} ((\boldsymbol{e}^2+1)\ (\boldsymbol{s}-(2\boldsymbol{i}+2)) + 2\boldsymbol{e}) \tag{3.10}$$

Top layer: These GN nodes are only responsible for communicating the final index for each sub-pattern stored/recalled to the SI module. The costs for communicating these indices have been included in the macro-communication evaluation.

3.5 EdgeHGN Algorithms and Functions

The original EdgeHGN implementation comprises four important functions: *the SI module, voting, adjacency comparison and bias calculation*. The SI module function deals with how patterns are communicated from the SI node to all other GN nodes within all subnets. Three distinguished commands have been used, namely '*init*', '*store*' and '*abort*'. These represent initialisation, recall/store and abort processes, respectively. This function communicates directly with each subnet via the base-layer GNs (see Algorithm 3.1).

Algorithm 3.1: SI Module Function

Determine the top GN id from sub-pattern size $\mathbf{GN_{top}} = \mathbf{2} \times \left(\frac{size+1}{2}\right)^2$

```
repeat
    read input from file
    broadcast input to all GN
    if command.input equals command.init then
        get next input from file
    else if command.input equals command.store then
        for i = 1 to GN(top) do
            get msg from GN(i)
            listen for MPI messages from each GN
            if i == GN(top) then
                idx.input = msg.GN(top)
                return idx.input
            end if
        end for
        invoke VOTING function
        get next input from file
    else if command.input equals command.abort then
        abort
    end if
until end of file
```

The voting function implements the voting procedure for the results of the recognition performed at the sub-pattern level. Each subnet will communicate each index retrieved, **idx**, to the SI module node. The SI module node will then perform this function (see Algorithm 3.2).

Algorithm 3.2: Voting Function

```
for i = 0 to max.pattern do
    for j = 0 to max.subnet do
        get idx[i][j]
    end for
end for
for i = max.storage to max.pattern do
    for j = 0 to max.subnet do
        for k = 0 to max.storage do
            if idx[i][j] == idx[k][j] then
                idx[i].count[k] + +
            end if
        end for
    end for
end for
for i = max.storage to max.pattern do
    for j = 0 to max.storage do
        find max idx[i].count[k]
        return max idx[i].count[k]
    end for
end for
```

The adjacency comparison function involves a process of communicating entries between adjacent GNs within each EdgeHGN subnet. Once a particular GN is activated, it will perform this function to obtain sub-pattern entries from adjacent nodes. This function produces bias entries for the bias calculation function. Note that in this pseudo-code, we implemented binary pattern recognition, in which each layer within the EdgeHGN subnet consists of two levels of GN. Level 0 corresponds to value 0, while Level 1 reflects value 1 in binary patterns (see Algorithm 3.3). The bias calculation function performs the bias matching process within the bias array structure of each GN node. Each bias entry received will be matched with stored entries. If an entry is found, then the index will be recalled. Otherwise, a new index will be generated (see Algorithm 3.4).

Algorithm 3.3: Adjacency Comparison Function (Base Layer)

```
check GN position within subnet (size = sub-pattern length)
if edge.GN == left then
    send idx.bias.GN(this) to
        GN + 1 and GN + (size + 1)
    receive idx.bias.GN from
        either GN + 1 or GN + (size + 1)
else if edge.GN == right then
    send idx.bias.GN(this) to
        GN - 1 and GN + (size - 1)
    receive idx.bias.GN from
        either GN - 1 or GN + (size - 1)
else if edge.GN == middle then
    send idx.bias.GN(this) to
        GN + 1 and GN + (size + 1)
    send idx.bias.GN(this) to
        GN - 1 and GN + (size - 1)
    receive idx.bias.GN from
        either GN + 1 or GN + (size + 1)
    receive idx.bias.GN from
        either GN - 1 or GN + (size - 1)
end if
```

Algorithm 3.4: Bias Calculation Function

```
receive entry.pattern from ADJACENCY COMPARISON function
for all test.pattern in pattern do
    for i = 0 to MAXBIAS do
        if entry.pattern == element.bias[i].GN then
            idx.pattern.GN = i
            exit FOR
        else
            idx.pattern.GN = MAXBIAS + 1
        end if
    end for
    return idx.pattern.GN
end for
```

3.6 EdgeHGN Time Complexity and Scalability Analysis

A series of analysis for EdgeHGN implementation were conducted. These evaluations focus on the recall time, complexity and scalability of the algorithm.

3.6.1 Time Complexity

Table 3.1 represents the terms that we will use to estimate the time complexity:

Table 3.1: EdgeHGN total recall time complexity terms

Symbol	Explanation
t_c	*Communication time*: the time it takes for the network to send or receive a message from a GN
t_i	*Interaction time*: the time it takes to send messages between nodes within the bias array
t_s	*Searching time*: the time it takes to search a bias entry in the array per entry
Δt	*Overhead time*: it is a small time overhead per node due to hardware latency or the time required for a node to parse a message
t_a	*Access time*: the time required for a node to read or write a bias entry in the array
s	Sub-pattern size
n_r	Number of rows of GN nodes within a layer.
n_l	Number of layers with an EdgeHGN subnet
n_c^i	Number of columns at the layer level i
n_c^0	Number of columns at the base level in the subnet, equal to sub-pattern size s
n_b^i	Number of entries in the bias array in a node at the layer level i
n_b^0	Number of entries in the bias array in a node at the base layer
n_e^i	Number of entries in the bias array in a node at an edge of the layer level i
n_b^t	Number of entries in the bias array of a top level node
n_t^b	Number of total bias entries in all bias arrays for an EdgeHGN subnet
$t_1^i, t_2^i, t_3^i, t_{t4}^i$	Times taken by each of the four stages of EdgeHGN sub-pattern recognition at level i
T_{total}^i	Total time taken for performing EdgeHGN sub-pattern recognition stages at layer i
T_{total}	Total time taken to perform all recognition stages of an EdgeHGN subnet

For a scalable pattern recognition scheme, especially when dealing with large datasets, the time complexity of the approach should not be heavily affected by an increase in the number of stored patterns. The recall time factor for the EdgeHGN distributed pattern recognition scheme could be determined from the total time it takes for the scheme to recall/store an input pattern. The following section represents the equations for calculating the time complexity of the EdgeHGN algorithm at different layers of the scheme.

Base layer (0):

$$t_1^0 = n\, n_c^0\, (t_c + \Delta t) \tag{3.11}$$

$$t_2^0 = 2n\, (n_c^0 - 1) \times (t_l + \Delta t) \tag{3.12}$$

The term $(n_c^0 - 1)$ factor instead of n_c^0 is due to the fact that both nodes at the edges each send messages to all nodes in one column only. It is assumed that the search happening within the bias array is a *binary search*. Thus, the terms $\log_2(n_b^0) - \log_2(e)$ and $\log_2(n_e^0) - \log_2(e)$ appear as the average number of steps required to perform a binary search for an entry in the bias array.

$$t_3^0 = (n_c^0 - 2)\,(t_s\,[\log_2(n_b^0) - \log_2(e)] + \Delta t + t_\alpha) + 2\,(t_s\,[\log_2(n_e^0) - \log_2(e)] + \Delta t + t_\alpha) \tag{3.13}$$

$$t_3^0 = t_s\,\{n_c^0\,[\log_2(n_b^0) - \log_2(e)] - 2\log_2(n_r)\} + n_c^0\,(\Delta t + t_\alpha) \tag{3.14}$$

The following equation shows how to obtain the average number of steps:

$$\#steps = \sum_{i=1}^{n} \rho(i) \log_2(i) \tag{3.15}$$

The term $\rho(i)$ represents the probability of entry index i being accessed, $\log_2(i)$ represents the number of steps (worst case) to access the bias entry index i, and n represents the number of entries in the bias array. Assuming that the access distribution of entries is uniform, then the value of $\rho(i)$ can be replaced by $1/n$. The equation can therefore be modified as follows:

$$\#\mathbf{steps} = \frac{1}{\mathrm{n}}\sum_{\mathrm{i}=1}^{\mathrm{n}} \log_2(i) \approx \frac{1}{\mathrm{n}} \int_1^{\mathrm{n}} \log_2(\mathrm{i})\, di \approx \frac{1}{n\ln(2)} \int_1^{\mathrm{n}} \ln(i)\, di \tag{3.16}$$

$$\#\mathbf{steps} \approx \frac{\ln(n)}{\ln(2)} - \frac{1}{\ln(2)} + \frac{1}{n\ln(2)} \approx \log_2(n) - \log_2(e)\,(1 - \frac{1}{n}) \tag{3.17}$$

$$\mathbf{for} \;\; >> 0 \;\; \rightarrow \;\; \#\mathbf{steps} \approx \log_2(n) - \log_2(e) \tag{3.18}$$

It is evident that the average number of steps of the binary search here would be considerably less than $\mathbf{0.5}n$ of the exhaustive sequential search.

$$t_4^0 = n_r\,(\mathrm{n_c^0} - 2)\,(t_i + \Delta t) + \mathrm{n_c^0}\,(t_c + \Delta t) \tag{3.19}$$

For the sake of simplicity, the functionality of each node is simulated within a thread, and the communication among the threads within the network is implemented using sockets. As a result, it is a safe assumption that the value of t_i and t_c are very close to each other. For the remaining equations we simply replace them both with t_e.

$$\mathcal{T}_{\mathbf{total}}^0 = t_1^0 + t_2^0 + t_3^0 + t_4^0 = (4n_r n_c^0 - 4n_r + n_c^0)\,(t_e + \Delta t) + t_3^0 \tag{3.20}$$

Middle layer (i):

For the next round of iterations at middle layer i, only t_2^{i} , t_3^{i} and t_4^{i} are the contributors to the total value of $\mathcal{T}_{\mathbf{total}}^i$. As a result, Equations (3.12), (3.14) and (3.19) are all still valid for middle layer i as long as appropriate values of n_c^i and n_b^i are used for the relevant level index i.

$$t_2^{\mathrm{i}} = 2n_r\,(\mathrm{n_c^i} - 1) \times (t_e + \Delta t) \tag{3.21}$$

$$t_3^{\mathrm{i}} = t_s\{\,\mathrm{n_c^i}\,[\,\log_2(\mathrm{n_b^i}) - \log_2(e)\,] - 2\log_2(\mathrm{n_r})\,\} + \mathrm{n_c^i}\;(\Delta t + t_\alpha) \tag{3.22}$$

$$t_4^{\mathrm{i}} = n_r\,(\mathrm{n_c^i} - 2)\,(t_e + \Delta t) + \mathrm{n_c^i}\,(t_e + \Delta t) \tag{3.23}$$

$$\mathcal{T}_{\mathbf{total}}^i = t_2^{\mathrm{i}} + t_3^{\mathrm{i}} + t_4^{\mathrm{i}} = (3\mathrm{n_r}\,\mathrm{n_c^i} - 4\mathrm{n_r} + \mathrm{n_c^i})\,(t_e + \Delta t) + t_3^i \tag{3.24}$$

Top layer (n_l-1):

It is worth noting that at the top level $t_2^{n_l-1} = 0$.

$$t_3^{n_l-1} = t_s\,[\,\log_2(n_b^t) - \log_2(e)\,] + (\Delta t + t_\alpha) = \quad (3.25)$$
$$t_s\,[\,\log_2(s) - \log_2(n_r) - \log_2(e)\,] + (\Delta t + t_\alpha)$$

$$t_4^{n_l-1} = (t_e + \Delta t) \quad (3.26)$$

$$\mathcal{T}_{total}^{n_l-1} = t_3^{n_l-1} + t_4^{n_l-1} = t_s\,[\log_2(s) - \log_2(n_r) - \log_2(e)] + (2\Delta t + t_\alpha + t_e) \quad (3.27)$$

On the basis of Equations (3.11) to (3.27), we can now calculate the total time it takes for an EdgeHGN subnet to perform recognition across all stages of all layers of a subnet. Given that EdgeHGN subnets are processed in a purely independent parallel fashion using the single-cycle (one-shot) learning approach, this time calculation can reasonably stand for the time that it takes for the EdgeHGN to process an input pattern through the divide-and-distribute processing mechanism of its subnets.

$$\mathcal{T}_{total} = \mathcal{T}_{total}^{0} + \sum_{i=1}^{n_l-2} \mathcal{T}_{total}^{i} + \mathcal{T}_{total}^{n_l-1} \quad (3.28)$$
$$= \mathcal{T}_{total}^{0} + \sum_{i=1}^{n_l-2} [(3n_r n_c^i - 4n_r + n_c^i)(t_e + \Delta t) + t_3^i] + \mathcal{T}_{total}^{n_l-1}$$

As a result of the above calculations, the EdgeHGN recall time complexity is **O(n)**, which clearly demonstrates the strength of this approach in providing a fast and low-complexity scheme for large-scale data analysis. An important aspect of the EdgeHGN implementation is the ability of the scheme to perform the recognition procedure within a single-cycle pass, without having to conduct an iterative training procedure to train the network for adaptation purposes. Rather, the EdgeHGN performs in situ recognition in which the training set can be memorised within a single pass (or cycle). This gives an edge to the EdgeHGN as a solution for large-scale pattern recognition.

The estimated recall time of an EdgeHGN network for processing 10,000 patterns (using Equation (3.28)) and the EdgeHGN actual recall time for processing the same number of patterns is plotted in Figure 3.8. As shown, the experimental findings closely match the estimated plot. *The flat slope in both figures demonstrates that the EdgeHGN response time remains consistently insensitive to the volume of processed patterns*. The spikes in actual timing of the EdgeHGN network processing is due to the time difference occurring in forming EdgeHGN subnets after applying Dropfall scheme. As a result, the EdgeHGN as a scalable associative memory framework is a suitable choice for processing large volumes of data.

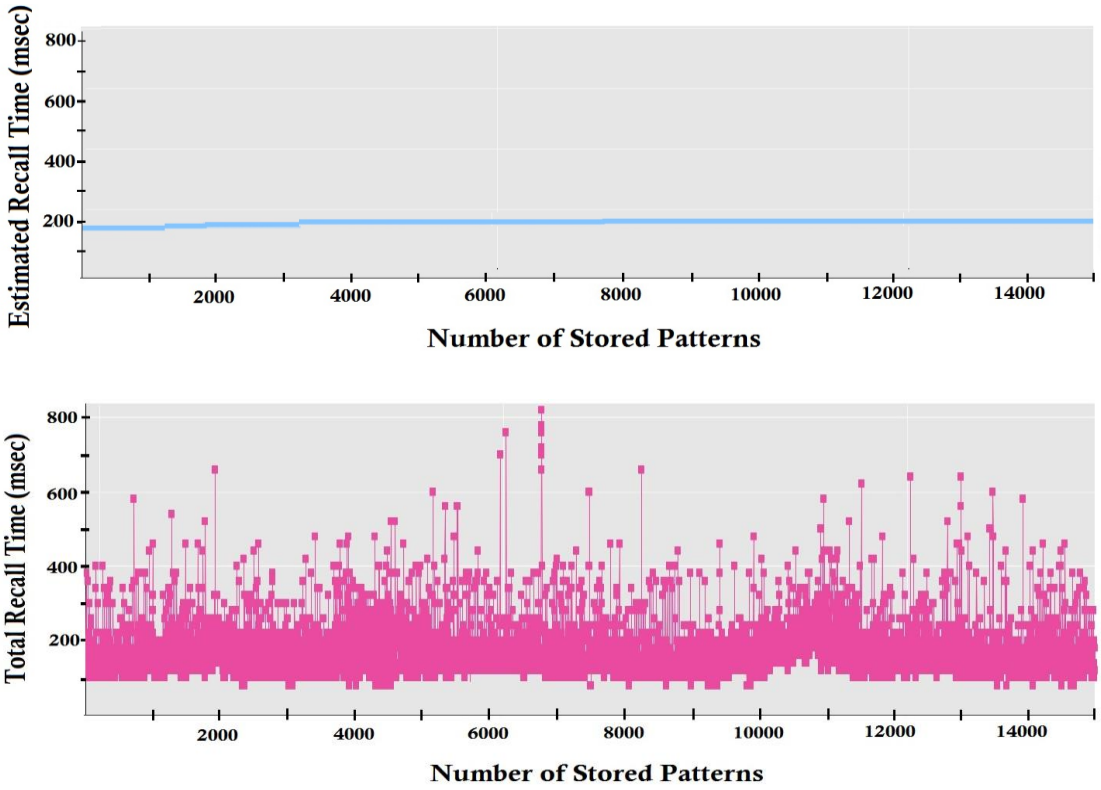

Figure 3.8: EdgeHGN estimated and actual recall times
for processing 10,000 stored patterns

3.6.1.1 Recall Time Comparative Study

To further demonstrate the efficient time complexity of the EdgeHGN algorithm, the two-stage process of network generation and recognition are considered, and

comparisons have been made with two different algorithms: The Hopfield network and Kohonen Self-Organizing Map. However, it should be noted that the comparative study that has been carried out here does not intend to outweigh the capabilities of these algorithms. Rather, it indicates that the EdgeHGN has the capacity to acquire significantly low computational complexity for its operations.

EdgeHGN v. Hopfield

Network generation stage: This stage involves the formation of a network that comprises computing elements known as neurons. Table 3.2 shows the details of the Big-O notation derived for the Hopfield network and EdgeHGN implementations. The estimated time derived is based on the assumption that the instruction speed used is **1 microsecond (*μsec*)** per instruction. In Hopfield network implementation, the number of neurons generated is equivalent to the size of the pattern. On the other hand, for EdgeHGN the number of neurons generated during network generation phase is equivalent to the number of neurons that are initialized within base layer of each EdgeHGN subnet as shown in equation 3.29 (***e*:** number of elements within the pattern, **P:** pattern size, ***n***: number of EdgeHGN subnets**).**

$$\mathcal{N}_{\mathbf{EdgeHGN}}^{\mathbf{Init}} = \boldsymbol{e} \times \mathbf{P} \times \boldsymbol{n} \tag{3.29}$$

Table 3.2: Big-O notations for Hopfield and EdgeHGN schemes in the network generation stage (Hopfield network, 2012)

Algorithm	Big-O	Efficiency	Iterations(n)	Estimated Time (sec)
EdgeHGN	**O(*n*)**	Linear	$\mathcal{N}_{\mathbf{EdgeHGN}}^{\mathbf{Init}}$	$\mathcal{N}_{\mathbf{EdgeHGN}}^{\mathbf{Init}} \times \mathbf{0.000001}$
Hopfield	**O(*n*)**	Linear	**P** (pattern size)	$\mathbf{P} \times \mathbf{0.000001}$

The results show that both the EdgeHGN and Hopfield networks acquire comparable computational complexity. However, in regards to the number of neurons generated, the EdgeHGN incurred higher complexity. Taking parallelism into

account, for each EdgeHGN subnet, the number of neurons generated was less than the overall neuron initialisation within the network. Hence, the estimated time for network generation in EdgeHGN will be lower compared with the Hopfield approach.

Recognition stage: The recognition stage is the core process within the pattern recognition application. Each algorithm uses a different approach to handle this process. In the Hopfield network, the recognition stage involves three sub-processes, namely weight accumulation, weight determination for the whole network and network propagation to derive an optimum solution. In contrast, EdgeHGN algorithm only implies a single-cycle process of recognition within the recognition stage. This process of recognition involves either store or recall process. Table 3.3 shows the Big-O notations derived from the analysis on the Hopfield network recognition process. Similarly, this is based on the assumption that the instruction speed used is **1 microsecond (*μsec*)** per instruction. The Hopfield network incurs a considerably high computational complexity, as indicated in Table 3.3, with respect to its weight determination and network propagation processes. The recognition stage for pattern recognition using the EdgeHGN algorithm involves a single-cycle process in which each input pattern will be passed through the EdgeHGN subnets once, and the store or recall process will be activated according to the instruction given.

Table 3.3: Big-O notations for the Hopfield and EdgeHGN networks in the recognition stage (Hopfield network, 2012)

Process	Big-O	Efficiency	Iterations(n)	Estimated Time (sec)
Weight accumulation	$O(n)$	Linear	$\mathbf{P}$	$\mathbf{P \times 0.000001}$
Weight determination	$O(n^2)$	Quadratic	$\mathbf{P^2}$	**Minutes**
Network propagation	$O(n^K)$	Polynomial	$\mathbf{P^k}$	**Hours**
EdgeHGN	$O(n)$	Linear	$\mathcal{N}_{\mathbf{EdgeHGN}}^{\mathbf{Init}}$	$\mathcal{N}_{\mathbf{EdgeHGN}}^{\mathbf{Init}} \times \mathbf{0.000001}$

Table 3.3 shows the Big-O notation for the recognition stage using the EdgeHGN algorithm. From the Big-O notations derived from the analysis, it is best to conclude that the *EdgeHGN incurs less computational complexity in pattern recognition processes as compared to the Hopfield network implementation*. *Specifically, the EdgeHGN employs a simple linear function, whereas the Hopfield network employs expensive polynomial and quadratic functions.*

EdgeHGN v. Kohonen SOM

The Big-O notations for both the SOM and EdgeHGN have been estimated to study their complexity levels. The supervised SOM consists of three important stages: (*i*) *weight initialisation*, (*ii*) *BMU calculation*, and (*iii*) *weight adjustment*. In the weight initialisation stage, nodes are created with a random assigned weight. At this stage, the computational complexity depends heavily on the number of created nodes. Hence, for a given weight initialisation process $\boldsymbol{w}$, the complexity of $\boldsymbol{n}$ nodes can be simplified as $\boldsymbol{f(w)} = \mathbf{O}(\boldsymbol{n}^3)$. In the BMU calculation stage, the complexity depends heavily on the number of iterations during training as well as the number of the input vector. Hence, for a given BMU calculation process $\boldsymbol{\rho}$, the complexity of $\boldsymbol{\pi}$ training iterations can be simplified as $\boldsymbol{f(\rho)} = \mathbf{O}(\boldsymbol{\pi}^4)$. In the last stage, the weight adjustments are provided not only for the winning neuron, but also for its neighbours in a certain neighbourhood.

The Big-O for the weight adjustment is similar to the BMU calculation. Conversely, the EdgeHGN initialisation stage, as discussed previously, is a low-computational process, and hence acquires less computational time in comparison to the SOM's weight initialisation process. The computational complexity for the classification process is somewhat similar to the network generation process. The EdgeHGN classification process requires less computational complexity in comparison to the SOM's BMU calculation and weight adjustment activities. *In summary, the estimated time graph of the EdgeHGN algorithm is linear, while the corresponding graph of the SOM algorithm is exponential. This proves that the EdgeHGN provides a lightweight and fast algorithm comparable to the SOM*.

3.6.2 Scalability Analysis

Table 3.4 presents the terms used in this thesis to estimate the scalability analysis of the EdgeHGN algorithm.

Table 3.4: EdgeHGN storage and communication complexity terms

Symbol	Explanation
$\mathcal{S}_{\text{Sub}}$	Size of sub-pattern
$\mathcal{N}_{\text{SI}\rightarrow\text{Sub}}$	Number of message between SI module and GN nodes in the network generation stage
$\mathcal{N}_{r}$	Number of rows of GN nodes within a layer
$\mathcal{N}_{s}$	Number of EdgeHGN subnets within a network
$\mathcal{N}^{\text{Base}}_{\text{None}-\text{Edge}}$	Number of messages communicated from non-edge GN nodes in base layer
$\mathcal{N}^{\text{Base}}_{\text{Edge}}$	Number of messages communicated from GN nodes at the edge of base layer
$\mathcal{N}^{\text{Base}}_{\text{Total}}$	Total number of messages communicated from GN nodes at base layer
$\mathcal{N}^{\text{i}}_{\text{None}-\text{Edge}}$	Number of messages communicated from non-edge GN nodes at middle layer $\boldsymbol{i}$
$\mathcal{N}^{\text{i}}_{\text{Edge}}$	Number of messages communicated from GN nodes at the edge of middle layer $\boldsymbol{i}$
$\mathcal{N}^{\text{i}}_{\text{Total}}$	Total number of messages communicated from GN nodes at middle layer $\boldsymbol{i}$
$\mathcal{N}^{\text{Subnet}}_{\text{Total}}$	Total number of messages communicated between all GN nodes in subnet
$\mathcal{B}^{\text{Base}}_{\text{None}-\text{Edge}}$	Maximum size of bias array for each non-edge GN node at base layer
$\mathcal{B}^{\text{Base}}_{\text{Edge}}$	Maximum size of bias array for each GN node at the edge in base layer
$\mathcal{B}^{\text{Base}}_{\text{Total}}$	Total maximum bias array size for all GN nodes in base layer
$\mathcal{B}^{\text{i}}_{\text{None}-\text{Edge}}$	Maximum size of bias array for each non-edge GN node at middle layer $\boldsymbol{i}$
$\mathcal{B}^{\text{i}}_{\text{Edge}}$	Maximum size of bias array for each GN node at the edge in middle layer $\boldsymbol{i}$
$\mathcal{B}^{\text{i}}_{\text{Total}}$	Total maximum bias array size for all GN nodes in middle layer $\boldsymbol{i}$
$\mathcal{B}^{\text{Top}}_{\text{Total}}$	Maximum bias array size for all GN nodes in top layer
$\mathcal{B}^{\text{Subnet}}_{\text{Total}}$	Total maximum bias array size for all GN nodes in EdgeHGN subnet
$\mathcal{S}^{\text{Msgs}}_{\text{Net}}$	Size of messages in the network generation stage
$\mathcal{S}^{\text{Total}}_{\text{Net}}$	Total size of messages communicated in the network generation stage

The scalability factor for the EdgeHGN distributed pattern recognition scheme can be determined from two different aspects: *storage capacity and communication efficiency*. A high requirement for storage capacity would affect the scalability of the algorithm. For an efficient pattern recognition scheme, the storage requirement should not be heavily affected by an increase in the number of stored patterns, and the communication should stay relatively contention-free. Analyses have been conducted on the computational complexity of the EdgeHGN algorithm for pattern recognition. In doing so, the two computational factors mentioned previously: storage capacity and communication efficiency have also been considered.

3.6.2.1 Storage Capacity Analysis

Storage capacity estimation for the EdgeHGN algorithm involves the analysis of the bias array capacity for all GN nodes within the distributed architecture, as well as the storage capacity of the SI module node. In analysing the capacity of the bias array, the size of the bias arrays is observed as different patterns are being stored. The number of possible pattern combinations increases exponentially with an increase in the pattern size. The effect of the pattern size on the bias array storage is an important factor in the bias array scalability analysis. In this regard, the analysis is conducted by segregating the bias arrays according to the layers within a particular EdgeHGN subnet. The following equations show the bias array size estimation for binary patterns. In this analysis, an EdgeHGN implementation for one-dimensional binary patterns has been considered, wherein a two-dimensional pattern is represented as a string of bits.

Base layer: For each non-edge GN node, the maximum size of the bias array is:

$$\mathcal{B}^{\mathbf{Base}}_{\mathbf{None-Edge}} = (\mathcal{N}_r)^2 \tag{3.30}$$

For each GN node at the edge of the layer:

$$\mathcal{B}^{\mathbf{Base}}_{\mathbf{Edge}} = \mathcal{N}_r \tag{3.31}$$

The maximum size of the bias array is mostly determined by the number of possible combinations of values within a pattern. The cumulative maximum size of bias arrays at the base layer in each EdgeHGN subnet could be derived as shown in Equation (3.32):

$$\mathcal{B}_{\text{Total}}^{\text{Base}} = \mathcal{N}_r \left(\mathcal{B}_{\text{None-Edge}}^{\text{Base}} + (\mathcal{N}_r)^2 (\mathcal{S}_{\text{Sub}} - 2) + 2\mathcal{B}_{\text{Edge}}^{\text{Base}}\right) \tag{3.32}$$

Middle layers: The maximum size of the bias array at a middle layer depends on the maximum size of the bias array at the layer below it. For a non-edge GN node in a middle layer, the maximum size of its bias array may be derived as follows:

$$\mathcal{B}_{\text{None-Edge}}^{\text{i}} = \mathcal{B}_{\text{None-Edge}}^{\text{i-1}} \times (\mathcal{N}_r)^2 \tag{3.33}$$

For each GN node at the edge, the maximum size of the bias array can be given by:

$$\mathcal{B}_{\text{Edge}}^{\text{i}} = \mathcal{B}_{\text{None-Edge}}^{\text{i-1}} \times \mathcal{N}_r \tag{3.34}$$

Therefore, the cumulative maximum size of the bias arrays in a middle layer (of a subnet) can be estimated using the following equation:

$$\textbf{for } 1 \leq \text{i} \leq \text{top-1} \tag{3.35}$$

$$\mathcal{B}_{\text{Total}}^{\text{i}} = \mathcal{N} \left(\mathcal{B}_{\text{None-Edge}}^{\text{i}} (\mathcal{S}_{\text{Sub}} - (2\text{i} + 2)) + 2\mathcal{B}_{\text{Edge}}^{\text{i}}\right)$$

Top layer: At the top layer, the maximum size of the bias array can be derived from the preceding level non-edge GN node's maximum bias array size. Hence, the maximum size of the bias array of a GN node at the top level is:

$$\mathcal{B}_{\text{Total}}^{\text{Top}} = \mathcal{B}_{\text{None-Edge}}^{\text{i-1}} \times \mathcal{N}_r \tag{3.36}$$

From these equations, the total maximum size of all bias arrays within a single EdgeHGN subnet can be deduced as shown in Equation (3.37):

$$\mathcal{B}_{\text{Total}}^{\text{Subnet}} = \mathcal{B}_{\text{Total}}^{\text{Base}} + \sum_{\text{i=1}}^{\text{top-1}} \mathcal{B}_{\text{Total}}^{\text{i}} + \mathcal{B}_{\text{Total}}^{\text{Top}} \tag{3.37}$$

3.6.2.2 Communication Complexity Analysis

The EdgeHGN is a distributed pattern recognition algorithm. In any distributed algorithm, communication plays an important role in ensuring the efficiency of the algorithm. High communication costs will incur an additional overhead for the network to support the core functions of the algorithm. Hence, the intention is to minimise the communication costs within the EdgeHGN. In conducting an analysis of the communication costs, all four steps in the distributed pattern recognition scheme have been considered. This subsection estimates the communication costs for the implementation.

Network generation step: Network generation in the EdgeHGN implementation involves the initialisation of EdgeHGN subnets for recognition processes. Within this step, the SI module is responsible for communicating the possible input values of the patterns, which will be used in the recognition process, to all of the base-layer GN nodes within EdgeHGN subnets. Equation (3.38) shows the number of messages that need to be communicated by the SI module to these GN nodes:

$$\mathcal{N}_{\mathbf{SI}\rightarrow\mathbf{Sub}} = \mathcal{N}_{\boldsymbol{r}} \times \mathcal{N}_{\boldsymbol{s}} \times \mathcal{S}_{\mathbf{Sub}} \tag{3.38}$$

This equation is based on the assumption that all EdgeHGN subnets are the same size. In addition, the cumulative size of all messages that will be transmitted is shown in Equation (3.39):

$$\mathcal{S}_{\mathbf{Net}}^{\mathbf{Total}} = \mathcal{S}_{\mathbf{Net}}^{\mathbf{Msgs}} \left(\mathcal{N}_{\boldsymbol{r}} \times \mathcal{N}_{\boldsymbol{s}} \times \mathcal{S}_{\mathbf{Sub}}\right) \tag{3.39}$$

Pattern input step: As part of this step, the SI module is required to decompose the pattern into sub-patterns and distribute these sub-patterns to all available EdgeHGN subnets. The distribution of sub-patterns to all EdgeHGN subnets requires communication between the SI module and all base-layer GN nodes within the subnets. The communication costs incurred during this step are similar to those in the previous step.

Recognition at the sub-pattern level: The following relations show the communication complexity of EdgeHGN scheme at the sub-pattern recognition level.

Base layer*:* For each GN node in the base layer, the communication costs can be derived from the number of messages communicated between adjacent nodes for each input sub-pattern. For GN nodes at the edge of the base layer:

$$\mathcal{N}_{\mathbf{Edge}}^{\mathbf{Base}} = \mathcal{N}_{r} \tag{3.40}$$

For non-edge GN nodes:

$$\mathcal{N}_{\mathbf{None-Edge}}^{\mathbf{Base}} = (\mathcal{N}_{r})^{2} + 1 \tag{3.41}$$

Note that for non-edge GN nodes, communication is required between adjacent nodes in both the preceding and succeeding columns, as well as the communication of bias indices to the GN nodes at the next higher layer. The cumulative communication costs for all GN nodes in the base layer can be derived as:

$$\mathcal{N}_{\mathbf{Total}}^{\mathbf{Base}} = \mathcal{N}\ (\mathcal{N}_{\mathbf{None-Edge}}^{\mathbf{Base}}\ (\mathcal{S}_{\mathbf{Sub}} - 2) + 2\mathcal{N}_{\mathbf{Edge}}^{\mathbf{Base}}) \tag{3.42}$$

Middle layer*:* The communication costs for the GN nodes in the middle layers are similar to those for the GN nodes at the base layer. However, the difference is in the number of nodes available within each layer. The cumulative communication costs for all GN nodes in each middle layer can be derived as:

$$\textbf{for } 1 \leq \mathbf{i} \leq \textbf{top-1} \tag{3.43}$$

$$\mathcal{N}_{\mathbf{Total}}^{\mathbf{i}} = \mathcal{N}\ (\mathcal{N}_{\mathbf{None-Edge}}^{\mathbf{i}}\ (\mathcal{S}_{\mathbf{Sub}} - (2\mathbf{i} + 2)) + 2\mathcal{N}_{\mathbf{Edge}}^{\mathbf{i}})$$

Top layer*:* These GN nodes are only responsible for communicating the final index for each sub-pattern stored/recalled to the SI module. Therefore, there is only one message that needs to be passed to the SI module for each input sub-pattern. The total cumulative number of communications required for each sub-pattern stored/recalled in an EdgeHGN subnet can be derived from Equation (3.44) as:

$$\mathcal{N}_{\mathbf{Total}}^{\mathbf{Subnet}} = \mathcal{N}_{\mathbf{Total}}^{\mathbf{Base}} + \sum_{\mathbf{i=1}}^{\mathbf{top-1}} \mathcal{N}_{\mathbf{Total}}^{\mathbf{i}} + 1 \tag{3.44}$$

Recognition at pattern level: The recognition at the pattern level does not require any communication because recognition takes place within the SI module.

The EdgeHGN requires fewer hierarchical layers in comparison with the HGN and DHGN due to fewer processing neurons. This in turn minimises the overall communication cost, as shown in Figure 3.9.

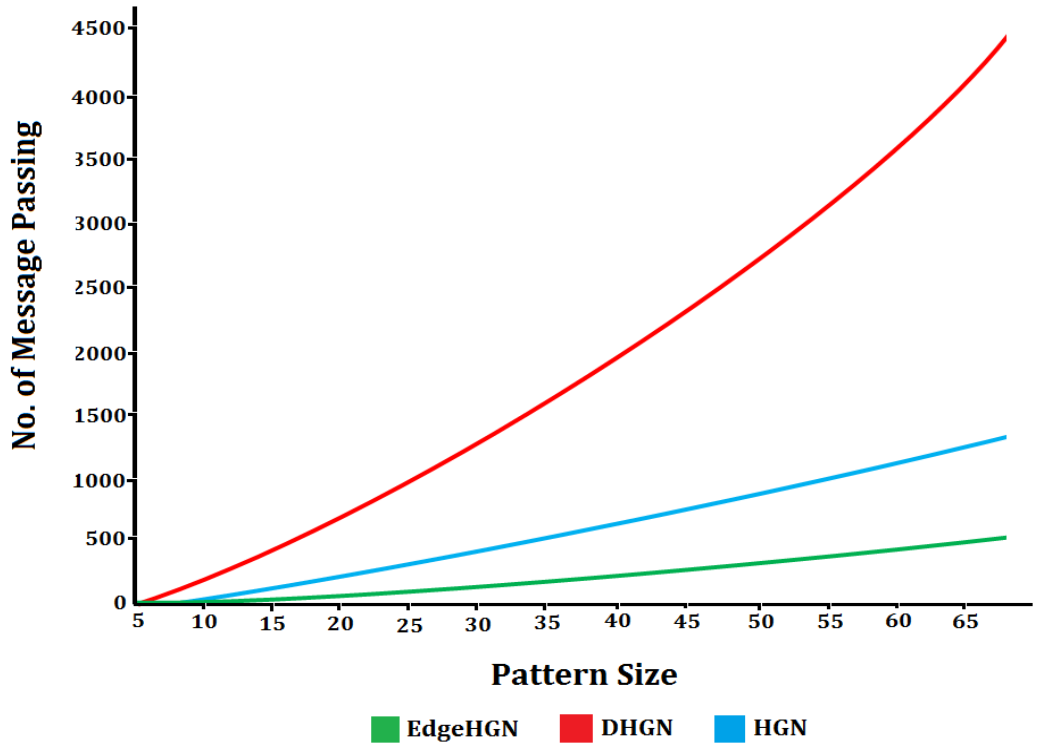

Figure 3.9: Comparison of communication costs between the HGN, DHGN and EdgeHGN (Khan & Muhamad Amin, 2007)

The figure compares the communication costs between the EdgeHGN, HGN and DHGN. The formation of the EdgeHGN subnets helps to improve the efficiency of the GN-based approach for pattern recognition. Further, the EdgeHGN provides high scalability towards increasing size and dimension of patterns through the use of the divide-and-distribute approach within single-cycle learning. The following section will describe these in more detail.

3.7 Pattern Recognition Simulation and Results

A series of recognition tests using the EdgeHGN's distributed pattern recognition approach have been conducted. In this chapter, two different sets of patterns have been tested and discussed. The first set of patterns is binary character patterns, while the second set is 16KB binary images. These sets were used as the base for generating noisy patterns in their respective categories. For this purpose, the EdgeHGN subnets, capable of storing either 5-bit or 9-bit binary patterns, were adopted for these tests. The subnet size may be calculated by dividing the largest input pattern size with the number of available nodes within the computer cloud. Inter-node communications and SI-to-subnet communications were implemented using the MPICH-2 library for the message-passing interface (MPI) (Gropp, Thakur & Lusk, 1999).

3.7.1 Binary Character Pattern Recognition

The character patterns used in this recognition test have been grouped into three different representations: 5-by-7 bit, 8-by-8 and 16-by-16. For this test, each EdgeHGN subnet is used to store/recall 5-bit binary sub-patterns. Each character image used has been decomposed by the SI module into 5-bit sub-patterns of various sizes. The recall rate using the precision and recall technique with a voting mechanism has been used as a classification parameter in these tests. Recall rates $\boldsymbol{\mathcal{R}}$, for the tests were obtained using the following equation, where both $\boldsymbol{\mathcal{N}}^{\mathbf{True}}_{\mathbf{EdgeHGN}}$ and $\boldsymbol{\mathcal{N}}^{\mathbf{False}}_{\mathbf{EdgeHGN}}$ represent the number of EdgeHGN subnets with correct and incorrect recalls respectively.

$$= \frac{\boldsymbol{\mathcal{N}}^{\mathbf{True}}_{\mathbf{EdgeHGN}}}{\boldsymbol{\mathcal{N}}^{\mathbf{False}}_{\mathbf{EdgeHGN}} + \boldsymbol{\mathcal{N}}^{\mathbf{True}}_{\mathbf{EdgeHGN}}} \tag{3.45}$$

Figure 3.10 shows the comparison of the recall rates among the three character patterns $\boldsymbol{\mathcal{A}}$ of different sizes, which have been used in this test. The recall rates are similar for all different representations and indicates that EdgeHGN can recognise distorted character patterns with up to 20% distortion,

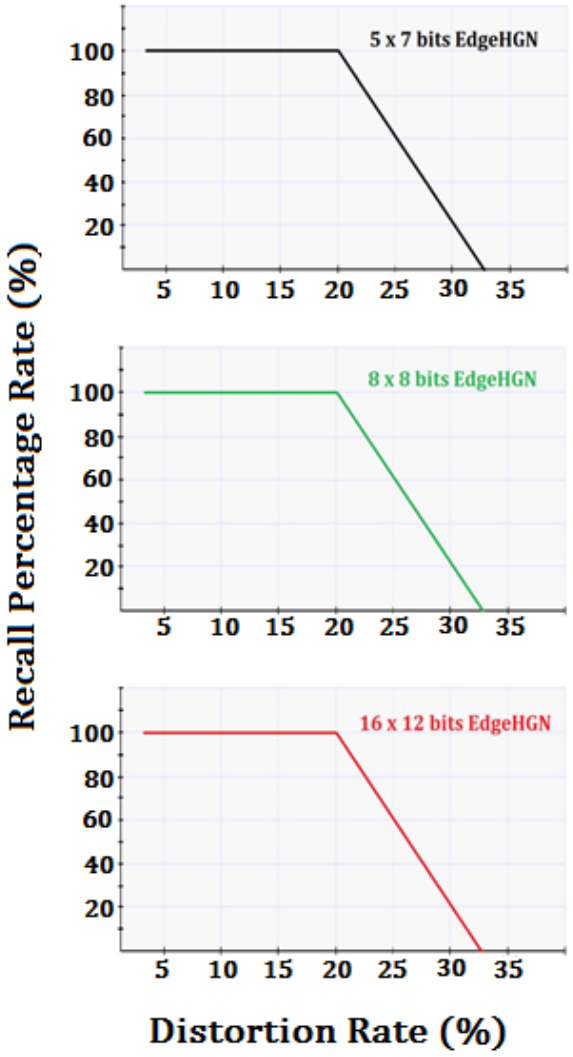

Figure 3.10: EdgeHGN recall percentage for the three Character patterns ***A*** of different sizes

The recognition test has also been conducted on a set of binary character patterns with random distortion. The random distortion applied to the character patterns varies according to the level of distortion. For this test, each EdgeHGN subnet can store/recall 9-bit binary sub-patterns. Therefore, the SI module is responsible for decomposing each character pattern into 9-bit sub-patterns. *As shown in Figure 3.11, the EdgeHGN can recognise distorted character patterns with up to 20% distortion*. This shows that EdgeHGN offers a reasonably high level of recall accuracy as a single-cycle learning algorithm.

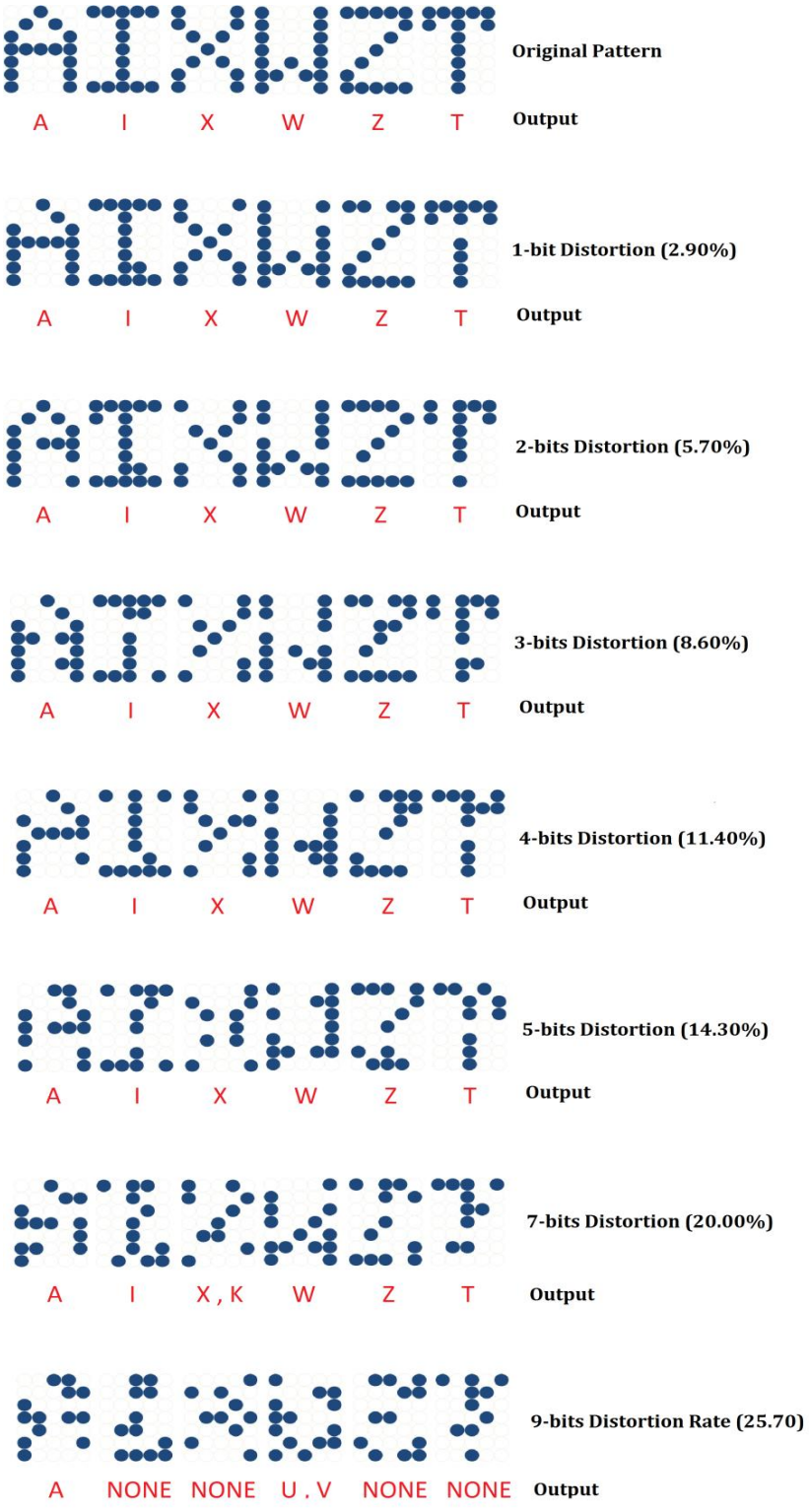

Figure 3.11: Seven different levels of random distortion applied to binary character patterns

Figure 3.12 depicts the recall accuracy of the EdgeHGN for various distortion rates in the input pattern. The figure shows that the EdgeHGN offers a reasonably high level of recall accuracy as a single-cycle learning scheme. In fact, *for binary character patterns with a distortion rate of up to 20%, an exact recall is achieved. Moreover, the scheme results in close recalls for distortion rates of 20% – 25%*. It should be noted that patterns with a random distortion of higher than 20% are difficult to identify, even with the human eye.

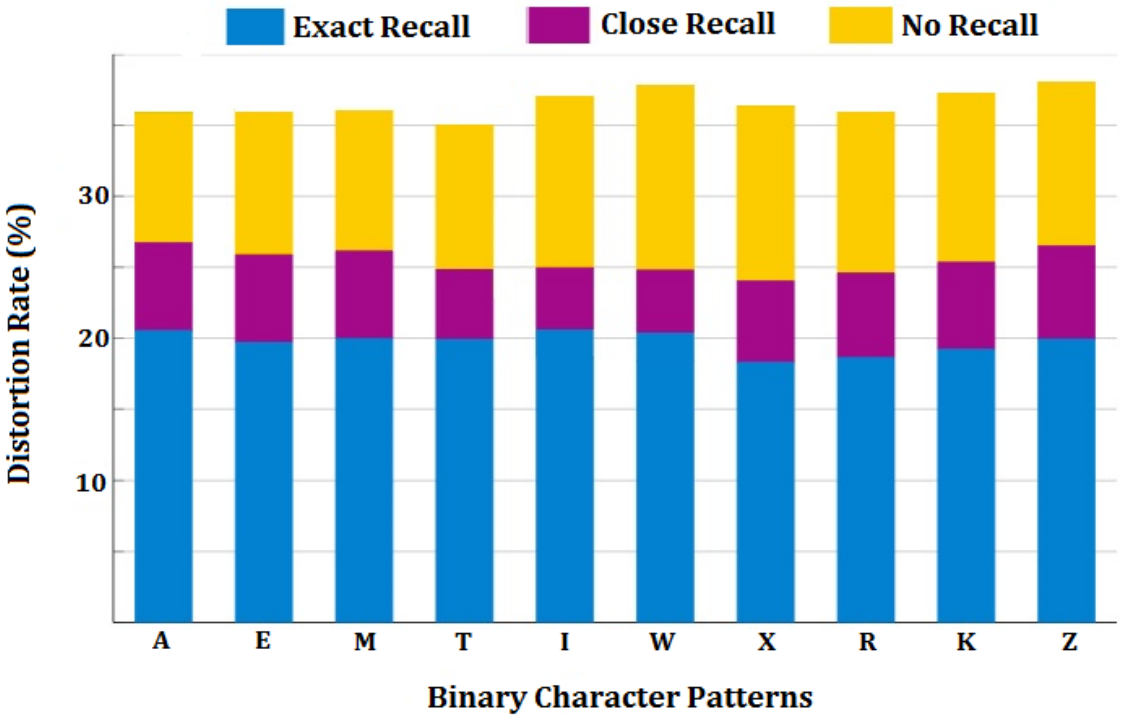

Figure 3.12: EdgeHGN recall accuracy for various distortion rates

Figure 3.13 shows the percentage of recalled nodes for the EdgeHGN scheme operating on patterns with low (<10%), medium (10% - 20%) and high (20% - 30%) levels of distortion. As shown in the figure, *more than 80% of EdgeHGN nodes are recalled when dealing with low-level distorted patterns, and almost 70% of nodes are recalled when the distortion rate is between 20% and 30%. Even with the presence of highly distorted input patterns, the EdgeHGN can recall almost 50% of nodes, which is a significant performance*.

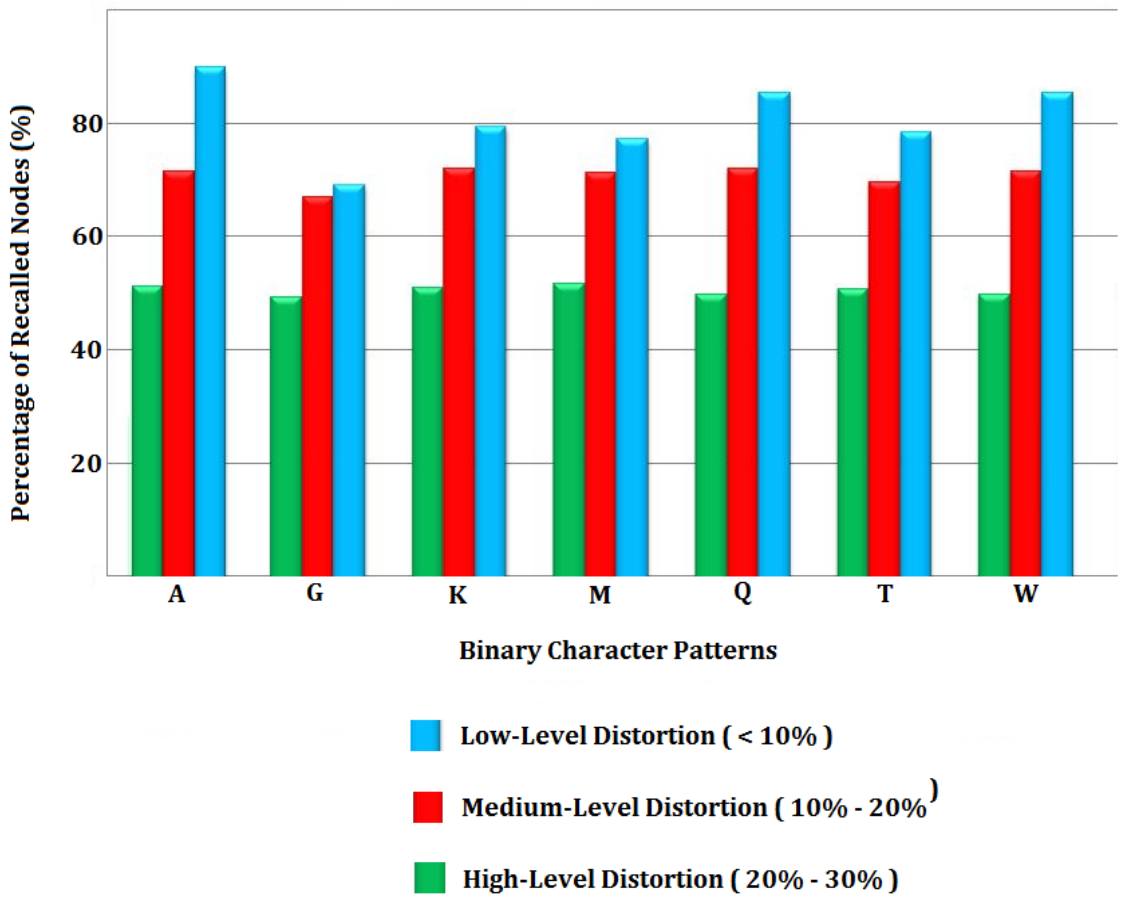

Figure 3.13: EdgeHGN node recall percentage for various distortion rates

As part of this testing setup, we have also compared the performance of the EdgeHGN with its predecessor scheme, the DHGN. The test patterns used in this recognition experiment are represented as 7-by-7 bit binary patterns. A large number of randomly generated binary patterns have been used as the test data. The only condition for these binary patterns is that each pattern should have at least one bit in each row to form the DHGN and EdgeHGN subnets. Each input pattern will be pre-processed by going through a hybrid drop-fall scheme for the EdgeHGN approach. The output with the lowest number of processing neurons will be used as an input to the EdgeHGN pattern matching algorithm. As a result, each character image used has been decomposed into a maximum of 7-bit sub-patterns (EdgeHGN subnets). The recall rate and response time have been used as a classification parameter in these test

sets. The recognition test has also been conducted on a set of binary character patterns with random distortion. The random distortion applied to the character patterns varies according to the level of distortion. As shown in Figure 3.14, the *EdgeHGN recognition scheme provides a higher recall percentage compared to the previous implementation of the DHGN*. This higher recall percentage is due to the lower number of processing neurons used in the scheme, along with the drop-fall module, which enables the results of the recognition process at the sub-pattern level to be produced with higher recall accuracy through forming smaller-sized subnets along with exploiting edge detection capability.

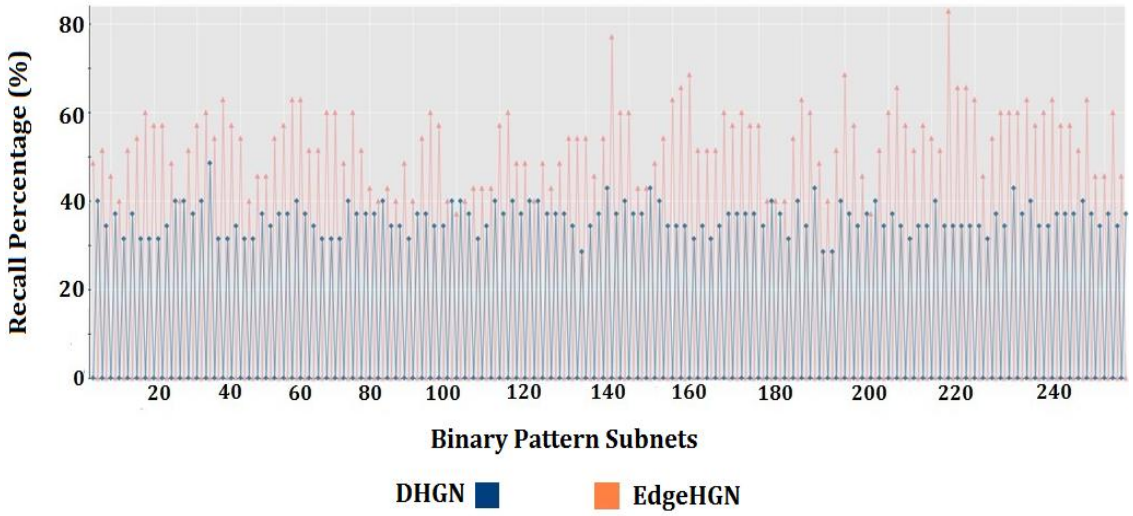

Figure 3.14: Recall percentage rate for EdgeHGN *v.* DHGN

In addition, a lower number of GNs within each EdgeHGN subnet results in a lower response time. In Figure 3.15, the lower response time is clearly depicted for EdgeHGN subnets in comparison with DHGN ones. A lower number of neurons due to the drop-falling pre-processing stage in the base layer of each EdgeHGN subnet forms a smaller hierarchy with fewer rows in the subnet. *This lower number of processing neurons will result in less communication overhead and can significantly minimise the response time for the overall recognition process while maintaining high recall accuracy*.

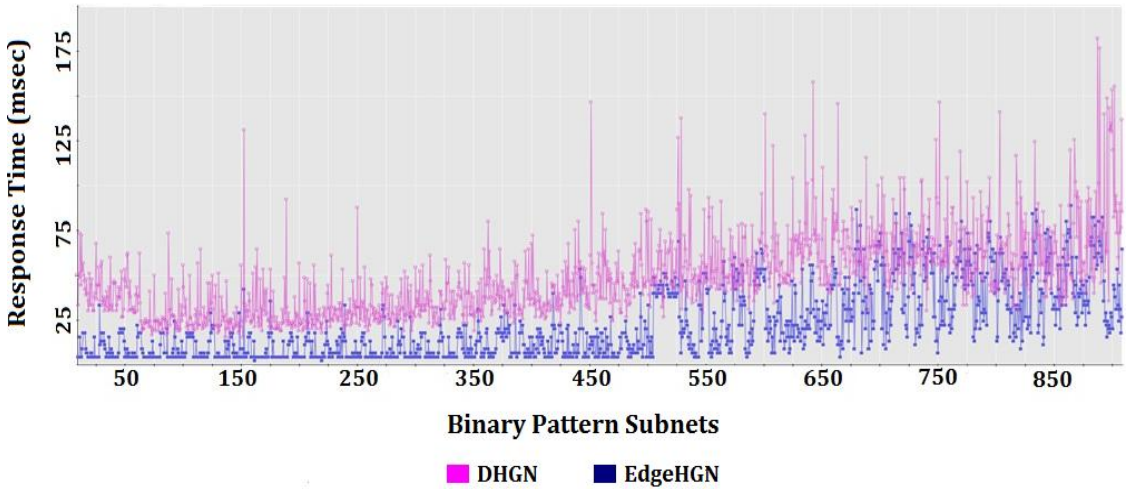

Figure 3.15: Response time for EdgeHGN *v.* DHGN

In Figure 3.16, the results of the response time per sub-pattern size are given using three different-sized datasets. The figure shows that the recognition process for each sub-pattern only requires a maximum of less than 100 milliseconds for 10,000 or more random patterns. Thus, *the EdgeHGN can perform fast recognition, while its response time is not significantly affected by an increase in the number of sub-patterns used.*

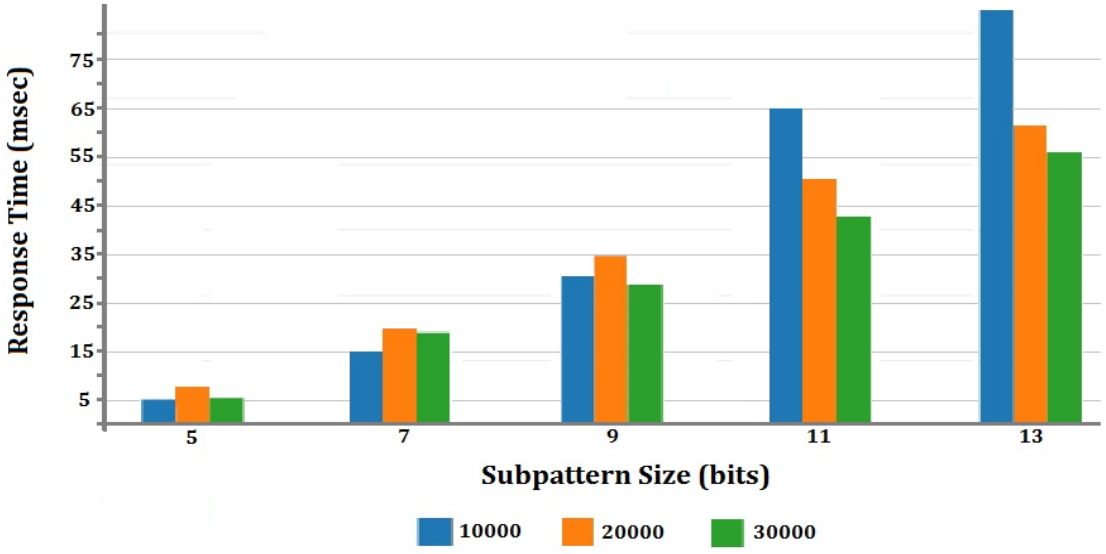

Figure 3.16: Recognition time for different sub-pattern sizes and different number of random sub-patterns

3.7.2 Recognition Test on Binary Images

This section plans to evaluate performance of applying EdgeHGN to binary image recognition case studies. *It should be strongly emphasized that we are not intending to do image processing, rather we aim to look at images as multi-dimensional objects which can be represented as discrete binary values for the purpose of pattern matching for classification*. The existing EdgeHGN implementation has been focusing on the recognition of the spatio-structural representation of an image via pixel-by-pixel analysis. This approach recognises the integrity of the contents of an image against any occurrence of random-bit distortion. However, it is insufficient for the recognition of images with multidimensional colour representation, including grayscale images. The changes in the colour of an image may influence the accuracy of the recognition system. Our proposed image recognition approach adopts the binary signature scheme for content-based image retrieval (CBIR) in the colour recognition process, while maintaining the binary analysis of the image for its spatio-structural recognition. For this purpose, each sub-signature (i.e., each signature that represents each colour) can be fed into a single EdgeHGN subnet. Cumulatively, this approach will lead to colour recognition within an image. In our image recognition exercise, we will implement a local binary signature approach where each image will be divided into grids (see Figure 3.17).

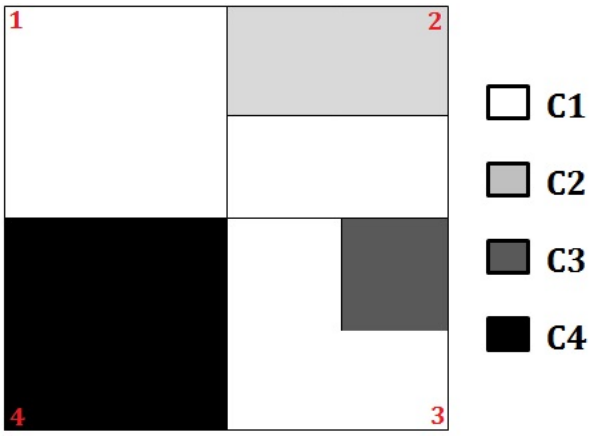

Figure 3.17: Block image with four different colours is divided into equally sized grids

Each grid will have its own signature representing each quadrant of the image, in which each bit value will correspond to the normalised percentage values of the colour within that image (see Table 3.5).

Table 3.5: Binary signatures for the image in Figure 3.17

Quadrant	Color	Density	Binary Signatures				
			b_1	b_2	b_3	b_4	b_5
1	C1	100%	0	0	0	0	1
	C2	0%	0	0	0	0	0
	C3	0%	0	0	0	0	0
	C4	0%	0	0	0	0	0
2	C1	50%	0	0	1	0	0
	C2	50%	0	0	1	0	0
	C3	0%	0	0	0	0	0
	C4	0%	0	0	0	0	0
3	C1	75%	0	0	0	1	0
	C2	0%	0	0	0	0	0
	C3	25%	0	1	0	0	0
	C4	0%	0	0	0	0	0
4	C1	0%	0	0	0	0	0
	C2	0%	0	0	0	0	0
	C3	0%	0	0	0	0	0
	C4	100%	0	0	0	0	1

With localised signatures, the colour distribution representation of an image will be further optimised to provide higher possible recall precision for a given set of images. It is worth noting that the quantisation level can have a significant effect on the recognition accuracy. Figure 3.18 shows the transformation of the global colour histogram for the image '*Lena*' from an original image to various quantisation levels. *Low quantisation levels produce similar sets of binary signatures for different images. Conversely, high quantisation levels can also have an adverse effect on recognition accuracy, as it tends to distribute colour frequency to a higher number of colour classes, thus reducing the possibilities of colours being grouped into similar classes*. By analysing recall rates and error value rates, we can pinpoint an optimal value for the quantisation level for our recognition purposes.

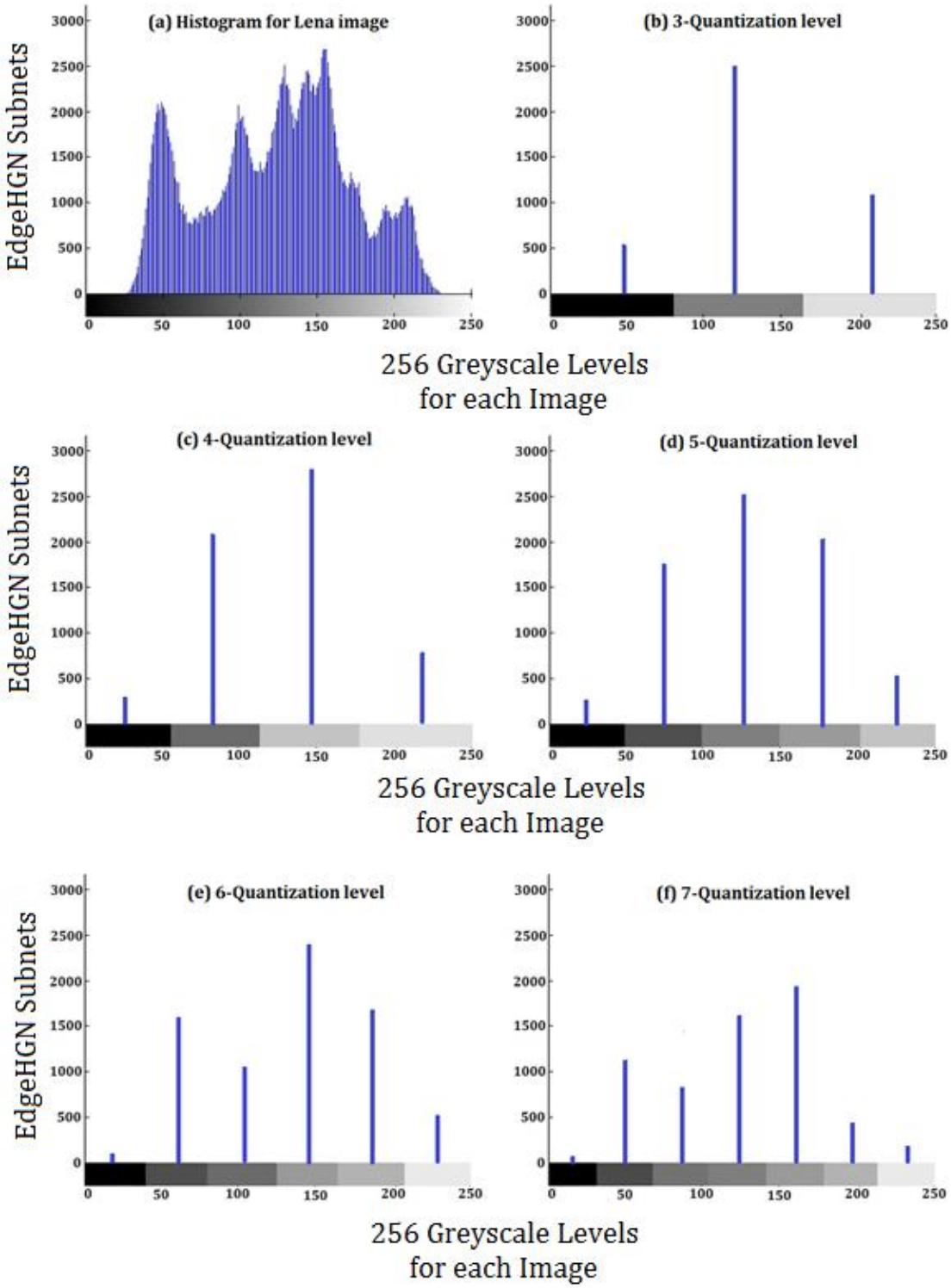

Figure 3.18: Transformation of global colour histogram of image Lena from original image to various quantisation levels

As shown in Figure 3.19, for the existing dataset, *the EdgeHGN recognition scheme performs best with the quantisation level 6.* The recognition test conducted also demonstrates that the EdgeHGN produces notably high recall rates as a result of its simple deterministic approach, while achieving this significant performance using single-cycle learning and through a one-shot recognition process.

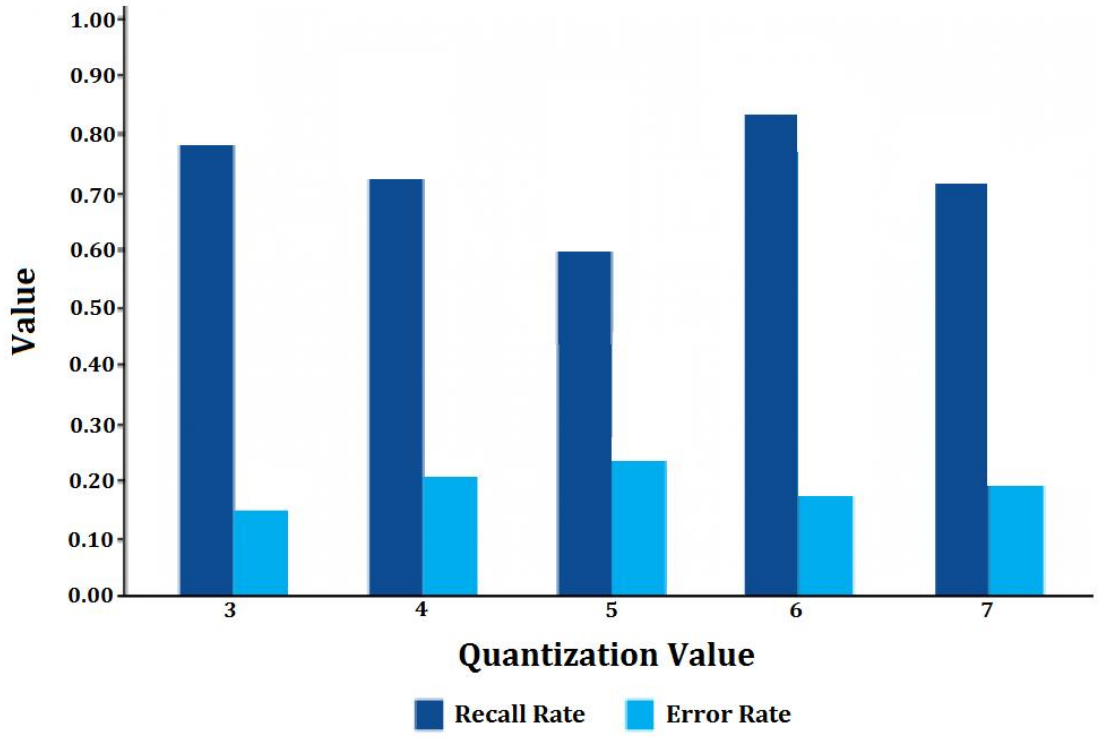

Figure 3.19: Average recall and error rates for EdgeHGN greyscale image recognition on 40 16KB binary images using various quantisation levels

A study has also been conducted into the EdgeHGN's performance with respect to its recognition time taken for each subnet, with different numbers of sub-patterns stored/recalled. These sub-patterns are derived from a similar set of 40 binary images used earlier for analysing optimal quantisation value. As shown in Figure 3.20, *an increase in the number of sub-patterns stored within the network does not have any adverse effect on the recall/store time of the EdgeHGN approach. In fact, the scalability of the EdgeHGN scheme will not be affected by the number of stored patterns within the EdgeHGN network.*

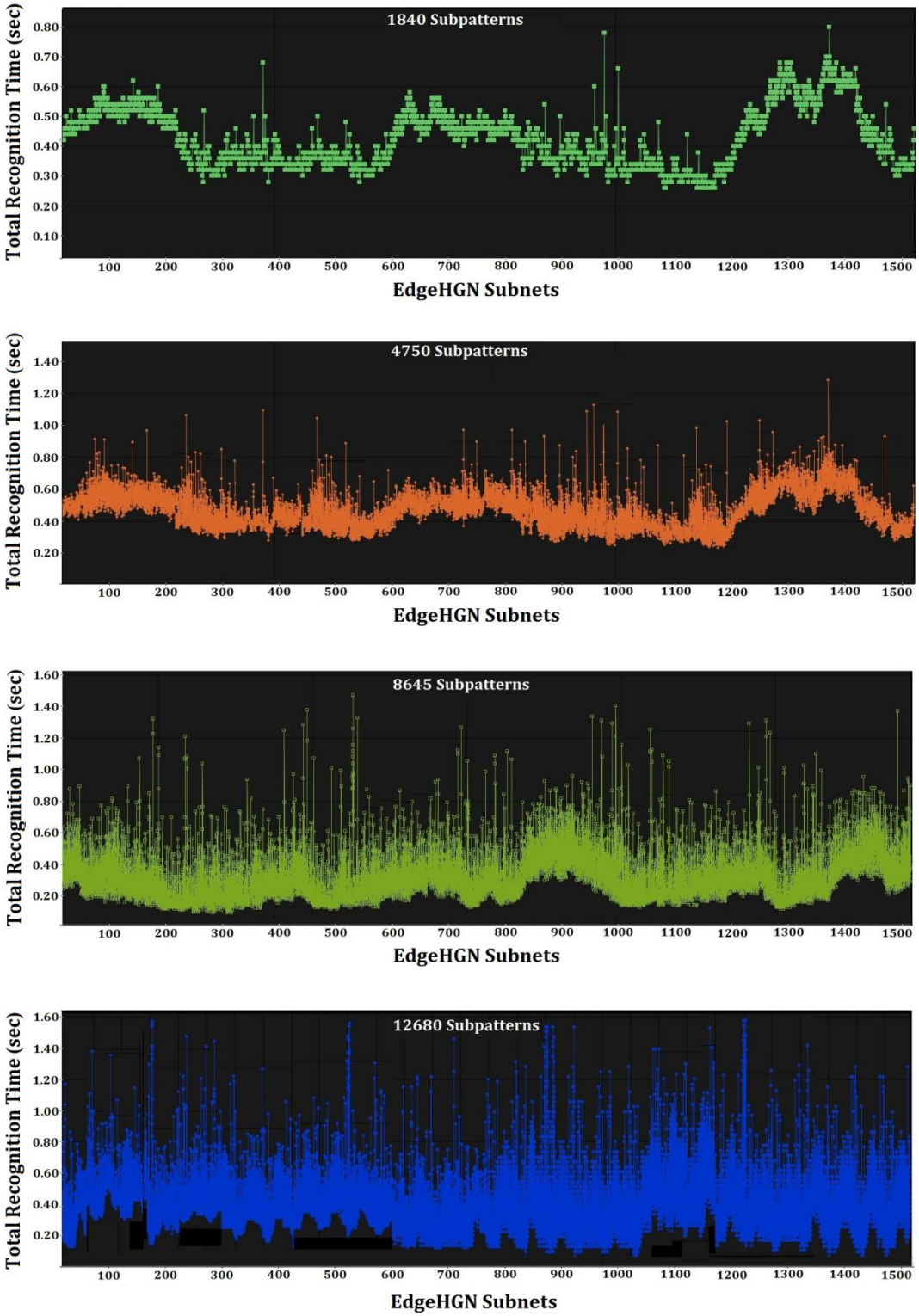

Figure 3.20: Total recognition time for each EdgeHGN subnet in binary pattern recognition with different number of sub-patterns derived from 16KB binary images

To further demonstrate the effectiveness of the EdgeHGN, we conducted a comparative study of its performance against Support Vector Machine (SVM) and an iterative Hebbian-based learning back propagation neural network (BPNN). For the purpose of this experiment, we classified 1000 grayscale facial binary images into 100 distinct classes corresponding to 100 individuals. Each class contains 10 grayscale binary images of the same person with 10 different facial expressions. To test various schemes, 100 randomly selected individual images were selected as the training set to test the remaining images against them for possible recall. Error rate was selected as the measurement metric to evaluate the performance of the related schemes. In this regard, recall error is defined as the number of wrongly classified images from the overall 950 test images when tested against the training dataset.

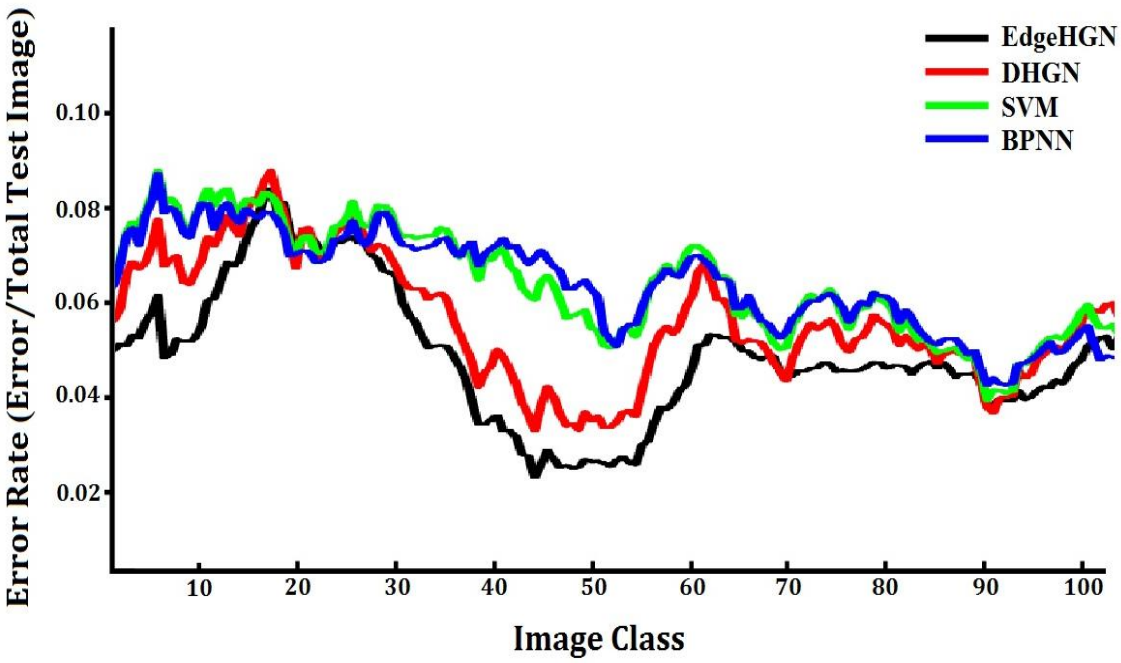

Figure 3.21: Recall error rates for binary image recognition of 100 facial image classes when tested against 1000 stored images using EdgeHGN, DHGN, SVM & BPNN schemes.

As depicted in Figure 3.21, *the EdgeHGN generally offers the lowest error rate compared with the DHGN, SVM and BPNN*. The experimental results show that the

SVM and BPNN perform relatively similar, and the SVM performs better for some image classes but worse for others. Nevertheless, *they both fall behind the EdgeHGN in terms of higher recall accuracy and lower error rate*. Higher error values for the BPNN may be due to the low number of training images fed into the network, resulting in fewer chances of achieving optimum outputs, which mostly reflect the original trained images. In the same context, the SVM performance deteriorates significantly when the size of the training set is much smaller than the number of the desired support vectors. While the EdgeHGN and DHGN both perform relatively well, EdgeHGN offers better recall accuracy due to its smaller-sized subnet networks, as well as the inclusion of edge information within the grayscale image analysis.

3.7.3 Recognition Test on Noisy Binary Images

In this section, we intend to demonstrate the capabilities of our proposed scheme to achieve multi-feature pattern recognition using collaborative-comparison single-cycle learning in EdgeHGN within a computational network. This part will demonstrate that our distributed pattern recognition scheme is able to include multiple image features as inputs within the recognition process. It is also capable of providing accurate classification within the bounds of single-cycle learning. The proposed multi-feature EdgeHGN is readily deployable within various network environments, ranging from coarse-grained computational networks such as computational cloud network to fine-grained networks such as the WSN. Our study here involves the recognition of noisy 128-by-128 bit binary images. Accuracy in image recognition is generally the benchmark that most contemporary schemes are measured against. The proposed scheme in this section takes a holistic approach towards incorporating both the colour and spatio-structural features into the image recognition process simultaneously. A binary signature scheme has been adopted for content-based image retrieval (CBIR) proposed by Nascimento and Chitkara (2002) within a pattern recognition procedure. The approach here integrates this global binary signature with Sobel's edge detection (Kimmel et al., 2005) for implementing EdgeHGN single-cycle image recognition.

3.7.3.1 Global Binary Signature Scheme for Colour Recognition

A common approach in representing colour distribution within an image is to use a global colour histogram (GCH). Given an n-colour model, a GCH is developed with an n-dimensional feature vector $\{\rho_1, \rho_2, ..., \rho_n\}$, where ρ_i represents the normalised percentage of colour pixels that corresponds to each colour element within an image. Nascimento and Chitkara (2002) proposed an alternative approach for colour distribution representation by using a global binary signature scheme, which is a compact form of the existing GCH that uses binary bit-strings as a signature. This signature is an abstract representation of the image's colour distribution. The bit-strings have a pre-determined size, which makes it ideal for use within EdgeHGN binary pattern representations.

3.7.3.2 Sobel's Edge Recognition for Structural Information

Edges provide important spatio-structural information for image recognition. The proposed approach towards image recognition here includes edge detection in the colour-based recognition process. Sobel's edge detection mechanism has been adopted, such that outputs from the edge detection process are represented as an edge map. Figure 3.22 shows the transformation of a colourful image into the corresponding edge map after applying the global binary signature to normalise the colour distribution. In our scheme implementation, we have used a common detection threshold value of 70 to generate the edge maps.

Figure 3.22. Edge map after applying Global Binary Signature and Sobel's edge detection

With the ability to capture and convert the two main features of an image – namely colours and edges – into binary patterns, the EdgeHGN approach could be applied to perform single-cycle binary pattern recognition.

3.7.3.3 Recognition Accuracy Analysis

There are two important factors that need to be investigated, namely *recognition accuracy* and *recognition speed*. For recognition accuracy, facial images were chosen that include similar background conditions and different structural representations as the test dataset. A set of 1000 facial images of 50 different individuals were used in this study. They were retrieved from Face Recognition Data, University of Essex, U.K. These images were of the size 180 x 200 pixels, as shown in Figure 3.23.

Figure 3.23. Fifty different individuals in the face image dataset obtained from the Face Recognition Data.

For colour recognition, all greyscale images were quantised into four grey levels. Ten different ranges of values were used for signature representation. It was determined that these values were able to represent distinctive colour features for all test images. Forty-bit signatures were created from this process, and each signature was decomposed into sub-signatures of 5-bits for building the EdgeHGN network. For edge detection, small-scale edge maps were developed for all facial images. Each edge map was a binary representation of the image with the size of 18 x 20 pixels, and it was developed using 3 x 3 Sobel's matrix kernel with both horizontal and

vertical scanning procedures. All edge maps were then input to a drop-fall algorithm to form 5-bit sub-patterns for the EdgeHGN recognition process. The recognition test involved classifying 1000 facial images corresponding to 50 distinct individuals into their respective classes. Each class consists of 20 images of the same individual with different facial expressions. To perform pattern matching, both base and test images were initially fed into the Sobel operator. In simple terms, the operator calculates the gradient of the image intensity at each point, giving the direction of the largest possible increase from light to dark and the rate of change in that direction. The result therefore shows how abruptly or smoothly the image changes at that point, and therefore how likely it is that that part of the image represents an edge, as well as how that edge is likely to be oriented (see Figure 3.24).

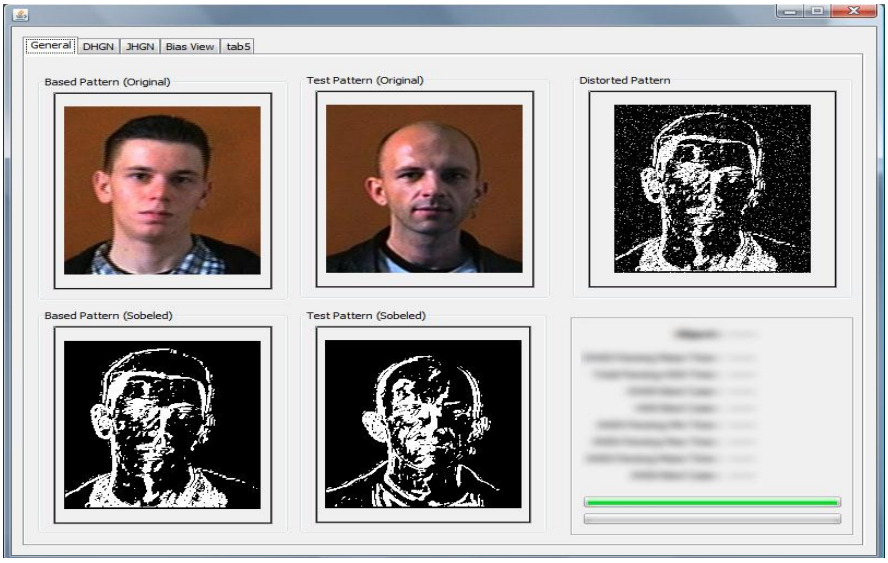

Figure 3.24. Applying the Sobel operator on both the base image and the test image before pattern matching

As discussed previously, there are four possible directions to apply the drop-fall scheme on the input pattern, and they generally produce four different paths to divide touching digits. They can start on the left or right side and can evolve downwards or upwards. One of the four is likely to produce the best result. Therefore, in our approach, a drop-fall scheme is chosen in a way that it ensures producing the least number of neurons, resulting in the least computational overhead (see Figure 3.25).

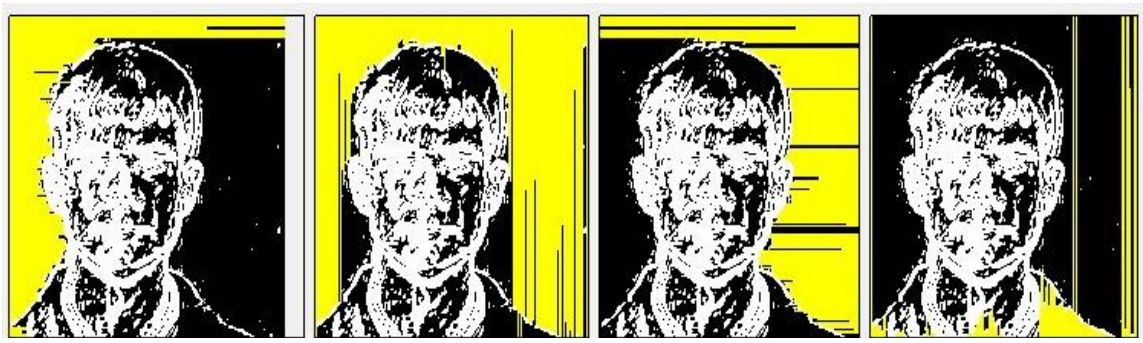

Figure 3.25. Applying four possible drop-fall directions to the input pattern

Each one of these drop-fall schemes will produce a different output, and the number of processing neurons will be different. As a result, EdgeHGN subnets will differ in size, which in turn results in varying response times for image processing and pattern matching. As shown in Figure 3.26, for 15 test images, the drop-fall scheme that is applied from the top direction will produce the lowest number of processing neurons, and in turn the lowest response times. Conversely, a drop-fall scheme that is applied from the bottom will generate a larger number of neurons in the network, resulting in the largest average processing time for pattern matching.

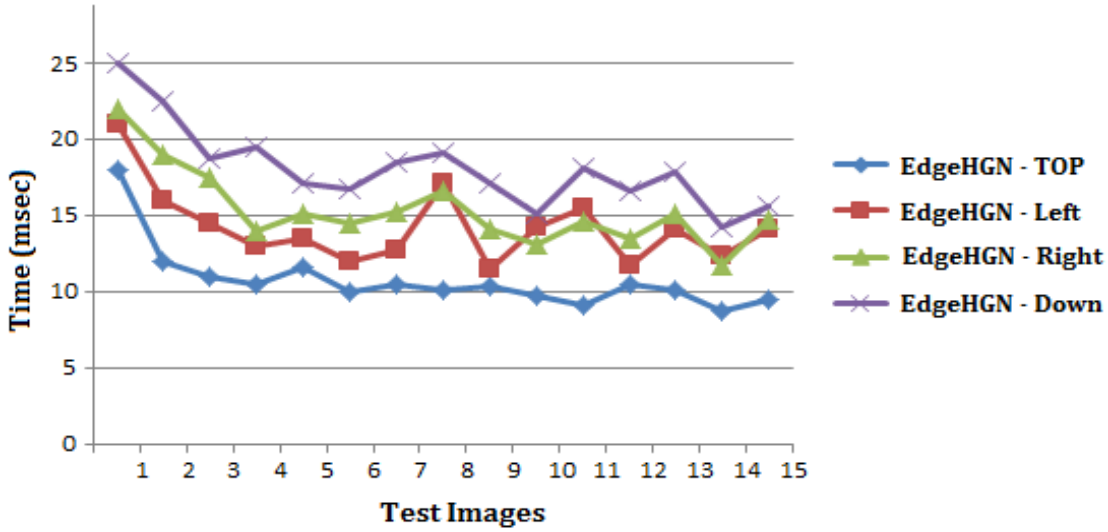

Figure 3.26. EdgeHGN recognition times after applying four drop-fall schemes on a test image

The results shown in Figure 3.27 demonstrate the error values for the EdgeHGN scheme processing 50 facial image classes of 1000 test images. *The EdgeHGN can achieve a very low average error rate of 0.02696% (~2.7%) in classifying 50 facial image classes.* To improve this error rate value, one option is to use larger quantisation values to more accurately represent greyscale values for test images.

Figure 3.27. Error values for EdgeHGN processing 50 facial image classes of 1000 test images.

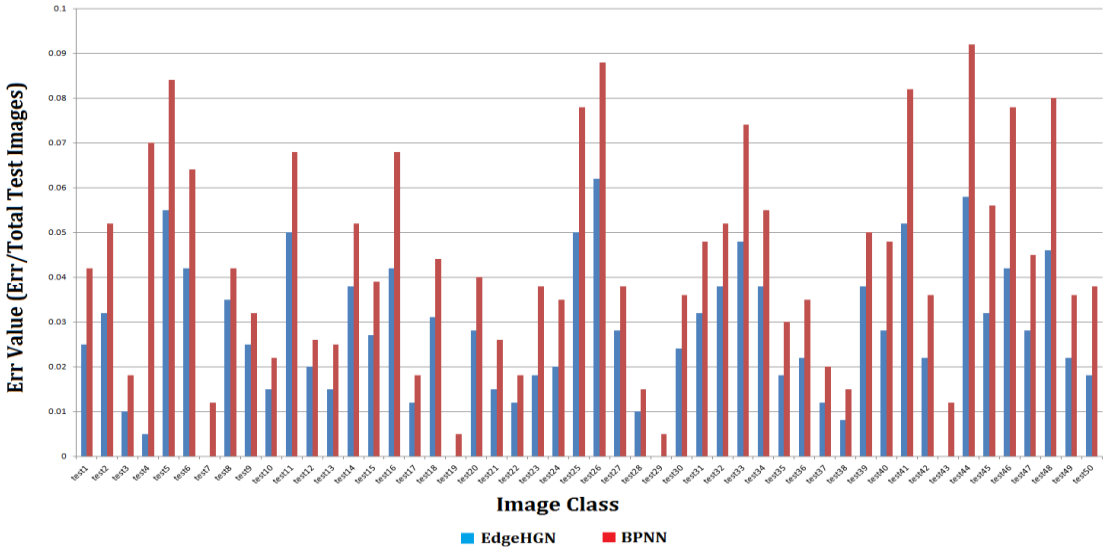

Figure 3.28. Error values for EdgeHGN and BPNN processing 50 facial image classes of 1000 test images.

In Figure 3.28, a comparative study on the error rate value of the EdgeHGN and iterative Hebbian-based learning back propagation neural network (BPNN) is given for processing 50 facial image classes of 1000 test images. *The EdgeHGN offers a better classification and accuracy rate when compared with the BPNN for all 50 image classes (error rate of 2.7% for the EdgeHGN compared with an error rate of 4.4% for the BPNN)*. The BPNN higher error rate might be due to the fact that the BPNN network was training with a low number of training instances, resulting in lower optimal classification accuracy for the scheme.

The next category of testing in our study is performed for the recognition of noisy 128-by-128 bit binary images. In this experiment, both Gaussian noise and impulse noise are added to the '*Lena, House, Cameraman and Boat*' images, and the recall accuracy of the EdgeHGN scheme is tested against recognising noisy images among a dataset of different heterogeneous images. By applying different levels of Gaussian distributed noise to the images, the original image pixel value distribution is changed. As a result, the pixel value distributions for noisy images are quite different compared with the original image. Moreover, adding Gaussian noise to the image will result in deterioration in the number of pixels corresponding to pure black and pure white values (0 and 255), making them significantly difficult to recognise as the original image. By further applying an impulse noise, the recognition task is made even more complicated, as the values of damaged pixels contain no information, while the positions of damaged pixels are also unknown. There are two important types of impulse noise: *salt-and-pepper noise* and *random-valued noise*. In general, the pixels damaged by salt-and-pepper noise are much easier to find, as the values are either $\boldsymbol{d_{min}}$ or $\boldsymbol{d_{max}}$. The detection of pixels corrupted by random-valued impulse noise is more difficult than salt-and-pepper impulse noise because the value of the damaged pixels can be any number between $\boldsymbol{d_{min}}$ and $\boldsymbol{d_{max}}$.

This recognition test is divided into two parts. The first part involves the recognition of noisy images contaminated by both Gaussian noise and salt-and-pepper noise (with $\boldsymbol{\sigma = 10}$ and $\boldsymbol{s = 30\%}$) among 20 heterogeneous binary images previously stored within the EdgeHGN network. The second part implies a similar

configuration, but with the test images contaminated by both Gaussian noise and random-valued noise (with $\boldsymbol{\sigma}$ **= 10** and $\boldsymbol{s}$ **= 25%**). Figure 3.29 and Figure 3.30 show the results of the recognition tests. *The EdgeHGN pattern recogniser is capable of providing high recall accuracy for distorted heterogeneous binary image recognition. In addition, the EdgeHGN can store the data of all 20 heterogeneous images within a single-cycle learning process without any reductions in its recall accuracy*.

Figure 3.29: (Top) images contaminated by both Gaussian and salt-and-pepper noise with $\boldsymbol{\sigma}$ **= 10** and $\boldsymbol{s}$ **= 30%** (bottom) recognition tests using the EdgeHGN scheme

Figure 3.30: (Top) images contaminated by both Gaussian and random-valued noise with $\boldsymbol{\sigma}$ **= 10** and $\boldsymbol{s}$ **= 25%** (bottom) recognition tests using the EdgeHGN scheme

3.7.4 Handwritten Object Recognition Test with Multiple Features

A series of classification tests are performed using the EdgeHGN to evaluate the performance of the scheme on handwritten character recognition. These experiments will demonstrate the strength of the EdgeHGN as a distributed classifier for complex pattern recognition tasks. The experimental results are compared with other methods discussed in the literature as part of the research work conducted by Duin and Tax (2000). The experimental dataset is taken from Frank and Asuncion (2010) and is publicly accessible from the ML repository. The dataset contains 10 classes of numerical characters ranging from '0' to '9', where each class holds 200 objects and each object is converted to a 30 x 48 binary image. Classification tests are designed so they work on the following four feature sets obtained from a similar set of objects (Frank & Asuncion, 2010):

1. Fourier: 76 Fourier coefficients of the character shapes
2. Pixel: 240 pixel averages in 2 x 3 windows
3. Zernike: 47 Zernike moments
4. Morph: 6 morphological features.

3.7.4.1 Classification Procedures

To perform recognition, the EdgeHGN implements a three-stage process of *pre-processing*, *classification* and *result calculation*.

Feature Pre-Processing

As part of the pre-processing phase, a discretisation process using a binning approach is applied to all selected data features to ensure that continuous feature values are converted into discrete format representations so they can be used and processed by the EdgeHGN algorithm. With the exception of the pixel average feature set, where all provided data are of a discrete nature, for all other feature sets, the process of discretisation is implemented by defining five bins (thresholds) with a different range of values. These values are determined based on the maximum, minimum and mean values calculated from the overall feature set. Table 3.6 illustrates how these bins are defined and formed.

Table 3.6: Discretisation of feature data values using variable-binning methods

Feature	min	max	μ	Bins				
				1	2	3	4	5
Zernike	0.0011	777.86	88.64	≤ 25	26-50	51-90	91-400	401-800
Fourier	0.0002	0.7965	0.1320	≤ 0.001	0.002-0.05	0.06-0.14	0.15-0.50	0.51-0.80
Morph	1.1431	17572.2	2104.4	≤ 50	51-500	501-2500	2501-10000	10001-18000

The process of converting the feature dataset from continuous data space to a discrete representation will reduce the inherent complexity of the dataset for classification at the cost of producing less accurate results because some data values are lost in the conversion phase. Nevertheless, the discretisation process yields a set of patterns corresponding to each feature, where the size of the patterns denotes the number of values for each feature, while the dimension of the patterns represents the number of bins (thresholds) used (i.e., five).

Feature Recognition

To perform classification tasks on the four feature datasets, four separate EdgeHGN networks with different sizes are constructed to implement recognition on a specific feature set resulting from the pre-processing phase. Table 3.7 illustrates the EdgeHGN topology structure used for processing these four feature sets.

Table 3.7: EdgeHGN networks setup details for processing four feature sets

Parameters	Values	
Number of EdgeHGN networks	4	
Sub-pattern size	9	
Number of GNs per subnet	25	
Number of subnets	**Zernike**	5
	Fourier	9
	Morph	1
	Pixel Avg.	27

To perform recognition, the entire feature set is first presented to the SI module node on each EdgeHGN network. The SI module node then divides and distributes the input data to all available subnets for the recognition process to be conducted at the sub-pattern level. The results from all subnets are then relayed back to the SI module before they are input to a maximum voting phase where the best fit/match for the respective pattern class is determined.

Result Calculation

The result calculation stage involves identifying the optimal feature as the best representative for each pattern class. It should be noted that this last phase of processing can be conducted within a coordinator node. For the purpose of this exercise, we have calculated error value, precision, recall and accuracy parameters as a comparative basis for the classification process. Table 3.8 shows how each of these parameters are defined and represented.

Table 3.8: Recognition parameters with their respective definitions

Recognition Parameters	Definitions
Precision	$\frac{\text{True Positive}}{\text{True Positive+False Positive}}$
Recall	$\frac{\text{True Positive}}{\text{True Positive+False Negative}}$
Accuracy	$\frac{\text{True Positive+True Negative}}{\text{True Positive+True Negative+False Positive+False Negative}}$
Error Value	$\frac{\text{False Positive+False Negative}}{\text{Total Number of Test Objects}}$

3.7.4.2 Recognition Analysis

Figure 3.31 illustrates the EdgeHGN outputs for multi-feature classification decisions, and Figure 3.32 shows average feature values and the best results achieved for each object class. *The classification results show that for the majority of object classes in the test, the morphological and pixel average feature values generate the best classification output.*

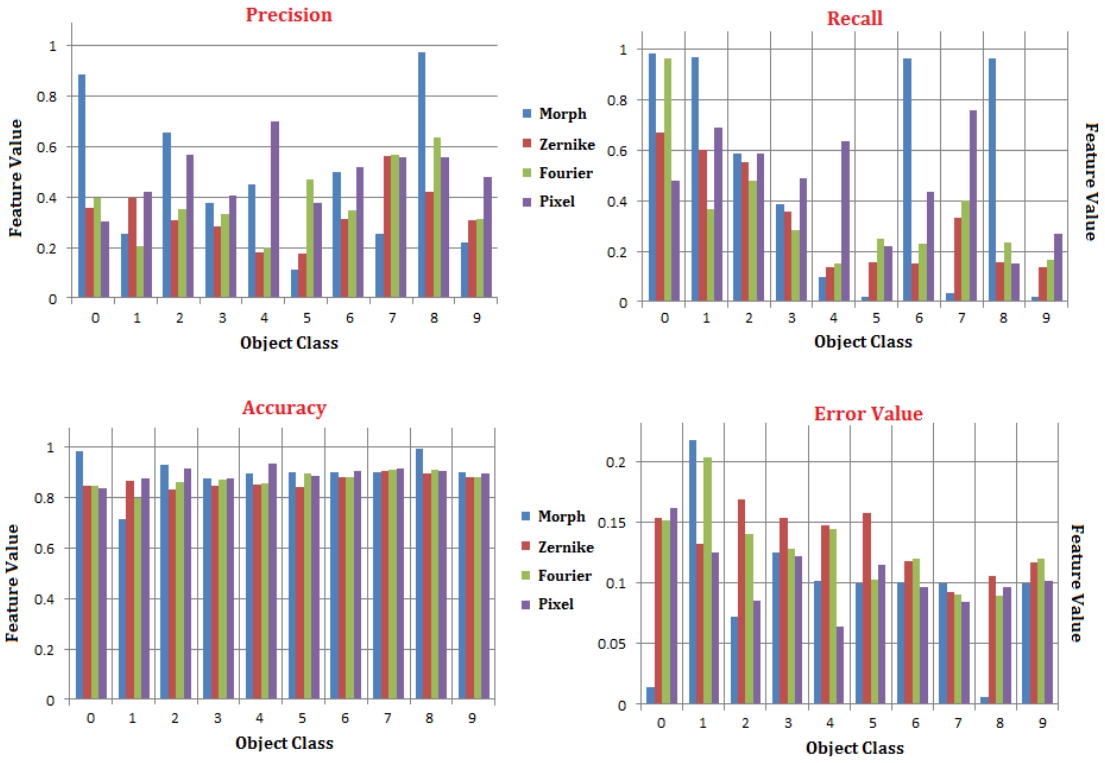

Figure 3.31: EdgeHGN classification results on four different features of numeral character objects.

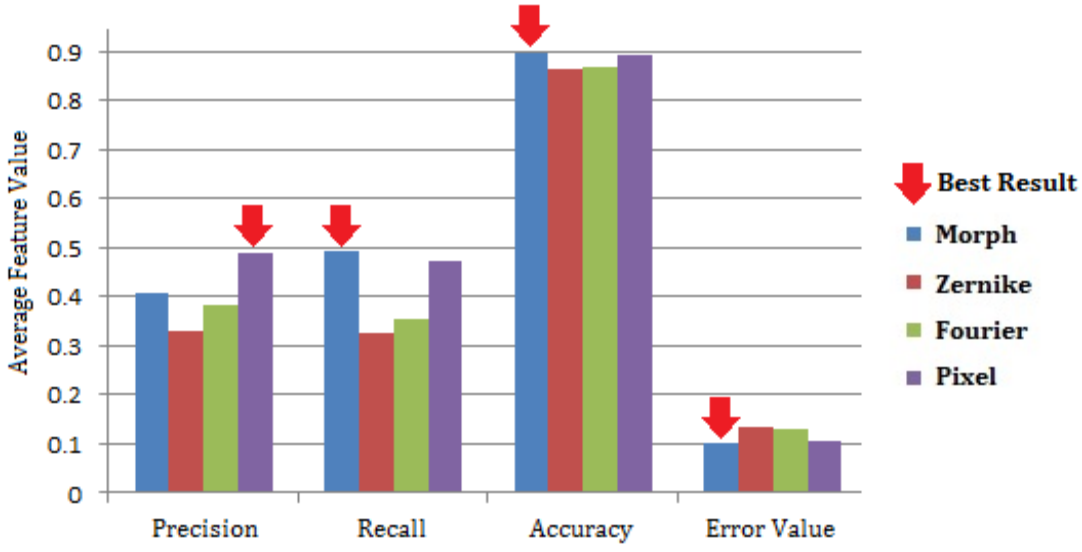

Figure 3.32: EdgeHGN classification best average results on four different features of numeral character objects

Figure 3.32 shows that morphological features generally exhibit low error values while producing reasonably high recall and accuracy rates. By performing a comparative study between the EdgeHGN and other classifiers discussed by Duin and Tax (2000), we can conclude that *the EdgeHGN is capable of producing comparable accuracy rates for the same feature sets*. Figure 3.33 conducts comparative analysis to determine error value rates for different classifiers.

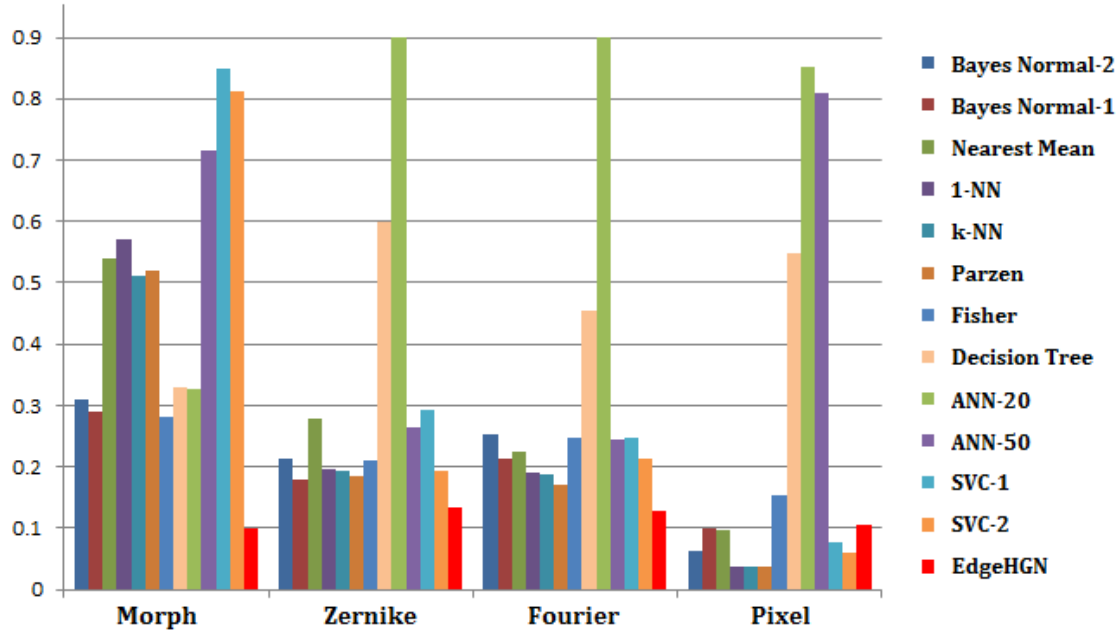

Figure 3.33: Comparative study on error rates between EdgeHGN and other classifiers for similar dataset with respective features.

It can be concluded from the results that the EdgeHGN performs remarkably well, with the lowest number of error rates for all tested features, with the exception of pixel average. However, even for pixel average, the EdgeHGN offers promising results when compared with other classifiers. Moreover, the results from the figure 3.33 indicate that artificial neural network (ANN) produce the least accurate results. While other statistical schemes seem to generate acceptable error rates, their cumbersome parameter estimation processes make them less favourable for feature classification tasks when compared with the EdgeHGN.

3.8 Conclusion

An important aspect in pattern recognition schemes is in their algorithmic design. A proper design will lead to higher efficiency and will be able to provide better classification accuracies. Our GN based algorithms have been developed with the scalability consideration being paramount. GN has the ability to perform pattern recognition processes on distributed systems due to its simple recognition procedure and lightweight algorithm. Furthermore, GN incurs low computational and communication costs when deployed in a distributed environment. In this regard, this chapter made an attempt to introduce and discuss a newly proposed approach for cloud distributed pattern recognition, known as Edge Detecting Hierarchical Graph Neuron (EdgeHGN). EdgeHGN reduces redundant data content for recognition through segmentation, by applying a hybrid drop-fall algorithm on the input pattern. EdgeHGN allows the recognition process to be conducted in a smaller sub-pattern domain, hence minimizing the number of processing nodes, which in turn reduces the complexity of pattern analysis. In addition, the recognition process performed using the EdgeHGN algorithm is unique in a way that each subnet is only responsible for memorising a portion of the pattern (rather than the entire pattern). A collection of these subnets is able to form a distributed memory structure for the entire pattern. This feature enables recognition to be performed in parallel and independently. The decoupled nature of the sub-domains is the key feature that brings dynamic scalability to our data processing approach for the cloud. Moreover, EdgeHGN provides a capability for a recognition process to be deployed as a composition of sub-processes executed in parallel across a distributed network. Sub-processes execute mutually independently. This approach is less cohesive compared to any other pattern recognition scheme.

Our experimental results demonstrate the fact that EdgeHGN is able to achieve very low error rate of (~2.7%) in classifying 50 facial image classes. This high accuracy rate is accompanied by remarkable scalability features. Our tests show that an increase in the number of sub-patterns stored within the network does not have any adverse effect on the recall/store time of EdgeHGN approach. In fact, the scalability

of EdgeHGN scheme will not affected by the number of stored patterns within the EdgeHGN network.

In contrast to the rest of hierarchical models already proposed in the literature, EdgeHGN's pattern matching capability and the small response time, that remains insensitive to the increases in the number of stored patterns, can make this approach remarkably suitable for clouds. Moreover, the EdgeHGN does not require definition of rules or manual interventions by the operator for setting of thresholds to achieve the desired results, nor does it require heuristics entailing iterative operations for memorization and recall of patterns. In addition, our approach allows induction of new patterns in a fixed number of steps. Whilst doing so it exhibits a high level of scalability i.e. the performance and accuracy do not degrade as the number of stored pattern increases over time. Its pattern recognition capability remains comparable with contemporary approaches. Furthermore, all computations are completed within the pre-defined number of steps and as such the approach implements one-shot, i.e. single-cycle or single-pass, learning.

This Page Intentionally Left Blank

Chapter 4

EdgeHGN_MR: Edge Detecting Hierarchical Graph Neuron based MapReduce

One of the main challenges for large-scale computer clouds dealing with massive real-time data is in coping with the rate at which unprocessed data are being accumulated. There are many *big data* demands in scientific and engineering applications – including biotechnology (e.g., characterisation using synchrotrons) and the global monitoring of fixed and mobile assets in industry, transport and defence – that entail massive real-time streams from and to stationary or mobile sensors and actuators. As a result of their dynamic and distributed nature, as well as their exponential growth, real-time data management is complicated, and storage, updates and analytics are costly (Szalay, et. al., 2006). This thesis hypothesises that fundamental changes and improvements in data access and movements are possible and beneficial for cloud-based processing. To provide improvements particularly regarding computational complexity and scalability, this chapter proposes a novel

associative-memory-based scheme for big data processing that is scalable, distributable and lightweight, and that overcomes some of the issues encountered in traditional data access mechanisms for data storage and retrieval. Thus, the primary aim of this chapter is to apply an access scheme that will enable fast data retrieval across multiple records and data segments associatively. In this regard, associative memory concepts open a new pathway for accessing data in a highly distributed environment that will facilitate a parallel-distributed computational model to automatically adapt to the dynamic data environment for optimised performance.

4.1 Neural Network based Classification Techniques

As discussed in detail in Chapter 2, high computational complexity and large memory requirements are common drawbacks in neural-network-based classification techniques such as the back propagation network and the self-organising maps. The computational complexity and memory requirements increase substantially with the increase in problem size. These algorithms often fail to scale up when presented with large and complex datasets, such as the datasets encountered in big data analysis. In fact, many of the ML schemes discussed in the literature do not offer acceptable levels of scalability and adaptability, making them infeasible for solving large-scale pattern recognition problems.

Hence, what is really required for any cloud system is a complete data access scheme that enables data partitioning on-the-fly and that has the ability to disseminate processing nodes for specific data retrieval/storage tasks and consolidate the data access scheme using an efficient partitioning approach. This integration within a complete end-to-end scheme will enable data storage and retrieval processes to be performed effectively, regardless of the distribution of data within the cloud system. In this regard, associative memory concepts open a new pathway for accessing data in a highly distributed environment that will facilitate a parallel-distributed computational model to automatically adapt to the dynamic data environment for optimised performance. *The problem is to marry such concepts with relevant advanced parallel processing patterns*. With this in mind, the proposed scheme in

this chapter targets a new type of data-processing approach that will efficiently partition and distribute data for clouds and facilitate content-based access for a wide range of applications. This chapter introduces an associative memory based MapReduce, referring to as EdgeHGN-based MapReduce or EdgeHGN_MR. In this chapter, three extensions of EdgeHGN_MR are presented, dealing with different data-intensive scenarios (EdgeHGN_MRv1, EdgeHGN_MRv2 and EdgeHGN_MRv3). EgdeHGN_MRv1 is designed to handle classification tasks efficiently when dealing with large datasets. In such cases, the input data are split among data chunks so they can be later processed by Mapper functions in parallel where each Mapper constructs the same EdgeHGN_MR classifier using a similar set of training data. EdgeHGN_MRv2 deals with scenarios where the training data are voluminous. To perform effective classification tasks, the training data are split into data chunks so they can be processed by Mapper functions in parallel. In such cases, each Mapper still creates the same EdgeHGN_MR classifier, but this time using only a subset of the training dataset to train the EdgeHGN network. To ensure we can still achieve acceptable levels of classification accuracy, EdgeHGN_MRv2 utilises a balanced bootstrapping approach (Alham, et. al., 2013) along with a majority voting scheme, as part of its processing framework. Lastly, EdgeHGN_MRv3 is designed to work with cases with an excessive number of processing neurons in the EdgeHGN network. To achieve high classification accuracy, EdgeHGN_MRv3 parallelises and distributes EdgeHGN processes across Mapper functions so each Mapper only utilises a subset of the processing neurons for training.

4.2 Associative Memory Concept for Implementing Large Scale Classification Operations

Unlike the existing relational, hierarchical and object-oriented schemes, associative models can analyse data in similar ways to which our brain links information. When implemented in voluminous data clouds, such interactions can assist in searching for overarching relations in complex and highly distributed datasets with speed and

accuracy. The proposal in this chapter improves MapReduce-based cloud applications in a number of different ways by replacing referential data access mechanisms with more versatile and distributable associative functions, which allow complex data relations to be encoded into the keys as patterns. These patterns can be applied in a variety of applications requiring content recognition (e.g., image databases, searches within large multimedia files and data mining). *The algorithmic strengths of the MapReduce approach are investigated in relation to the effectiveness of one-shot learning-based parallelism provisioned via our distributed pattern recognition approach, EdgeHGN*. The principle of AM-based learning will be implemented through the use of hierarchically connected layers, with local feature learning at the lowest layer and upper layers combining features into higher representations.

As we know, existing data access mechanisms for cloud computing such as MapReduce have proven the viability of parallel access approaches in cloud infrastructure. However, the MapReduce model does not explicitly provide support for processing multiple related heterogeneous datasets. While processing data in relational models is a common requirement, this restriction limits its functionality when dealing with complex and unstructured data such as images. Relational databases use a separate, uniquely structured table for each different type of data for specific applications, and programmers must know the precise structure of every table and the meaning of every column a priori. *To overcome this, our proposed scheme preserves the strength of the MapReduce model and eliminates/alleviates most of these constraints in a well-integrated manner where there is no outward change to the way in which MapReduce models are deployed and used*. In this context, our research will investigate the inclusion of an associative approach in the MapReduce model to support application-specific pattern recognition and data-mining operation. Hierarchical structures in AM models are of interest because they have been shown to improve scalability while preserving accuracy in pattern recognition applications (Ohkuma, 1993). Our proposal is based on an EdgeHGN associative memory model that has been specially designed for distributed processing and readily implemented within distributed architectures.

4.2.1 EdgeHGN Approach for Cloud Data Access

Through the redesign of the data management architecture, data records are treated as patterns. For this purpose, a data access scheme that enables retrieval to be conducted across multiple records and data segments in a single-cycle and parallel approach is considered. The access mechanism is implemented according to the nature of the database. The retrieval process will be conducted on a set of records that reside in a particular node. No alterations will be made to the condition of the record itself. A parallel retrieval approach is used, in which records in each storage node are analysed locally without incurring any communication costs. A distributed pattern matching/recognition approach, such as the EdgeHGN, can be used to retrieve data from the cloud. The EdgeHGN cloud access scheme relies on communications between adjacent nodes. The decentralised content location schemes are implemented to discover the adjacent nodes in a minimal number of hops. A GN-based algorithm for optimally distributing the EdgeHGN subnets (clusters or sub-domains) across the cloud nodes is provided to automate the bootstrapping of the distributed application and to investigate dynamic load balancing over the network. Note that the EdgeHGN subnets perform data mapping on each of the data nodes within the HDFS infrastructure. Within each EdgeHGN subnet, the records are stored in an associative pattern; each EdgeHGN neuron corresponds to a single data field. The mapping process occurs within the body of the EdgeHGN subnet.

4.3 EdgeHGN based MapReduce

MapReduce has issues and limitations. One might think that the absence of a rigid schema makes MapReduce the preferable option over DBMSs. However,

- Existing MapReduce implementations provide built-in functionality to handle simple key-value pair formats, but the programmer must explicitly write support for more complex data structures.
- In most MapReduce implementations, the map function conducts its operation assuming that all related data are distributed vertically (i.e., records are being

uniformly distributed across the network). However, it is possible that some parts of the related records are being stored at different physical locations – for instance, a large database table being split into multiple sub-tables and stored among the cloud nodes.

- In addition to the vertical distribution issue, another issue related to the MapReduce function is that the operations produce numerous intermediary entities between the map and reduce functions. These entities could be in the form of intermediate files. The contents of these files would need to be sorted before they are input into the reduce function. This system-wide sort and redistribution incurs additional processing and communication costs. Thus, data fragmentation can affect MapReduce schemes focused on vertical splitting where data are partitioned based on the file structure. In addition, in MapReduce, the underlying assumption is that the solution can be expressed in terms of the map and reduce functions working on key-value pairs, while in some cases this may not be natural, such as multi-stage processes, and this can lead to inefficiencies.

As a result, and to address the aforementioned concerns with regards to the MapReduce functionality, this section attempts to *take the MapReduce key-value scheme into a higher level of functionality by simply replacing the scalar key-value pair functionally with our AM-based scheme, EdgeHGN. By having an associative key-value model, we can deal with data simply by using a pattern matching model that treats data records as patterns and provides a distributed data access scheme that enables data storage and retrieval by association*. Our GN-based algorithms have been developed with the scalability consideration being paramount. The GN has the ability to perform pattern recognition processes on distributed systems due to its simple recognition procedure and lightweight algorithm. Further, it incurs low computational and communication costs when deployed in a distributed environment. In this regard, the EdgeHGN has been developed for the cloud environment, which allows the recognition process to be conducted in a smaller sub-pattern domain, hence minimising the number of processing nodes, which in turn reduces the complexity of pattern analysis. In addition, the recognition process performed using

the EdgeHGN algorithms are unique in that each subnet is only responsible for memorising a portion of the pattern (rather than the entire pattern).

A collection of these subnets can form a distributed memory structure for the entire pattern. This feature enables recognition to be performed in parallel and independently. *The decoupled nature of the sub-domains is the key feature that brings scalability to our data management approach for the cloud*. Moreover, the EdgeHGN provides capability for a recognition process to be deployed as a composition of sub-processes executed in parallel across a distributed network. Sub-processes execute mutually independently. This approach is less cohesive compared to any other pattern recognition scheme.

To achieve the aforementioned objectives, an initial step would be to develop a distributed data access scheme that enables record storage and retrieval by association where data records are treated as patterns. As a result, data storage and retrieval can be performed using a distributed pattern recognition approach implemented through the integration of loosely coupled computational networks, followed by a divide-and-distribute approach that facilitates the distribution of these networks within the cloud dynamically. Thus, *reconciling MapReduce with associated memory concepts, particularly for adaptive and fast data access, aggregation and movement is a key contribution of this chapter*.

In the following sections, three EdgeHGN based MapReduce approaches are presented to deal with different real-world data intensive scenarios. In particular, we have considered scenarios where we are dealing with large datasets, scenarios where the training data are voluminous and cases where we deal with an excessive number of processing neurons in the EdgeHGN network.

4.3.1 EdgeHGN_MRv1

EgdeHGN_MRv1 is designed to handle classification tasks efficiently when dealing with large datasets. In such cases, the input data are split among data chunks so they can later be processed by Mapper functions in parallel, where each Mapper constructs the same EdgeHGN_MR classifier using a similar set of training data.

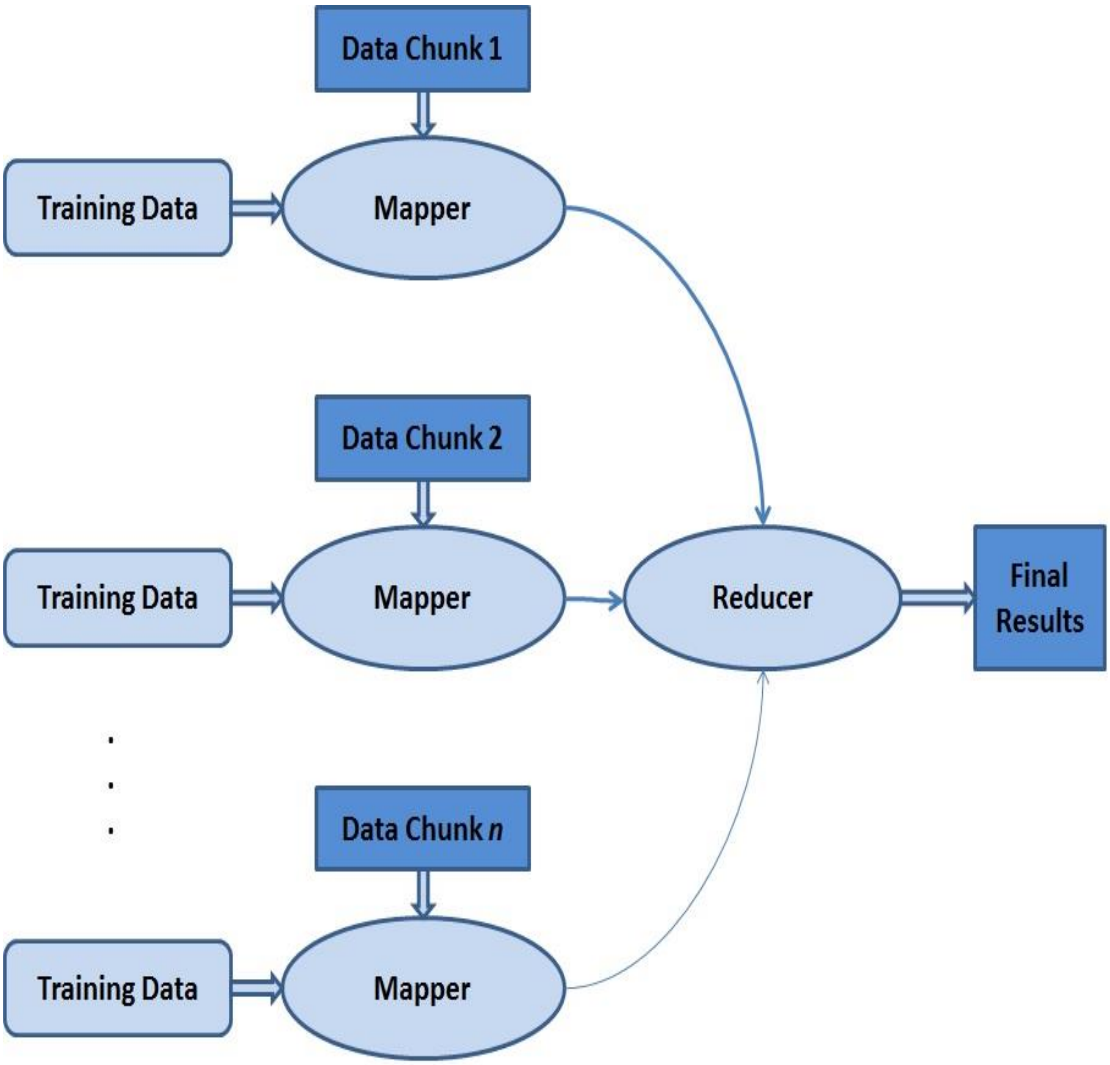

Figure 4.1: EdgeHGN_MRv1 architecture

Figure 4.1 shows the architecture of EdgeHGN_MRv1. Consider testing a scenario with a large set of testing data to be processed $\mathbb{T} = \{t_1, t_2, t_3 \ldots t_{in}\}$, $t_i \in \mathbb{T}$, where $1 \leq i \leq in$

i. t_i denotes a testing instance,
ii. $\mathbb{T}$ represents a testing dataset,

iii. in shows the number of inputs of an EdgeHGN network, it also represents the length of T

iv. inputs are represented by a format of ($\mathbf{instance}_i$, $\mathbf{target}_i$, **type**),

v. $\mathbf{instance}_i$ denotes t_i that is the input of an EdgeHGN network,

vi. $\mathbf{target}_i$ shows the desirable output in the case of $\mathbf{instance}_i$ being a training instance,

vii. **type** field consists of two values, ***train*** and ***test*** – which show the type of $\mathbf{instance}_i$ (if the ***test*** value is used, the $\mathbf{target}_i$ field value is set to null).

Testing data files are stored in the HDFS. Each data file contains a subset of testing instances along with all of the training data. As a result, the number of files $\mathbf{n}$ determines the number of Mapper functions used. File data contents are then fed into *EdgeHGN_MRv1* for classification. When the scheme starts processing, each Mapper initialises an EdgeHGN network, which results in the creation of $\mathbf{n}$ EdgeHGN networks in the cluster. It should be noted that all EdgeHGN networks perform an execution on the same set of parameters, and each Mapper function reads the data from an input file in the form of ($\mathbf{instance}_K$, $\mathbf{target}_K$, **type**). Algorithm 4.1 depicts the pseudo-code for *EdgeHGN_MRv1*. When the value of type field is set to ***train*** then the $\mathbf{instance}_i$ is input into the input layer of the EdgeHGN network. The network starts processing the input and calculating the result of each EdgeHGN subnet until the completion of the EdgeHGN process. Upon completion of the execution phase, the result will be recorded in the HDFS. The network then starts processing the next instance. This process is continued until all input data with ***train*** type values are processed by the EdgeHGN network. If the value of the type field is set to ***test***, the network starts performing a recognition test, where each Mapper only classifies a subset of the entire testing dataset. The algorithm improves efficiency through parallelism. Each Mapper produces intermediary results in the form of ($\mathbf{instance}$, $\mathbf{output}_{mi}$), where $\mathbf{instance}_i$ is the key and $\mathbf{output}_{mi}$ denotes the output of the $\mathbf{m}^{\mathbf{th}}$ Mapper. Upon completion of all Mappers, a Reducer starts processing and merging all outputs of Mappers with the same key and writing the result baxk in HDFS.

Algorithm 4.1: EdgeHGN_MRv1 (Classification Algorithm for Large Datasets)

Input: $\mathbb{T}$ **Output:** **AA**

(1) $\mathbf{n}$ Mappers and **one** Reducer, each Mapper constructs an EdgeHGN subnet

(2) Divide $\mathbb{T}$ into $\{t_1, t_2, t_3 \ldots t_n\}$, $\cup_{i=1}^{n} t_i = \mathbb{T}$

(3) Each Mapper builds an EdgeHGN subnet and inputs t_i where $t_i \in \mathbb{T}$

BAA ← new GN Bias Associative Array

For all term $t_i \in \mathbb{T}$ do

Calculate Adjacency Comparison Function (algorithm 3.3)

Calculate Bias Index (algorithm 3.4) & Update $\mathbf{BAA}_i$

(4) Mapper outputs (, $\mathbf{BAA}_i$)

(5) Reducer collects and merges all (, $\mathbf{BAA}_i$)

For all term **BAA** $(i = 1, 2, \ldots \mathbf{n})$ do

Calculate SI Module function (algorithm 3.1)

Calculate Voting function (algorithm 3.2)

Calculate **AA** ← Store/Recall

(6) Repeat (3), (4) and (5) until $\mathbb{T}$ is traversed and all testing data are processed

(7) Reducer outputs (**AA**) and writes it back into HDFS

4.3.2 EdgeHGN_MRv2

EdgeHGN_MRv2 deals with scenarios where the training data are voluminous. To perform an effective classification task, the training data are split into data chunks so they can be processed by Mapper functions in parallel. In such cases, each Mapper still creates the same EdgeHGN_MR classifier but this time using only a subset of the training dataset to train the EdgeHGN network. Assuming $\mathbb{T}$ represents a training dataset, as illustrated in Figure 4.2, *EdgeHGN_MRv2* splits $\mathbb{T}$ into $\mathbf{n}$ data chunks, where each data chunk t_i is proceeded by a Mapper function as part of the training phase:

$$\Upsilon = \bigcup_{1}^{n} t_i \quad , \quad \{\forall t \in t_i \mid t \notin t_n, i \neq n\} \tag{4.1}$$

It is worth noting that each of the Mapper functions in the Hadoop clusters form an EdgeHGN subnet where t_i denotes the training input data to be consumed by the Mapper i. Hence, each EdgeHGN function within a Mapper results in a classifier output based on the training data fed into the network:

$$(Mapper_i, EdgeHGN_i, t_i) \longrightarrow Classifier_i \tag{4.2}$$

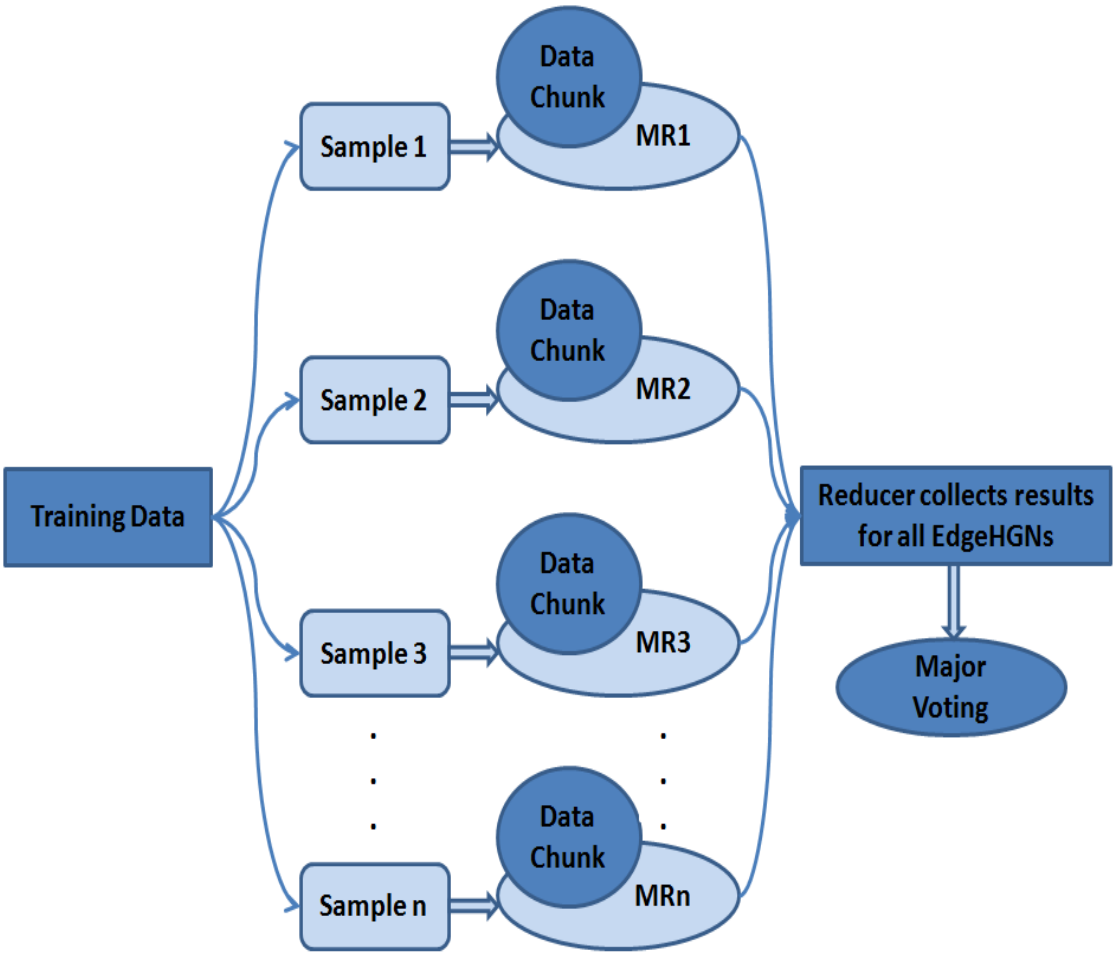

Figure 4.2: EdgeHGN_MRv2 architecture

To minimise the computational costs, each of the classifiers is only trained with a subset of the original training data. However, this can result in degradation of accuracy because each Mapper is trained using only a subset of the training data and the complete training dataset. To ensure we can still achieve acceptable levels of classification accuracy, *EdgeHGN_MRv2* utilises a balanced bootstrapping approach by combining a number of weak learners to produce a much stronger learner.

4.3.2.1 Bootstrapping

Researchers have already shown that finding a strong learner is more complex than training different classifiers using a single training dataset. To overcome this complexity, one well-known approach suggested is to conduct re-sampling of the training dataset, which can be implemented using bootstrap aggregating techniques such as bootstrapping and majority voting. Alham (2011) suggested balanced bootstrapping as a promising technique when looking for strong learners. In the balanced bootstrapping approach, an effort is made to ensure that bootstrap samples contain each training instance equally, but this is not guaranteed, and there might be some cases where bootstrap samples do not include all training instances. One possible approach for constructing balanced bootstrap samples is to form a sequence of instances $\mathbf{A_1}, \mathbf{A_2}, \ldots, \mathbf{A_n}$ that can be repeated $\mathbf{K}$ times to yield a sequence of $\boldsymbol{\mathcal{B}_1}, \boldsymbol{\mathcal{B}_2}, \ldots, \boldsymbol{\mathcal{B}_{Kn}}$. Then a random permutation integer value $\boldsymbol{\rho}$ is chosen from the range of integer values between **1** and $\mathbf{K_n}$. As a result, the initial bootstrap sample can be constructed from $\boldsymbol{\mathcal{B}_\rho(1), \mathcal{B}_\rho(2), \ldots \mathcal{B}_\rho(n)}$, while the second sample is formed from $\boldsymbol{\mathcal{B}_\rho(n+1), \mathcal{B}_\rho(n+2), \ldots \mathcal{B}_\rho(2n)}$, and this process goes on until the $\mathbf{K^{th}}$ bootstrap sample is achieved from $\boldsymbol{\mathcal{B}_\rho((K-1)n+1), \mathcal{B}_\rho((K-1)n+2), \ldots \mathcal{B}_\rho(Kn)}$.

4.3.2.2 Algorithm Design

EdgeHGN_MRv2 starts functioning by first splitting the training data into a number of datasets using the balanced bootstrapping mechanism:

$$\text{Balanced Bootstrappin of } \mathcal{T} : \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \ldots, \mathcal{T}_n\} \quad , \quad \mathcal{T} = \bigcup_{1}^{n} \mathcal{T}_i \tag{4.3}$$

where $\mathcal{T}_i$ stands for the i^{th} training subset and n denotes the total number of training data chunks. It should be noted that each training subset $\mathcal{T}_i$ is stored in one file in the HDFS. Each training instance t_k, $t_k \in \mathcal{T}$, where $1 \leq k \leq \text{length}(\mathcal{T}_i)$ is defined in the format of (**instance**$_k$, **target**$_k$, **type**), where:

i. **instance**$_k$ denotes a bootstrapped training instance t_k that is the input of an EdgeHGN subnet
ii. length $(\mathcal{T}_i)$ represents the length of $\mathcal{T}_i$ and it shows the number of inputs of an EdgeHGN subnet
iii. **target**$_k$ shows the desirable output in the case of **instance**$_k$ being a training instance
iv. **type** field consists of two values – ***train*** and ***test*** which illustrates the type of **instance**$_k$; if the ***test*** value is used, the **target**$_k$ field value should be set to null.

When *EdgeHGN_MRv2* starts processing the input, each Mapper function builds one EdgeHGN subnet and starts feeding the subnet one record from the HDFS input file in the form of (**instance**$_k$, **target**$_k$, **type**). By parsing the input, the Mapper function can determine if the type field is set to ***train*** or ***test***. If the type indicates the value of ***train***, then the input is fed into the base layer of the EdgeHGN subnet for processing. The training phase continues working on the input until all instances with the type value set to ***train*** are processed. Now it is time for the network to start processing and classifying all of the testing instances. Upon processing the testing instances, each Mapper produces intermediary results showing classification outputs in the form of (**instance**$_k$, ***output***$_{nk}$), where instance$_k$ is the key and ***output***$_{nk}$ denotes the output of the n^{th} Mapper. Upon completion of all Mapper tasks, a Reducer starts processing and merging all of the outputs of Mapper functions with the same key. Before calculating the store/recall result, the Reducer function performs majority voting and produces the output in the form of (**instance**$_k$, **output**$_{vk}$) where **output**$_{vk}$ stands for the voted classification decision for **instance**$_k$. Algorithm 4.2 depicts the pseudo-code for *EdgeHGN_MRv2*.

Algorithm 4.2: EdgeHGN_MRv2 (Classification Algorithm for Large Training Datasets)

Input: $\mathbb{T}$ Output: AA

(1) **n** Mappers and **one** Reducer, each Mapper constructs an EdgeHGN subnet

(2) Bootstrap $\{\mathbb{T}_1, \mathbb{T}_2, \mathbb{T}_3, \ldots, \mathbb{T}_n\}$, $\mathbb{T} = \cup_1^n \mathbb{T}_i$

(3) Each Mapper builds an EdgeHGN subnet and inputs t_k where $t_k \in \mathbb{T}_i$,

$1 \leq k \leq$ length ($\mathbb{T}_i$)

BAA ← new GN Bias Associative Array

For all term $t_k \in \mathbb{T}$ do

Calculate Adjacency Comparison Function (algorithm 3.3)

Calculate Bias Index (algorithm 3.4) & Update $\mathbf{BAA_k}$

(4) Mapper outputs (t_k , $\mathbf{BAA_k}$)

(5) Reducer collects and merges all (t_k , $\mathbf{BAA_{kj}}$) , **j** = (**1, 2,…, n**) and computes **BAA**

using Majority Voting function

$$\textbf{Compute BAA} = \max_{k=1}^{length\,(\mathbb{T}_i)} \sum_{j=1}^{n} \mathbf{BAA_{kj}}$$

Calculate SI Module function (algorithm 3.1)

Calculate **AA** ← Store/Recall

(6) Repeat (3), (4) and (5) until $\mathbb{T}$ is traversed

(7) Reducer outputs (**AA**) and writes it back into HDFS

4.3.3 EdgeHGN_MRv3

EdgeHGN_MRv3 is designed to work with cases where there is a large number of processing neurons in the EdgeHGN network. To achieve high classification accuracy, *EdgeHGN_MRv3* parallelises and distributes EdgeHGN processes across Mapper functions so each Mapper only utilises a subset of the processing neurons for training. Hence, each Mapper function may contain one or more processing neurons. It should be noted that there will be a number of iterations as part of this MapReduce

cluster setup to execute the algorithm with **L** layers. To implement those iterations, *EdgeHGN_MRv3* utilises **L-1** MapReduce jobs as shown in Figure 4.3.

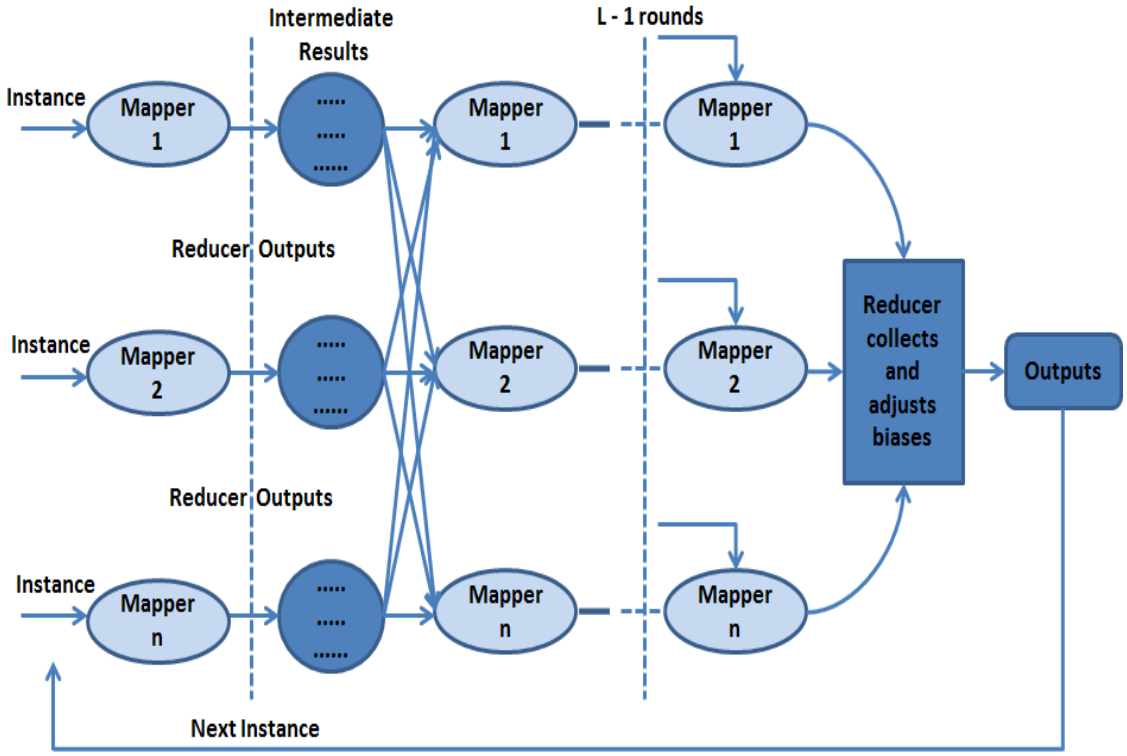

Figure 4.3: EdgeHGN_MRv3 Architecture

To guarantee the data flow between the map and reduce operations within each iteration layer, data instances will be presented to the network in the form of (**index$_\kappa$, instance$_i$, target$_i$**), where:

i. **index$_\kappa$** denotes the **κ^{th}** Reducer

ii. **instance$_i$** denotes the **i^{th}** data instance, which can be either a *training* or a *testing* instance.

iii. **target$_i$** stands for the expected output if the training instance **instance$_i$** is fed into the network.

All data instances and data format entry information are stored in a file in the HDFS so that they can be processed when the *EdgeHGN_MRv3* starts functioning. The number of MapReduce operation layers, **L**, is determined by the size of the EdgeHGN bias array. One major difference between *EdgeHGN_MRv3* and its predecessor versions, *EdgeHGN_MRv1* and *EdgeHGN_MRv2*, is that *EdgeHGN_MRv3* aims to maintain the neural network parameters based on the input data format entries rather than initialising an explicit EdgeHGN network. When *EdgeHGN_MRv3* starts its execution cycle, each Mapper function initially reads a data record from the HDFS, performs some computations and then generates the <*key* , *value*> pair output where $\mathbf{index_{\kappa}}$ will form the MapReduce key, while the MapReduce value will be represented in the form of ($\mathbf{index_{\kappa}}$ **,** $\mathbf{instance_{\iota}}$ **,** $\mathbf{output_{j}}$ **,** $\mathbf{target_{\iota}}$), where $\mathbf{output_{j}}$ denotes the computational result for neuron **j**. The $\mathbf{index_{\kappa}}$ parameter assures that the output of the Mapper functions is collected and processed by the $\boldsymbol{\kappa}^{\mathbf{th}}$ reducer, maintaining the EdgeHGN network state in a consistent fashion during the execution phase of the algorithm. The output from the Mappers will be presented to the $\boldsymbol{\kappa}^{\mathbf{th}}$ reducer in the form of ($\mathbf{index_{\kappa'}}$, $\mathbf{output_{j}}$, $\mathbf{target_{\iota}}$). It is obvious that these $\boldsymbol{\kappa}$ Reducer functions can produce $\boldsymbol{\kappa}$ number of outputs. The $\mathbf{index_{\kappa'}}$ in the Reducer output format explicitly instructs the $\boldsymbol{\kappa'}$ Mapper function to start processing the relevant output data file. As a result, the number of Mappers for the next layer of processing can be calculated based on the number of Reducer output files that will represent the number of input files for the succeeding layer. Upon receiving the input from the previous layer, Mappers start their processing and generate outputs for Reducers, and this process continues until the execution phase reaches the last round of processing. In this last round of execution, Mapper functions first process ($\mathbf{index_{\kappa}}$ **,** $\mathbf{output_{j}}$ **,** $\mathbf{target_{\iota}}$) and generate output in the form of ($\mathbf{output_{j}}$, $\mathbf{target_{\iota}}$). Then one single Reducer function gathers the output results from all Mappers in the form of ($\mathbf{output_{j1}}$, $\mathbf{output_{j2}}$ . . . $\mathbf{output_{j\kappa}}$, $\mathbf{target_{\iota}}$) and writes the result ($\mathbf{output_{\iota}}$, $\mathbf{target_{\iota}}$) back into the HDFS. This process continues by reading the next instance from the input data file until all instances are processed. Algorithm 4.3 depicts the pseudo-code for *EdgeHGN_MRv3*.

Algorithm 4.3: EdgeHGN_MR_3 (Classification Algorithm for Networks with Large Processing Neurons)

Input: $\mathbb{T}$ Output: AA

(1) $\mathbf{n}$ Mappers and $\mathbf{n}$ Reducers

(2) Initially each Mapper inputs ($\mathbf{index_{\kappa}}$, $\mathbf{instance_i}$, $\mathbf{target_i}$) where $\mathbf{instance_i}$ denotes the i^{th} data instance, $\mathbf{target_i}$ stands for the expected output if the training instance $\mathbf{instance_i}$ is fed into the network and $\mathbf{index_{\kappa}}$ denotes the κ^{th} Reducer

(3) Mapper outputs ($\mathbf{index_{\kappa}}$, $\mathbf{instance_i}$, $\mathbf{output_j}$, $\mathbf{target_i}$) $\mathbf{k = (1, 2 \ldots n)}$ where $\mathbf{output_j}$ denotes the computational result for neuron $\mathbf{j}$

(4) The κ^{th} Reducer gathers output ($\mathbf{index_{\kappa'}}$, $\mathbf{output_j}$, $\mathbf{target_i}$), $\kappa' = \mathbf{(1, 2 \ldots n)}$ where the $\mathbf{index_{\kappa'}}$ in the Reducer output format explicitly instructs the κ' Mapper function to start processing the relevant output data file

(5) Each Mapper builds an EdgeHGN subnet and inputs $\boldsymbol{t_m}$ where $\boldsymbol{t_m} \in \mathbb{T}$,

$\mathbf{1 \le m \le length\ (\mathbb{T})}$

$\mathbf{BAA} \leftarrow$ new Boolean Associative Array

For all term $\boldsymbol{t_m} \in \mathbb{T}$ do

Calculate Adjacency Comparison Function (algorithm 3.3)

Calculate Bias Index (algorithm 3.4) & Update $\mathbf{BAA_m}$

(6) Mapper outputs ($\mathbf{BAA_m}$, $\mathbf{target_i}$)

(7) $\mathbf{n^{th}}$ reducer collects and merges all ($\mathbf{BAA_m}$, $\mathbf{target_i}$) from $\mathbf{n}$ Mappers

For all term $\mathbf{BAA_m\ (1 \le m \le length\ (\mathbb{T}))}$ do

Calculate SI Module function (algorithm 3.1)

Calculate $\mathbf{AA} \leftarrow$ Store/Recall

Writing the result ($\mathbf{output}$, $\mathbf{target_i}$) back into HDFS

(8) Retrieve $\mathbf{instance_{i+1}}$ and $\mathbf{target_{i+1}}$

Update input to ($\mathbf{index_{\kappa}}$, $\mathbf{instance_{i+1}}$, $\mathbf{target_{i+1}}$)

(9) Repeat (2), (3), (4), (5), (6), (7), (8) until all input dataset instances are processed

4.4 Performance Evaluation

All three EdgeHGN_MR implementations are set up in a Hadoop-based framework to evaluate their respective performance when dealing with real-world big dataset examples. The experimental Hadoop cluster was configured with eight *DataNodes* and one *NameNode*. The *NameNode* machine acts as both *JobTracker* and *NameNode*, while each of the eight *DataNodes* act as both *TaskTracker* and *DataNode*. The Hadoop cluster configuration details are listed in Table 4.1.

Table 4.1: Hadoop Cluster Details

NameNode	CPU: Core i7 @ 3.2 GHz Memory: 16 GB , SSD: 1 TB OS: Redhat Linux
DataNode	CPU: Core i7 @ 3.8 GHz Memory: 32 GB , SSD: 500 GB OS: Redhat Linux
Network bandwidth	1 Gbps
Hadoop version	2.5.2, 64 bits
Java version	OpenJDK 1.6
JVM heap size	16 GB

For the training and testing dataset, the MNIST (Mixed National Institute of Standards and Technology) dataset is chosen (MNIST Database, 2012). The MNIST is an enormous set of handwritten digits that is commonly used for testing large-scale classification systems (see Figure 4.4). The database includes 60,000 training images and 10,000 testing images. Each image sample has 784 dimensions. That is, each MNIST data sample is a 28 x 28 array of integers in the range of 0 – 255, depicting the intensity of the blackness of the image at that location. The number of instances varies from 100 to 10,000 to calculate the accuracy rate of the scheme, and the size of the dataset varies between 1MB and 1GB to test the computational efficiency of the approaches. To ensure we can achieve a fair experimental result, all tests were repeated five times, and the average calculated value of all five attempts was used.

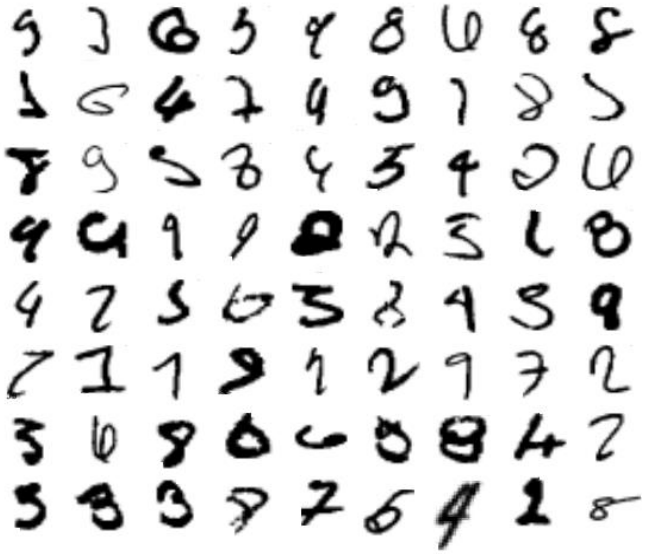

Figure 4.4: Handwritten digits (MNIST Database)

4.4.1 Classification Accuracy

To evaluate the accuracy rate of *EdgeHGN_MRv1*, a different number of training instances was used. However, the maximum number of training and testing data instances was capped at 10,000. Accuracy rate is calculated as per equation (4.4):

$$\textbf{Accuracy rate} = \frac{\textbf{True Positive}}{\textbf{True Positive} + \textbf{False Positive}} \tag{4.4}$$

Figure 4.5 illustrates the accuracy rate result of the *EdgeHGN_MRv1* classification approach when using sixteen Mappers. *The results show that increasing the number of training instances can result in an improved classification precision rate*. To evaluate the performance of *EdgeHGN_MRv2*, the maximum number of training and testing data instances was capped at 10,000. Subsets of the training dataset were used by 16 Mappers in the scheme to generate classification results of 10,000 testing instances, utilising both bootstrapping and majority voting techniques. Each of the Mappers in the network inputs various numbers of training instances ranging from 100 to 1000.

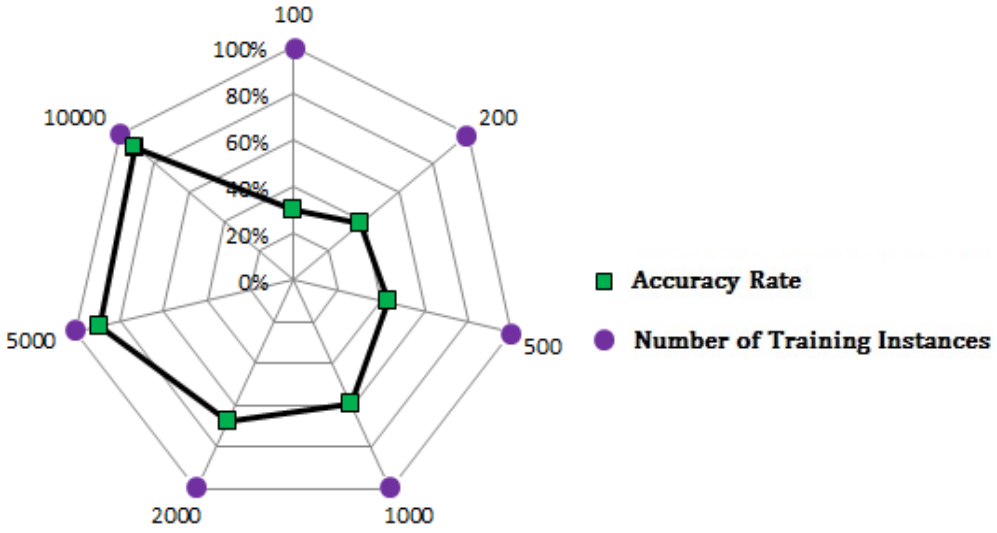

Figure 4.5: Accuracy rate of EdgeHGN_MRv1

Figure 4.6 illustrates the accuracy rate result for *EdgeHGN_MRv2*. As shown, *the precision rate of the scheme keeps improving by the increase in the size of the training instances in each EdgeHGN network*. Considering the same size training datasets, *EdgeHGN_MRv2* achieves higher accuracy rates compared with *EdgeHGN_MRv1* for all cases of the training dataset.

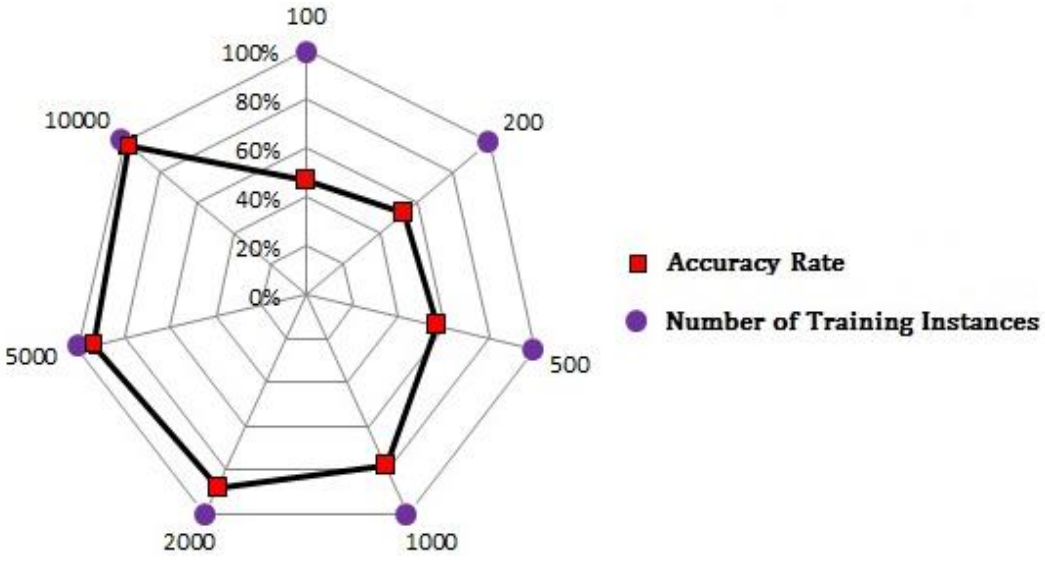

Figure 4.6: Accuracy rate of EdgeHGN_MRv2

EdgeHGN_MRv3 utilises a fully distributed Hadoop-based EdgeHGN setup to perform a classification task using a large number of processing neurons. Figure 4.7 illustrates the accuracy rate for an *EdgeHGN_MRv3* scheme with 16 Mappers. Again, *the precision rate improves along with the increase in the size of the training dataset*.

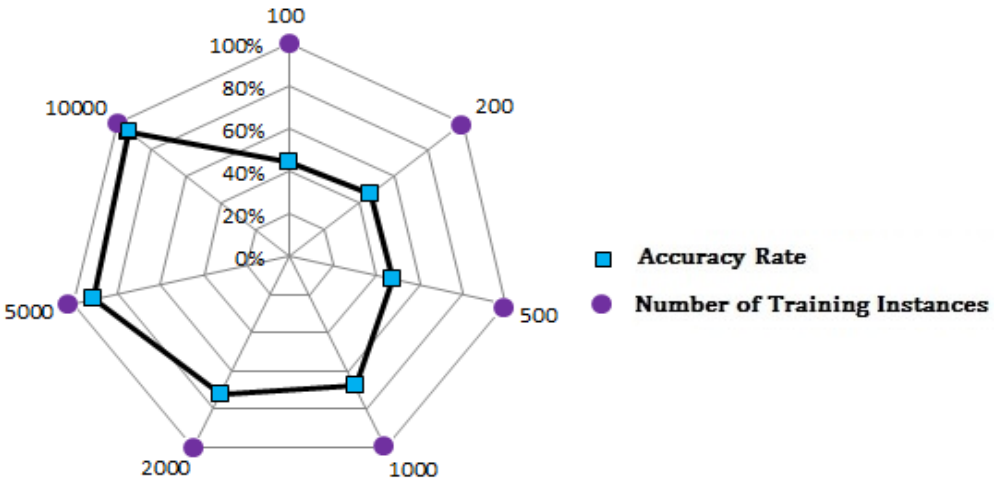

Figure 4.7: Accuracy rate of EdgeHGN_MRv3

Figure 4.8 shows the accuracy rate comparison between all three schemes. *While EdgeHGN_MRv1 and EdgeHGN_MRv3 perform similarly, EdgeHGN_MRv2 outperforms the other two because the scheme makes use of strong learners by utilising bootstrapping and majority voting techniques*.

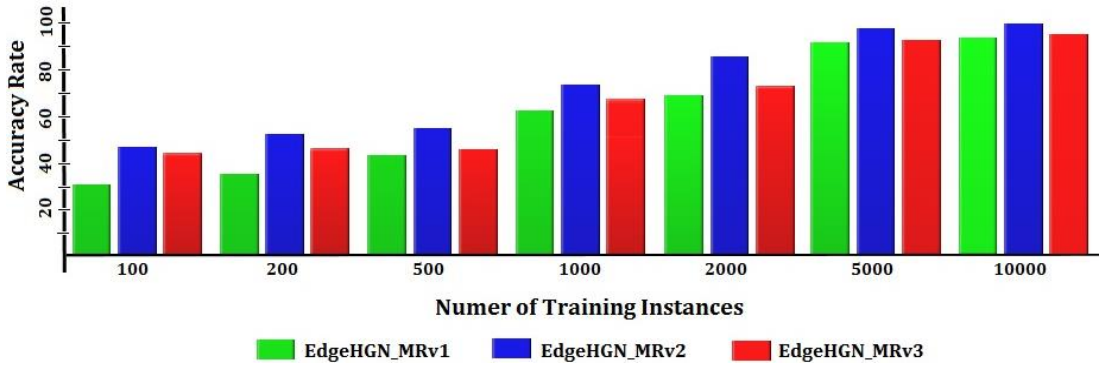

Figure 4.8: Accuracy rate comparison between EdgeHGN_MRv1, EdgeHGN_MRv2 and EdgeHGN_MRv3

Figure 4.9 illustrates the consistency and stability of all three algorithms for their five individual runs. *As shown, EdgeHGN_MRv2 exhibits high stability compared with the other two approaches, again due to the use of bootstrapping and majority voting techniques to achieve strong learners*.

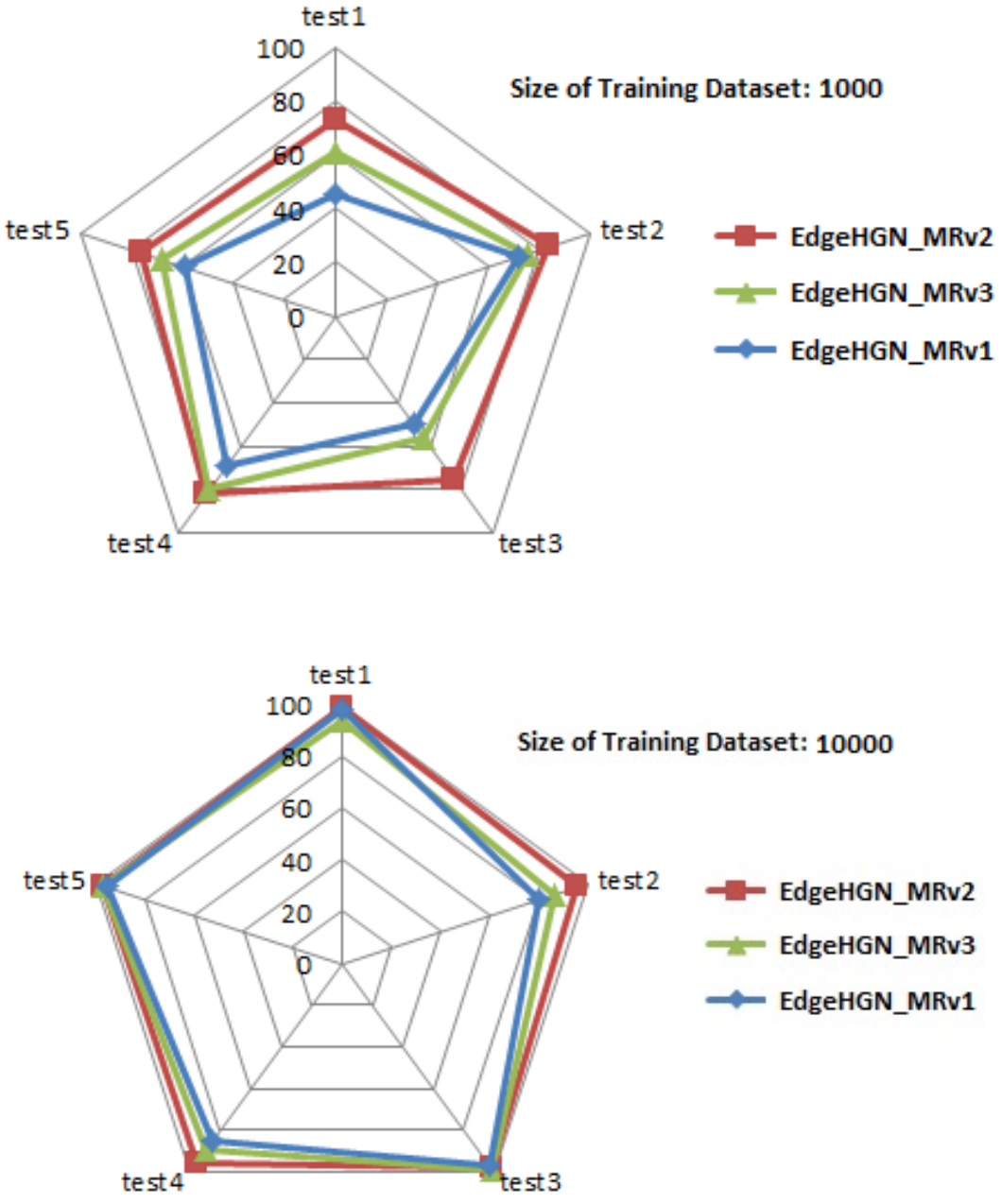

Figure 4.9: Consistency and stability rate comparison between EdgeHGN_MRv1, EdgeHGN_MRv2 and EdgeHGN_MRv3

4.4.2 Computational Efficiency

To evaluate the computational efficiency of the EdgeHGN-based MapReduce scheme when dealing with large datasets, a number of experiments have been carried out to examine the effect of an increase in the size of the dataset on the execution time of the algorithm. Figure 4.10 illustrates the execution time of all three schemes with 16 Mappers for processing datasets of various sizes, ranging from 1MB to 1GB.

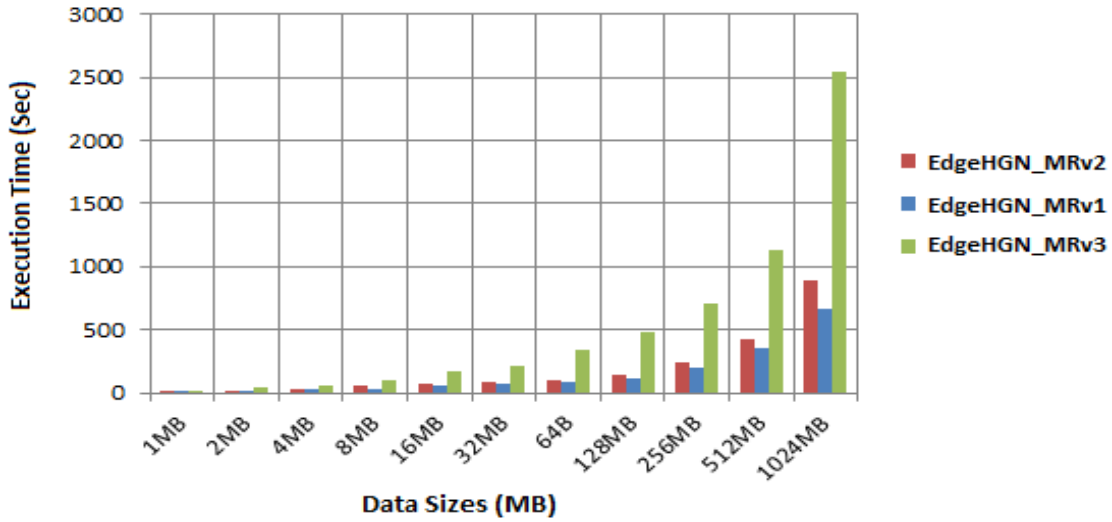

Figure 4.10: Computational efficiency comparison between EdgeHGN_MRv1, EdgeHGN_MRv2 and EdgeHGN_MRv3

As shown above, *the computational cost of EdgeHGN_MRv1 and EdgeHGN_MRv2 are reasonably low when dealing with large data sizes because both schemes distribute the testing data and computational load among all eight DataNodes in the Hadoop cluster and utilise parallel processing to scale with the increase in the size of input dataset.* However, EdgeHGN_MRv3 incurs an additional computational overhead when compared with the other two approaches because both EdgeHGN_MRv1 and EdgeHGN_MRv2 implement training and classifications tasks within one MapReduce job and, as a result, Mappers and Reducers are required to be initialised and started once. Conversely, EdgeHGN_MRv3 includes multi-stage Hadoop processes of stopping and starting Mappers and Reducers, resulting in extended execution times. This is an area that can be improved in future work by

looking into enhancing methods of in-memory processing to better empower MapReduce operations working on data-intensive scenarios with many iterations.

4.5 Comparative Performance Results

Our proposed EdgeHGN-based MapReduce scheme (EdgeHGN_MR) is primarily focused for use within the clouds and it is fundamentally different from all published approaches in data management. The large heterogeneous datasets created for the case studies will provide an excellent resource to compare and contrast the one-shot learning, scalability and accuracy of our approach with a number of well-established data management techniques, which in turn will generate detailed information on trade-offs and benefits. In this regard, performance results already derived from various experiments are promising. While EdgeHGN outperforms its former GN extensions – HGN and DHGN – in terms of accuracy and processing time, as discussed in Chapter 3, it also provides comparable performance benchmarks when tested against well-known large-scale data management schemes such as Distributed MapReduce (Dean & Ghemawat, 2004), Google Pregel (Percolator, Dremel & Pregel, 2012), Microsoft Dryad (Isard. et. al., 2007) and GraphLab (GraphLab Open Source, 2009). To prove this, a series of experiments have been conducted, and the results are presented in the following sub-sections.

4.5.1 EdgeHGN-based MapReduce versus Hadoop MapReduce

In one of the conducted experiments, the aim was to process webserver logs to examine the HTTP sessions accessed by users of particular webpages. One important factor in this test was to evaluate the overall time spent by the end-user for each requested page. This time metric can be utilised later to gain more information about visitors and to better design a website structure. For the purpose of this test, the extracted log files were from NASA Kennedy Space Center (2014), which stores two months' worth of entire HTTP requests to the NASA Kennedy Space Center WWW server in Florida. The file format is in ASCII, with one line per request. In each line,

various data fields are captured, representing host, timestamp, request, HTTP reply code and bytes in the reply. Traditional data-processing approaches working on relational data will not fit the purpose for the processing of this huge log file size of more than 50,000,000 request lines due to the semi-structured nature of log data, along with its excessive volume. To evaluate the HTTP session information, Internet protocol (IP) address, timestamp and URL data are extracted from log files and stored in the HDFS. The session affinity data provide all web pages visited by a particular IP address with a unique page identity and timeout value. In most cases, this timeout value should not exceed 30 minutes; otherwise, a new session with a new identity will be generated for that IP address. The session time-span can be determined by calculating timestamp differences for the same IP from the time of logging in to the point of logging out. All log data fields can be input into map tasks, where the output of map tasks includes the session affinity number as the key and all other fields in the data log files as values. Then the Reducer function calculates the overall time-span for each session ID (key) and generates the final output in the form of (IP, Session ID, Time). Table 4.2 illustrates the results of processing the log files using EdgeHGN_MR and Hadoop MapReduce run in fully distributed mode. The version of Hadoop implementation was 2.2.0 at the time of conducting this test.

Table 4.2: Processing time comparison between EdgeHGN_MR and MapReduce

Data Count (rows)	File Size (MB)	MapReduce	EdgeHGN-based MR
10,000	2MB	14 Seconds	5 Seconds
100,000	25MB	68 Seconds	26 Seconds
1,000,000	246MB	92 Seconds	72 Seconds
10,000,000	2412MB	173 Seconds	119 Seconds

As shown in table 4.2, the distributed operation suits better for processing large data volumes. In fact, processing small data with the distributed operation is not

desirable because the time it takes to collect the distributed data during a reduce operation within the same processing node outweighs the advantages of distributing data chunks between map functions. Conversely, *EdgeHGN-based MapReduce works well in dealing with both small and large size data counts due to its parallel one-shot learning mechanism, where the size of the input data has a minimal effect on the time of its single-cycle in-network processing*.

4.5.2 EdgeHGN-based MapReduce versus Pregel-like Graph Processing Systems (Giraph, GPS, Mizan and GraphLab)

Pregel is an efficient, scalable and fault-tolerant framework that enables large-scale graph processing using simple code, and it is capable of performing computations over large graphs in a very fast fashion while hiding relevant distribution details behind an abstract API (Malewicz, et. al., 2010). Its architecture is inspired and developed by the Bulk Synchronous Parallel (BSP) model (Valiant, 1990), which empowers the programmer to come up with parallel-computing solutions for a specific problem without the hassle of knowing how communication and memory allocations are performed in a distributed setup. To minimise the communication overhead, Pregel tries to preserve the data locality by moving the computations to where the data reside. Pregel is considered a simple parallel processing framework with many opportunities for improvement, and it has led to the invention of several other graph processing approaches, including Apache Giraph (Apache Giraph, 2013), GPS (Salihoglu & Widom, 2013), Mizan (Khayyat, et. al., 2013) and GraphLab (Low, et. al. 2010) which is now referred to as PowerGraph (Gonzalez, et. al. 2012).

Apache Giraph is one of the schemes selected for the purpose of performance evaluations. The reason Apache Giraph was selected as one of the processing frameworks to compare EdgeHGN_MR against is that it offers an open-source alternative for Pregel where worker processes are run as map-only tasks on top of the HDFS data structure (Apache Giraph, 2013). Apache Giraph v1.0.0 is chosen due to its voluminous developer and user base, which includes Facebook. GPS is another

open-source implementation of the Pregel scheme taken for our experiments. The reason for this selection is that GPS has been implemented in many graph processing experimental systems, thereby providing confidence in the approach. It also performs comparatively with Giraph v1.0.0, and it previously outperformed Giraph 0.1 (12 x faster) (Salihoglu & Widom, 2013). Mizan is another open-source implementation of Pregel selected for our performance benchmarking. As it provides similar graph data-processing approach as Giraph and GPS, it is a good candidate for our performance evaluation (Khayyat, et. al., 2013). Finally, GraphLab is chosen as another open-source implementation for large-scale graph processing, mainly due to its popularity and maturity for handling graph processing tasks (Low, et. al. 2010).

4.5.2.1 System Setup and Datasets

All Experiments are designed to run on setups of two, four and eight Amazon EC2 spot instances. Table 4.3 lists the setup details for our experiment.

Table 4.3: Experiments Setup Details

Amazon EC2 Instances (2 , 4 , 8)	4 Virtual CPUs: 8 Xeon 1.7GHz Memory: 16GB OS: Ubuntu 12.04.1
Tested Frameworks	Giraph v1.0.0 GPS rev 110 Mizan 0.1bu1 GraphLab 2.2
Network bandwidth	1Gbps
Hadoop version	1.0.4
Java Version	jdk1.6.0 30
JVM Heap Size	16GB

Table 4.4 illustrates the datasets **eT** and **eF** used for evaluating the performance of each scheme. These datasets are taken from Stanford Network Analysis Project (2015). All datasets are recorded in the HDFS as uncompressed ASCII formatted text files.

Table 4.4: Dataset Details

ego-Facebook (eF) **(Social circles from Facebook)**	4K vertices 88K edges
ego-Twitter (eT) **(Social circles from Twitter)**	81K vertices 1.77M edges

4.5.2.2 PageRank Algorithm

PageRank is one of the first selected algorithms to evaluate the performance of schemes in-scope due to its popularity and simplicity. It could also represent the memory, computation and communication challenges when processing large scale data. The PageRank algorithm is a well-known scheme used to express the relative *importance* of webpages computed based on the hyperlink structure and by weighting each incoming link to a page (Brin & Page, 1998). The PageRank of the webpage $\boldsymbol{x}$ is calculated by the recurrence equation as:

$$\mathbf{PR}[x] = \frac{1-\lambda}{\mathbf{n}} + \lambda \sum_{\mathbf{y\ links\ to}\ x} \frac{\mathbf{PR}[x]}{\mathbf{OutLinks}[y]} \tag{4.5}$$

Where $\boldsymbol{\lambda}$ is the random reset probability and $\boldsymbol{n}$ is the number of WebPages. Since the PageRank value for page $\boldsymbol{i}$ is dependent on the PageRank of those pages which are linked to page $\boldsymbol{i}$, the recurrence formula is applied in an iterative fashion until the PageRank of each page converges. For this performance testing, a PageRank scheme with damping factor (λ) of **0.85** is selected, the same value as it is taken in (Malewicz, et. al., 2010). This damping factor value means that, assuming a user is browsing a webpage, there is an 85% chance of switching to a random webpage link from the outgoing links of the current visited page, and a 15% chance of switching to a random webpage taken from the entire web (the input graph). The PageRank algorithm has an elegant MapReduce implementation. The mapper emits initial PageRank values for every node. The reducer receives all PageRank contributions for a given node, adds them up, and emits its contribution to its own outgoing links (See Figure 4.11).

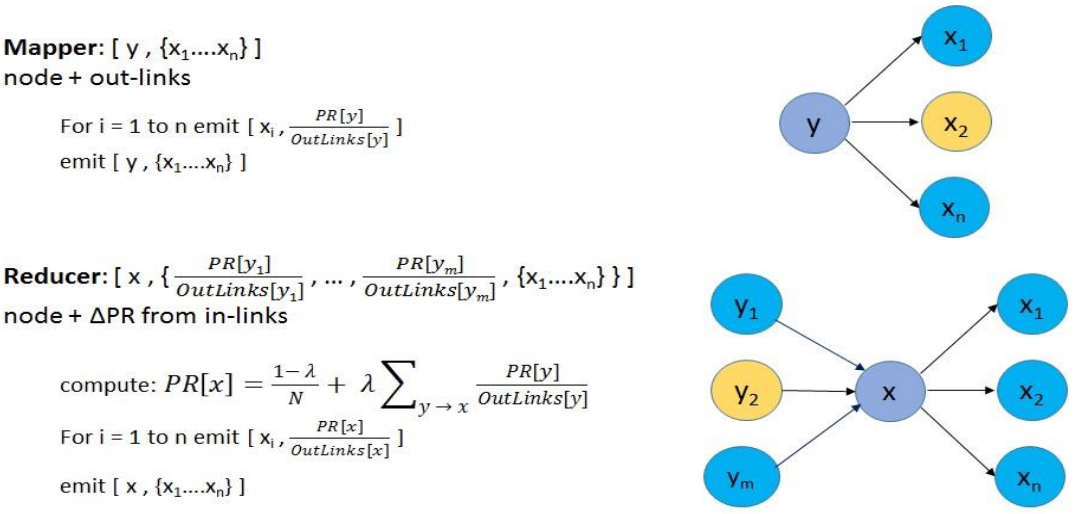

Figure 4.11: MapReduce implementation of PageRank algorithm where the mapper emits initial PageRank values for every node. The reducer receives all PageRank contributions for a given node, adds them up, and emits its contribution to its own outgoing links.

As shown in Figures 4.12 and 4.13, for processing *ego-Facebook* and *ego-Twitter* datasets using the PageRank algorithm, GraphLab outperforms all other schemes due to being capable of performing adaptive computations. Conversely, Mizan performs poorly for all conducted experiments. The reason for this slow computational time is that Mizan implements a slow graph partitioning process separately from computation, in which it tries to process the graph by conducting multiple large reads and writes to the HDFS. Giraph and GPS also perform very closely, while Giraph takes slightly longer than GPS to conduct computations. *EdgeHGN_MR outperforms Giraph and Mizan for both datasets in most setups, but it consumes more time conducting computations compared with GPS and GraphLab. The reason for this slower computational time is mainly because of longer initialisation and start-up times. In fact, while EdgeHGN is capable of producing parallel single-cycle computations, due to the utilisation of the number of Mappers and Reducers, initialisation and start-up times contribute to longer computation times. Nevertheless, EdgeHGN's execution time is comparable with other state-of-the-art techniques in the literature.*

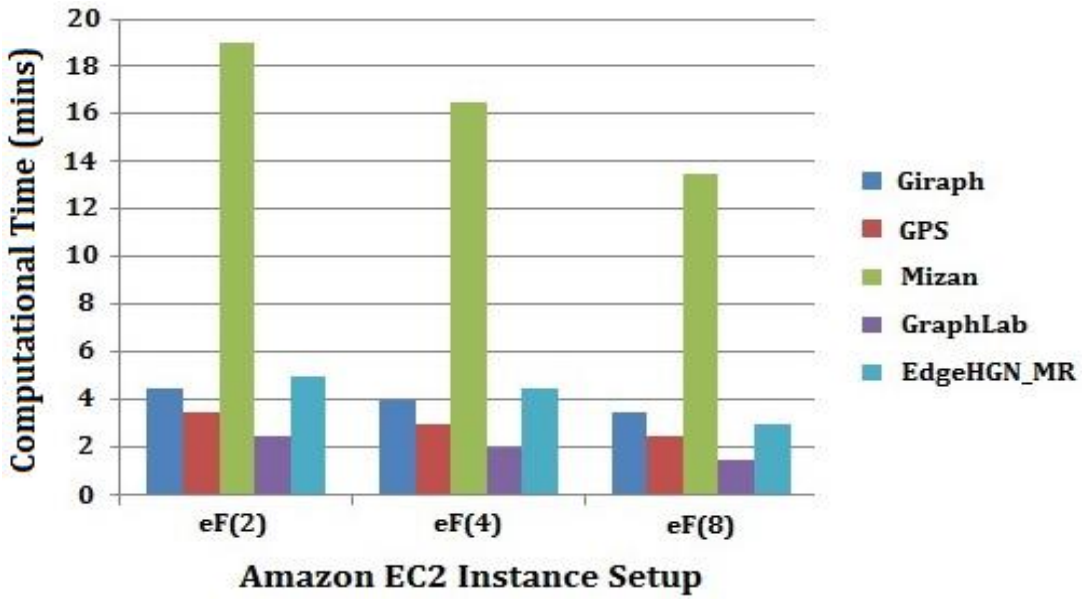

Figure 4.12: Computing time comparison between Giraph, GPS, Mizan, GraphLab and EdgeHGN_MR using PageRank algorithm for ego-Facebook Dataset

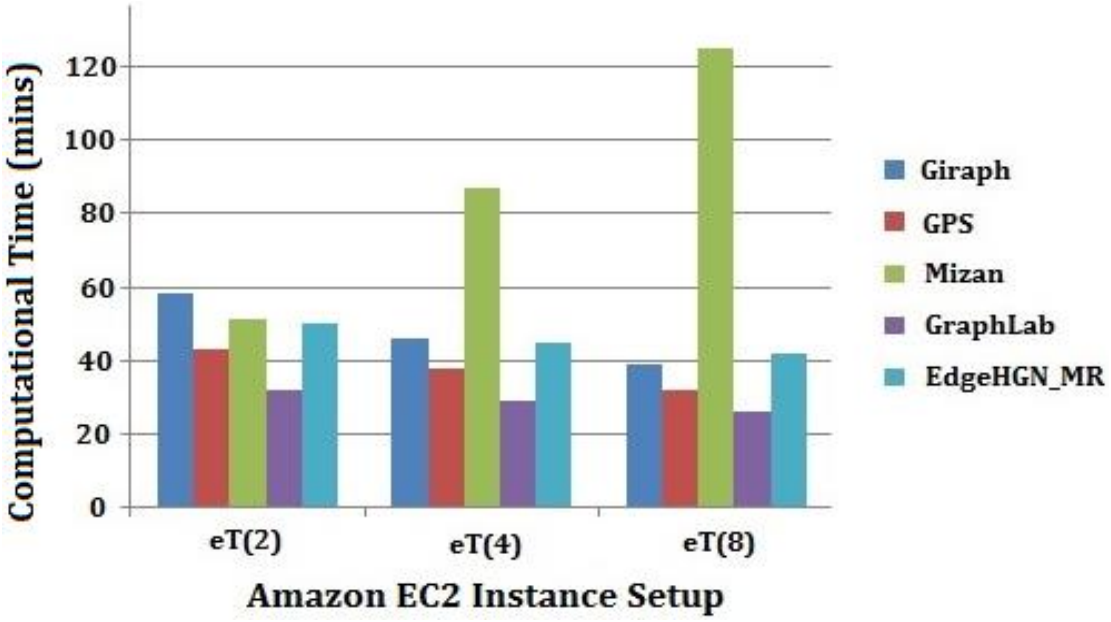

Figure 4.13: Computing time comparison between Giraph, GPS, Mizan, GraphLab and EdgeHGN_MR using PageRank algorithm for ego-Twitter Dataset

Figures 4.14 and 4.15 show the maximum memory usage for the Giraph, GPS, Mizan, GraphLab and EdgeHGN_MR schemes processing *ego-Facebook (eF)* and *ego-Twitter (eT)* datasets using PageRank algorithm with Amazon EC2 cluster setups of two, four and eight machine instances. As shown, GPS demonstrates remarkable memory efficiency across all experiments due to its built-in scheme optimisations such as canonical objects, reducing the cost of allocating multiple java objects (Salihoglu & Widom, 2013). Mizan's memory usage efficiency is improved by adding more resources, but it downgrades when larger graphs are being processed. This is again due to its lack of built-in system optimisations. In terms of memory requirements, Giraph performs poorly when compared with most schemes, with the exception of Mizan. The reason for this poor memory usage efficiency is that it utilises memory-inefficient adjacency list data structures to implement graph mutations. GraphLab generally performs reasonably well across all experiments in terms of memory usage due to its wide built-in system optimisations. Similar to GraphLab, *EdgeHGN_MR demonstrates efficient memory usage across all experiments. This is because in EdgeHGN_MR, memory requirements per GN node to maintain the bias array do not increase disproportionately with the increase in the number of stored graphs*.

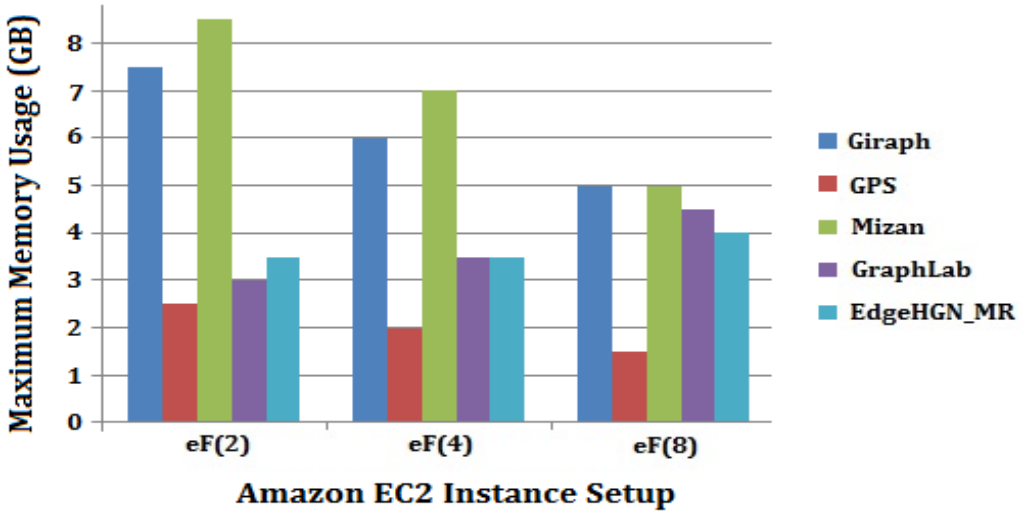

Figure 4.14: Maximum memory usage comparison between Giraph, GPS, Mizan, GraphLab and EdgeHGN_MR using PageRank algorithm for ego-Facebook Dataset

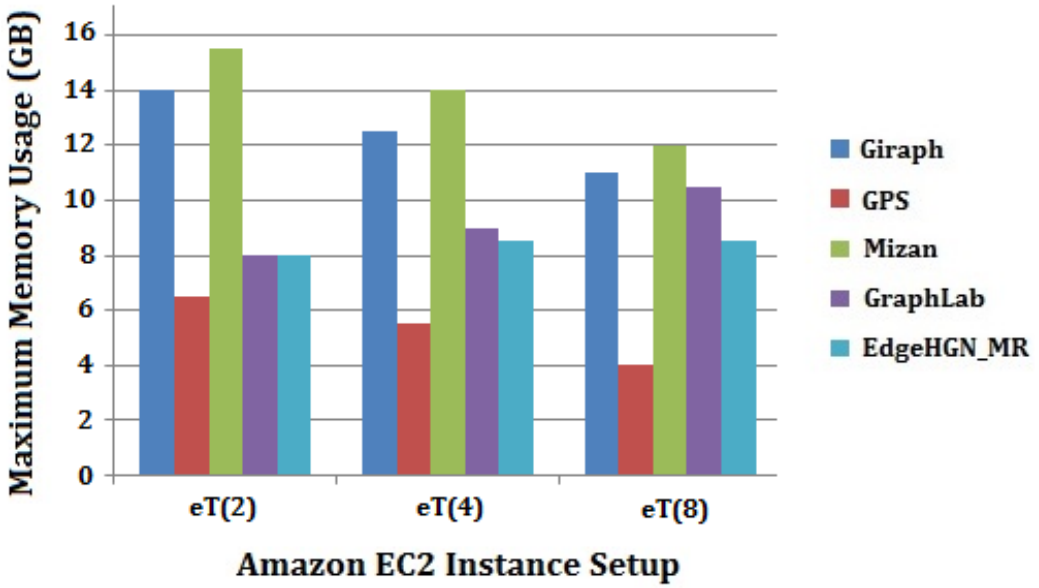

Figure 4.15: Maximum memory usage comparison between Giraph, GPS, Mizan, GraphLab and EdgeHGN_MR using PageRank algorithm for ego-Twitter Dataset

4.6 Conclusion

Existing large-scale data-processing schemes such as MapReduce involve isolating basic operations within an application for data distribution and partitioning. This excludes their applicability to many applications with complex data dependency considerations. MapReduce models when used with complex data requirements generally entail additional difficult and error-prone application-level customisations. Adding higher and complex data representations within the model will vastly improve its usability and provide an important – pattern recognition based – data analysis option. Also our loosely coupled associative computing (GN-based) method, EdgeHGN, as discussed in Chapter 3, provides means to deliver dynamic data management. Hence, the goal of this chapter was to create a formal model, methods and prototypical realisations of a combination of MapReduce – as architectural patterns for parallel processing structures for widely distributed computing – with AM concepts, represented in EdgeHGN scheme. The aim was to achieve algorithms that provide demonstrably more efficient, robust and scalable end-to-end data access to distributed real-time information for clouds through applying an access scheme

that can enable fast data retrieval across multiple records and data segments associatively, utilising a parallel approach.

To address these, in this chapter, three parallel EdgeHGN based MapReduce schemes were introduced and discussed in detail. *EdgeHGN_MRv1*, *EdgeHGN_MRv2* and *EdgeHGN_MRv3* each utilise EdgeHGN network processing model in conjunction with MapReduce computational framework to effectively deal with data-intensive scenarios in the face of excessive amount of classification data to process, voluminous training datasets or massive number of processing neurons in the network, respectively. Proposed schemes in this chapter preserve the strengths of the MapReduce model and eliminate/alleviate most of its constraints in a well-integrated manner by replacing referential data access mechanisms with more versatile and distributable associative functions, which allow complex data relations to be easily encoded into the keys as patterns. These patterns can be applied in a variety of applications requiring content recognition e.g., image databases, search within large machine log files and data mining. Algorithmic strengths of the MapReduce approach was investigated for the first time in context with the effectiveness of one-shot learning-based parallelism provisioned via our distributed pattern recognition approach, EdgeHGN. The principle of associative-memory-based learning was implemented through the use of hierarchically connected layers, with local feature learning at the lowest layer and upper layers combining features into higher representations. The EdgeHGN-based MapReduce approach to cloud-based data processing is unique. It elevates the MapReduce key-value scheme to a higher level of functionality by replacing the purely quantitative key-value pairs with higher-order data structures that will improve the parallel processing of data with complex associations (or dependencies). By using an associative key-value framework, we can deal with data in any form and in any representation simply by using a pattern matching model (including fuzziness) that treats data records as patterns and provides a distributed data access scheme that enables balanced data storage and retrieval by association.

Our experimental results show that EdgeHGN based MapReduce works exceptionally well in dealing with both small to large size data counts due to its parallel one-shot learning mechanism where the size of input data has minimal effect on the time of its single-cycle in-network processing. Performance evaluation results against state-of-the-art parallel processing techniques in the literature (Distributed MapReduce, Giraph, GPS, Mizan and GraphLab) demonstrate that the performance of MapReduce parallelism as a scalable scheme for data processing in clouds can be significantly improved by transforming the data processing operations into one-shot distributed pattern matching sub-tasks, in which distributed computations are performed in-network, enabling data storage and retrieval by association (instead of pre-set referential data access mechanisms). In the next chapter we will further demonstrate applicability of EdgeHGN_MR approach to distributed data processing within fine-grained wireless sensor networks with limited resource considerations.

Chapter 5

EdgeHGN Application in Fine-grained Wireless Sensor Networks (WSNs)

This chapter investigates the capabilities of the proposed EdgeHGN scheme in the context of distributed data processing in large-scale cloud of wireless sensor networks (WSNs). The chapter will examine WSNs as a platform of operation for EdgeHGN distributed pattern matching to demonstrate the ability of the proposed recognition technique to learn and recognise complex patterns using minimal information and resources to effectively perform classification tasks. The rapid technological advancement of wireless technologies and the increasing miniaturization of RF micro electro-mechanical systems have resulted in the advancement of small and tiny computational systems, such as WSN technology. These inter-connected computing devices create a computational network that is capable of offering a frontline processing platform for various purposes, such as event detection and remote monitoring. Such networks are mainly referred to as fine-grained networks since they normally consist of a large group of connected tiny computing nodes with limited on-board resources for power, storage, and processing. In their widely acclaimed article,

Wireless sensor networks 2010 – 2020, Peter Harrop and Raghu Das write about billions of sensors guarding us against events such as avalanches, hurricanes, forest fires, failures of critical services and assisting in hospitals and with traffic through the use of the wireless sensor network (WSN) (Harrop & Das, 2010). In spite of the enormous potential existing for WSN use, the fact is that so far WSN technology has been mostly implemented for humble applications such as meter reading in buildings and simple forms of ecological monitoring. Current approaches mainly address the issue of conveying retrieved sensory data to a central entity referred to as base station for most of the processing which can create bottlenecks in the system, resulting in less scalable networks. As a result, and to address scalability concerns, WSN networks require new generation of processing schemes which are capable of processing their sensory findings internally to produce highly condensed and sophisticated outputs within the network. This approach can eliminate the bottleneck problem by offering on-site computations through adoption of a completely distributed and decentralised technique.

In order to build such a framework as stated above, the first step will be to establish a level of computability within the WSN in a way that sensory readings can be immediately translated into and represented as event patterns, so that they can later be locally processed and analysed by the network in a purely distributed fashion. This approach will entail two-fold benefit. On one hand, translation of sensory data into patterns can improve event detection (e.g. surveillance) and on the other hand due to the distributed nature of pattern matching techniques deployed with the WSN network, this approach can yield scalable processing schemes for future large-scale networks. The main challenge here is to evolve a real-time approach, which is capable of processing complex real-life patterns generated from various types of sensors, and to create an efficient computational model for processing WSN heterogeneous datasets. For this purpose, events of interest initially need to be correlated to particular pattern classes of our definition. Upon completion of this translation phase, implementing a distributed pattern matching approach, such as EdgeHGN, would assist us with integrating large volume of sensor nodes into a smart

monitoring platform for observing phenomenon of interest which in turn can bring unprecedented functionalities within our reach for performing large-scale distributed data processing within resource-constrained WSNs.

Application of parallel pattern matching approach within fine-grained WSN networks has been previously discussed by Khan and Mihailescu (2004). As part of their research findings, they presented a distributed recognition technique for locating stress patterns from a basic finite element model stored within WSN network. In the research work conducted by Baqer, et. al. (2005) and Baig, et. al. (2006), pattern recognition schemes in WSN are further investigated with a major focus on GN-based pattern matching/event detection approach. Later, Nasution and Khan (2008) proposed a more advanced distributed pattern recognition technique for event detection based on their Hierarchical Graph Neuron (HGN) associative memory model offering.

The motivation for this chapter lies in the aforementioned applications using GN-based algorithms. In this regard, EdgeHGN, with its light-weight distributable computational model along with its high scalability features can be a suitable candidate for performing distributed on-site computations within WSNs. With the ability of parallelising the computational process within the body of the network, EdgeHGN enables recognition scheme to be implemented on tiny resource-constrained computing devices such as sensor nodes in WSN network. In a complete distributed setup, each GN is assigned to a single compute node, and the collaborations of such inter-connected compute nodes form an EdgeHGN subnet. The simple EdgeHGN light-weight bias array search mechanism makes this configuration well-suited for fine-grained WSN networks that suffer from limited processing and storage capabilities. In this chapter we will demonstrate EdgeHGN as a lightweight and distributed event detection scheme that simplifies the existing WSN infrastructure and develops a single-cycle learning pattern recognition capability within the WSN for event detection. In this chapter, we will also demonstrate the robustness and scalability of the EdgeHGN for performing distributed recognition tasks over a fine-grained network.

5.1 Distributed Data Processing Scheme for WSNs

Many of existing data processing approaches deployed in WSNs suffer from highly complex computations, iterative learning procedures and large training set requirements which restrict their use as a suitable method that can easily scale up to meet large-scale WSN resource-constrained operational requirements. In fact, majority of such schemes often apply conventional neural network techniques or machine learning methods that need extensive amount of retraining as well as large number of training datasets for their effective generalisation. Furthermore, the centralised processing or single-processing approach used in existing methods puts a practical burden on developing scalable schemes for WSNs. As an example, the constant flow of retrieved sensory data will produce extensive communication overheads. In addition, re-routing procedures and relocation activities of sensor nodes that regularly happen in real-time applications will result in significantly long delays in detecting critical events and these even get worse when computational bottlenecks are present in the network. These limitations make WSN even a less suitable candidate for applications with large-scale data processing requirements. Therefore, we are in need of a new approach for data processing within WSN that enables processing to be conducted within the body of the network in situ and with decentralised manner and generates highly condensed data outputs internally within WSN. Having such an internal processing setup will alleviate the bottleneck issue within WSN through on-site computations, and improves performance by minimising the processing delay experienced using the existing methods. Artificial neural networks (ANNs) and other machine learning schemes are the most commonly deployed classification methods for performing event detection in WSNs. Some of these approaches implement the Kohonen Self-Organizing Map (SOM) (Kohonen, 2000) or other activation-based neural networks, such as the Radial Basis Function (RBF) neural network (Yang & Paindavoine 2003). However, due to their extensive learning complexity as well as their highly cohesive training-validation approach, these methods cannot scale up effectively to the dynamics of the WSN.

5.1.1 WSN Event Detection

On a macro scale, a WSN comprises a network of wireless sensor nodes that are linked and connected together through a common entity, referred to as the base station or sink. Because of restricted on-board resources available in terms of limited power and processing capabilities, communications between sensor nodes and the base station usually involve a series of data aggregation techniques to reduce data exchange and minimise the volume of traffic routed to the base station. In fact, issues with WSN deployment are mainly due to their resource-constrained characteristics, which include restricted communication bandwidth, limited power and processing capability and limited memory capacity (Culler, Estrin and Srivastava, 2004). Research in the area of event detection in WSN is commonly classified into performance-specific research and application-specific research. The performance-specific research is more concerned with the efficiency of the event detection method. The main research goal in this area is to develop event classification techniques with minimum energy consumption and extended lifetime of the WSN network. On the other hand, application specific research focuses on the development of event detection methods that provide accurate and reliable detection strategy for predefined applications such as intrusion detection or phenomenon detection. The following section will further describe these two common research areas.

5.1.1.1 Performance-specific Event Detection Schemes

Most of the recent research works on performance-specific event detection schemes are focused on developing efficient localisation and routing mechanisms for WSN. Localisation and routing are the two important factors in determining the optimum coverage and performance of a WSN network. A collaborative event detection and tracking in wireless heterogeneous sensor networks has been proposed by Shih, Wang, Chen and Yang (2008). In this research, emphasis has been put into tracking procedure and localization of sensors attribute region for event detection. Banerjee, Xie and Agrawal (2008) introduces multiple-event detection scheme with fault tolerant within WSN. They propose the use of polynomial-based scheme that

addresses the problems of Event Region Detection (PERD). There are two-components involved, including event recognition and event report with boundary detection. For event recognition, they adopt min-max classification scheme which classifies event according to the sensor reading values. These values would then be transformed into polynomial coefficients and passed through a data aggregation scheme. The proposed event detection scheme has enabled a 33% savings in the communication overhead experienced by the network. The development of energy-efficient scheme for event detection within WSN has also been carried out by Baqer (2008) using GN pattern recognition scheme with voting capabilities. This work provides a foundation for energy-efficient pattern recognition scheme to be deployed within WSN infrastructure for real-time applications such as structural health monitoring (SHM). Cellular Weighted Graph Neuron (CwGN) was later proposed by Alfehaid (2013), as a distributed in-network processing paradigm that depends on local computations and adopts weighting technique that searches for pattern edges and boundaries. The model addresses the constraints of timing requirements by allowing a number of CwGN networks to perform recognition operations in a parallel paradigm. The research in this chapter intends to extend the capabilities of parallel pattern recognition scheme using a more scalable EdgeHGN distributed approach.

5.1.1.2 Application-specific Event Detection Schemes

Application-specific schemes for event detection refer to the area of research involving development of application middleware for WSN. This middleware provides enhanced capability and accuracy for event detection using sensor networks. Several machine learning algorithms have been applied by a number of research studies, including Fuzzy-ART neural network, multi-layer perceptrons (MLPs), and Self-Organizing Maps (SOMs). The use of Adaptive Resonance Theory (ART) neural network for event tracking was introduced by Kulakov and Davcev (2005). In these research, the use of artificial neural networks (ANNs) in the form of an ART network has been used as pattern classifier for event detection and classification. The scheme offers reduction in communication overhead with only cluster labels being sent to the

sink, instead of the overall sensory data. However, the implementation of ART neural network incurs excessive iterative cycle to achieve optimum cluster matches. The research by Kulakov and Davcev (2005) on ART neural network for event tracking has also been further researched by Li and Parker (2008) in their study on intruder detection using a WSN with fuzzy-ART neural networks. Self-organisation for event detection has also been a major focus in application specific research within WSN networks. Catterall et. al., (2003) propose a concept of distributed event classification through the use of Kohonen self-organising map (SOM) approach (Kohonen, 2000). The occurrence of events, which are signified by changes in sensor parameter values, could be mapped into clusters representation. The proposed scheme however, imposes significant iterative learning procedure and the classification process is carried out on each input unit, rather than collective input units.

5.1.1.3 Distributed Pattern Recognition Scheme within WSN

It should be noted that, any algorithm that may entail computations, communications, and storage resources within a sensor node would lead to a rapid exhaustion of the limited battery power available per node. This implies the simple fact that the data processing and communication must be minimal in order to conserve limited energy and computational resources of sensors (Khan & Muhamad Amin, 2009). To address this concern, system designers must be able to come up with a well-managed setup for WSN deployment that includes principles such as data-centric processing approach, localised algorithms and lightweight middleware. Current schemes deployed for event detection in WSN commonly involves a centralised processing phase at the sink or base station. Efforts to reduce the tendency for this singular processing stage base have been shown in both performance and application-specific research works. However, a complete decentralisation strategy has yet to be realised. This new scheme should solve the existing issues of complex learning algorithms and tightly-coupled techniques that are currently being deployed for event detection.

In the following sections, a new design for event detection in WSN is introduced and discussed that incorporates the above discussed principles for highly-scalable

sensor networks. This chapter proposes a holistic solution for event detection in WSN. The proposed scheme incorporates a distributed pattern recognition approach within WSN network and provides on-site and localised computation. The remaining of this chapter details the implementation of EdgeHGN single-cycle learning distributed pattern recognition algorithm. Within this scheme, a dimensionality reduction approach has been employed for minimising the need for complex computations. The proposed scheme is also capable of providing scalable detection, enabling allowance for the outgrowth of event classes. Details of the EdgeHGN distributed pattern recognition scheme can be further referred to as in Chapter 3.

5.2 Integrated EdgeHGN-WSN Processing Scheme

Using distributed nature and lightweight features of EdgeHGN, an event detection scheme for WSN network is able to be carried out at the sensor node level. In fact, it acts as a front-end middleware that could be deployed within each sensor nodes in the network, building a network of event detectors. As a result, our proposed scheme minimises the processing load at the base station and provides near real-time detection capability. Preliminary work on EdgeHGN integration for WSN has been conducted in (Basirat & Khan, 2013). In a fully distributed EdgeHGN configuration, a collection of sensor nodes collaborate and form an EdgeHGN subnet to perform event detection based on the sensory readings obtained from the environment (See Figure 5.1). Note that the SI module will be implemented in a controlling node, such as the base station, or a super-node. The EdgeHGN subnet module is located within each WSN subnet that is located in a specific sensory region. The event classification process (evaluation of event/non-event signals) in the EdgeHGN event detection scheme is a dual-layer process. The first layer focuses on the sub-pattern recognition in the EdgeHGN subnet, whereas the second layer involves pattern classification using a voting scheme that is conducted by the SI module. Sub-pattern recognition is the process of determining the recall/store status of an input sub-pattern that is conducted in EdgeHGN subnets.

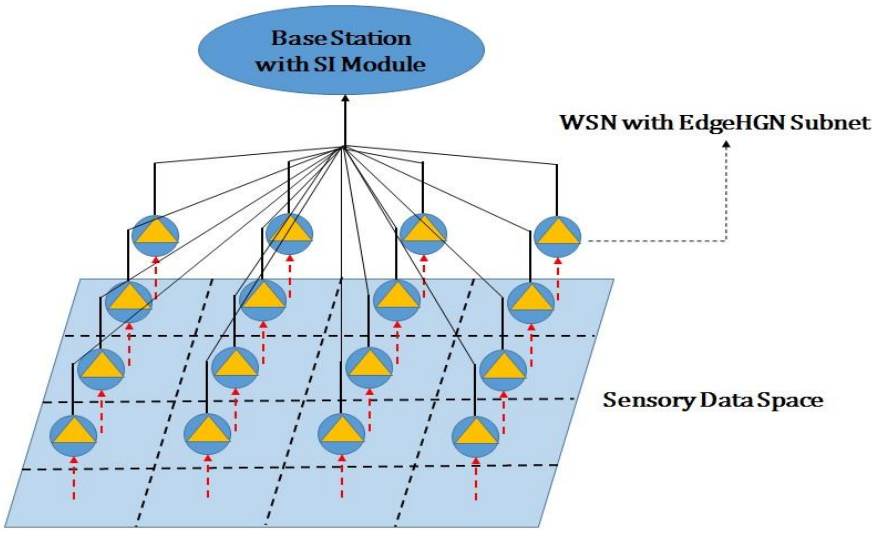

Figure 5.1: EdgeHGN distributed event detection framework

The output of this process is either a recalled index of the stored sub-pattern or a new index for the input sub-pattern. This index is sent to the SI module for pattern classification. Note that the EdgeHGN considers an event as a pattern that represents a state of normality or abnormality for the entire sensory network. For complex event detection (multiple sensory schemes), each EdgeHGN subnet is mapped to a sensor node using a clustered configuration. We examine the deployment of the WSN in a two-dimensional plane with $\boldsymbol{n}$ sensors, represented by a set $\boldsymbol{S} = (\boldsymbol{S_1}, \boldsymbol{S_2}, \ldots \boldsymbol{S_n})$, where $\boldsymbol{S_i}$ denotes the $\boldsymbol{i}^{\text{th}}$ sensor. The sensors are uniformly placed in a grid-like area, $\mathbf{A} = (\boldsymbol{x} * \boldsymbol{y})$, where $\boldsymbol{x}$ represents the x-axis coordinate of the grid area, and $\boldsymbol{y}$ represents the y-axis. Each sensor node is assigned to a specific grid area (See Figure 5.2). The location of each sensor node is represented by the coordinates of its grid area ($\boldsymbol{x_i}$, $\boldsymbol{y_i}$). For the communication model, a single-hop mechanism for data transmission from the sensor node to the sink is proposed. The "*auto-send*" approach is also used to minimise errors associated with the loss of packets during data transmission (Saha & Bajcsy, 2003). Communication between the sink and the sensor nodes is performed using a broadcast method. It should be noted that due to the front-end processing approach, the proposed scheme does not involve massive transmissions of sensory readings to the sink.

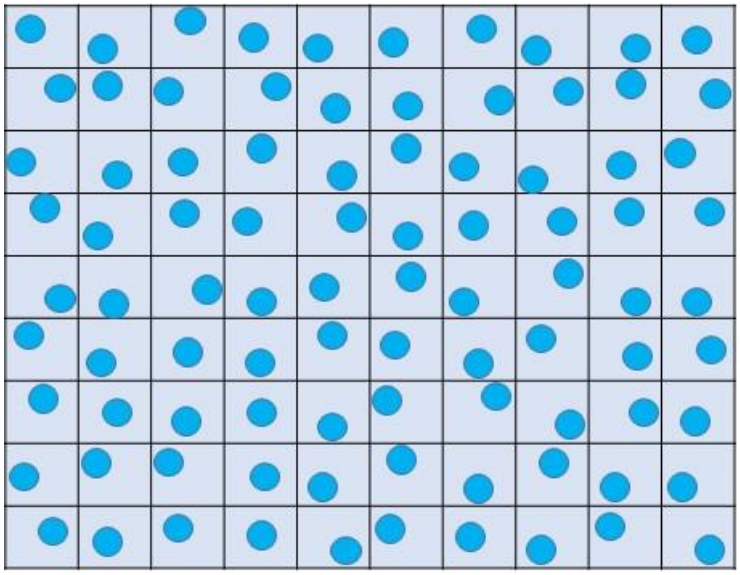

Figure 5.2: Sensor node placement in a Cartesian grid where each node is allocated to a specific grid area

5.2.1 Dimensionality Reduction in Sensory Data

Event detection usually involves the process of recognising significant changes or abnormalities in sensory readings. In heterogeneous sensor networks, sensory readings are of different types and values, e.g., temperature, light intensity and pressure. For the EdgeHGN implementation, the input sensory data must be first pre-processed and translated into a proper format while maintaining the integrity and accuracy of the readings. For this purpose, in our EdgeHGN implementation and to perform dimensionality reduction, adaptive threshold binary signature scheme is used to produce standardised format for the input pattern from various sensory readings. The binary signature is a condensed representation of various types of data with different values in a binary format (Nascimento & Chitkara, 2002). Given a set of $\boldsymbol{n}$ sensory readings $\boldsymbol{S} = (\boldsymbol{S_1}, \boldsymbol{S_2}, \ldots \boldsymbol{S_n})$, each reading $\boldsymbol{S_i}$ would have its own set of $\mathbf{K}$ threshold values $\boldsymbol{T}(\boldsymbol{S_i}) = (\boldsymbol{T_1}, \boldsymbol{T_2}, \ldots, \boldsymbol{T_K})$, showing different levels of acceptance. These values could also represent acceptable value range for the input. The following steps show how the adaptive threshold binary signature method is being implemented:

i. Each sensor reading $\boldsymbol{S_i}$ is first discretised into **j** binary bins ($\boldsymbol{\mathcal{B}_i} = \boldsymbol{b_1^i\ b_2^i}$ ... $\boldsymbol{b_j^i}$) of equal or varying capacities. The number of bins used for each data is determined by the number of threshold values $\boldsymbol{T}(\boldsymbol{S_i})$. In fact, $\boldsymbol{\mathcal{B}_i}$ is used to signify the presence of data that is either equivalent to the threshold value or within a range of the specified $\boldsymbol{T_i}$ values using a binary representation.

ii. Each bin would correspond to each of the threshold values. Consider a simple data as shown in Table 5.1. If the temperature reading is between the range 40-45 degrees Celsius, the third bin would be activated. Thus, a signature for this reading would be 00100.

iii. The final format of the binary signature for all sensory readings could be represented as a list of binary values that correspond to a specific data value, in the form of $\boldsymbol{S} = \boldsymbol{b_1^1\ b_2^1\ b_1^2\ b_2^2}$ ... $\boldsymbol{b_j^n}$, where $\boldsymbol{b_j^k}$ represents the binary bin for $\mathbf{K^{th}}$ sensory reading and $\mathbf{j^{th}}$ threshold value.

Table 5.1: Temperature readings example with their respective binary signature

Temperature Threshold Range (◦C)	Binary Signature
0 – 20	10000
21 – 40	01000
41 – 60	00100
61 – 80	00010
81 – 100	00001

5.2.2 EdgeHGN Event Classification

The EdgeHGN distributed event detection method conducts a bottom-up classification approach, in which the classification of events is determined from the sensory readings obtained through the WSN. The approach first pre-processes the input patterns and implements dimensionality reduction technique using the adaptive threshold binary signature method. These input patterns are propagated to all available EdgeHGN subnets for performing recognition and classification tasks. The recognition process involves determining differences between the input patterns and

the previously stored ones. While similar patterns will be recalled by the EdgeHGN network, any dissimilar pattern will trigger a response for further analysis. This scheme uses the supervised single-cycle learning approach in EdgeHGN processing algorithm to perform event classification based on the previously stored patterns. It should be noted that the stored patterns in our proposed model include a set of ordinary events that are transformed into regular surrounding/environmental conditions. These patterns are determined from the analysis conducted at the base station and is based on the continuous feedback from the sensor nodes. The event classification approach using the EdgeHGN incorporates twos levels of recognition: the front-end recognition and the back-end recognition. EdgeHGN Front-end recognition is the process of pattern matching that determines if the sensory readings retrieved from the sensor network indicate an abnormal reading or a normal surrounding condition. On the other hand, the spatial occurrence detection is performed as part of the back-end recognition phase where signals (patterns) sent by sensor nodes are processed for classifying event occurrences in a particular area or location.

5.2.2.1 Pattern Matching at Sensor Level

The determination of abnormal events is conducted by deploying a pattern matching approach. Sensory readings are represented as patterns and any significant changes in the structure of normal patterns are of interest and should be classified as events or critical events that must be reported back to the sink (or other master node). By having a clustered EdgeHGN network configuration, each sensor node in the network can be mapped with a particular EdgeHGN subnet. In this network setup, Each EdgeHGN subnet is capable of accepting a number of different sensory readings as a single input sub-pattern. The following algorithm (algorithm 5.1) shows our proposed pattern matching method for event classification at the sensor level. In this scheme, the output of the pattern matching process is a signal that alerts the SI module of the detection/occurrence of a new event. The base station will respond by conducting a spatio-temporal analysis on the readings obtained.

Algorithm 5.1: Pattern Matching Algorithm at the Sensor Level

(1) given $\boldsymbol{n}$ sensory readings for time $\boldsymbol{t}$: $\boldsymbol{\mathcal{S}_t} = (\boldsymbol{\mathcal{S}}_1, \boldsymbol{\mathcal{S}}_2, \ldots \boldsymbol{\mathcal{S}}_n)$
convert $\boldsymbol{\mathcal{S}_t}$ to a binary signature $\boldsymbol{\mathcal{B}_t}$. Therefore, $\boldsymbol{f}(\boldsymbol{binsig})$: $\boldsymbol{\mathcal{S}_t} \mapsto \boldsymbol{\mathcal{B}_t}$
(3) $trigger = FALSE$
(4) $eventAlert.Sensor = FALSE$
(5) $repeat$
$for\ i = 0$ to $MAXREADINGS$ do
{check for matched sensory reading sub-patterns in sensor data}
$if\ new.\boldsymbol{\mathcal{B}_t} = \boldsymbol{\mathcal{S}}[i].Sensor$ then
{$new.\boldsymbol{\mathcal{B}_t}$: new readings, matching process conducted using EdgeHGN}
$exit\ for$
$else$
$\boldsymbol{\mathcal{S}}[MAXREADINGS + 1].eventAlert.Sensor = new.\boldsymbol{\mathcal{B}_t}$
$trigger = TRUE$
$eventAlert.Sensor = TRUE$
$end\ if$
$end\ for$
$until\ trigger = TRUE$
(6) send $eventAlert.Sensor$ and $\boldsymbol{\mathcal{S}}[MAXREADINGS + 1].Sensor$ to SI module function at base station
(7) $MAXREADINGS = MAXREADINGS + 1$

5.2.2.2 EdgeHGN Classification Approach

The pattern matching process within EdgeHGN approach is a dual-layer process. The first layer focuses on the sub-pattern recognition at EdgeHGN subnets, while the second layer involves pattern matching using a voting scheme that is performed by the SI Module. Sub-pattern recognition at EdgeHGN subnets is the process of determining the recall/store status of an input sub-pattern. The result of this process is either a recalled index of the previously stored sub-pattern or a new index generated for the respective input sub-pattern. The results of sub-pattern recognition phase are then sent out to the SI module for implementing pattern classification. Each of the sub-pattern indexes received from EdgeHGN subnets is proceeded and the result of this analysis process are recorded in the form of class labels. It is worth noting that

for supervised classification, the number of class label is fixed, while in unsupervised classification, this number can be incremented. The proposed approach only needs binary input patterns and accepts multiple sensory readings that are used to detect the occurrence of critical events. Given a set of $\boldsymbol{n}$ sensory readings $\boldsymbol{S} = (\boldsymbol{S_1}, \boldsymbol{S_2}, \ldots \boldsymbol{S_n})$, a dimensionality-reduction technique such as binary threshold-signature is used to convert each reading value to its respective binary signature. The threshold binary signature method utilises threshold classes to translate a single data range into a binary format. Given a sensory reading $\boldsymbol{S_i}$ where $\boldsymbol{i}$ = **1, 2, ...** $\boldsymbol{n}$ and with **K** -threshold class, the equivalent binary signature that implies $\boldsymbol{B_i} \rightarrow \boldsymbol{S_i}$ is in the form of $\boldsymbol{B_i} \in \{0, 1\}^{K}$. Hence, $\boldsymbol{n}$-set sensory readings $\boldsymbol{S} = (\boldsymbol{S_1}, \boldsymbol{S_2}, \ldots \boldsymbol{S_n})$ will be converted into a set of binary signatures ($\boldsymbol{B_1}$, $\boldsymbol{B_2}$, ..., $\boldsymbol{B_n}$). If the output index from EdgeHGN subnet matches the previously stored pattern for the critical event, then a signal is sent to the base station in the form of a data packet represented by (*node_id, timestamp, class_id*). The *class_id* parameter stands for the class identification of the event that has been detected by the scheme.

5.3 EdgeHGN-WSN Performance Evaluation

The analysis of EdgeHGN distributed event classification scheme is performed by using a simulation approach. The sensory data taken from the research by Catterall et. al., (2003) have been used to evaluate the performance of our proposed classifier. The test data includes three Smart-It wireless sensor node readings that for various environmental conditions such as *light* (Smart-It 1), *temperature* (Smart-It 2) and *pressure* (Smart-It 3). The first recognition test is performed over Smart-It 1 sensory reading that represented light. The test involves assigning an EdgeHGN subnet to each Smart-It sensor dataset. EdgeHGN retrieves sensory readings in the form of binary representation using the discussed threshold-signature technique. Figure (5.3) shows the results of the recognition test conducted on light sensor dataset Smart-It 1 for 1800 sensory datasets. As it is clearly shown, *EdgeHGN is capable of detecting light event occurrences with remarkable accuracy*.

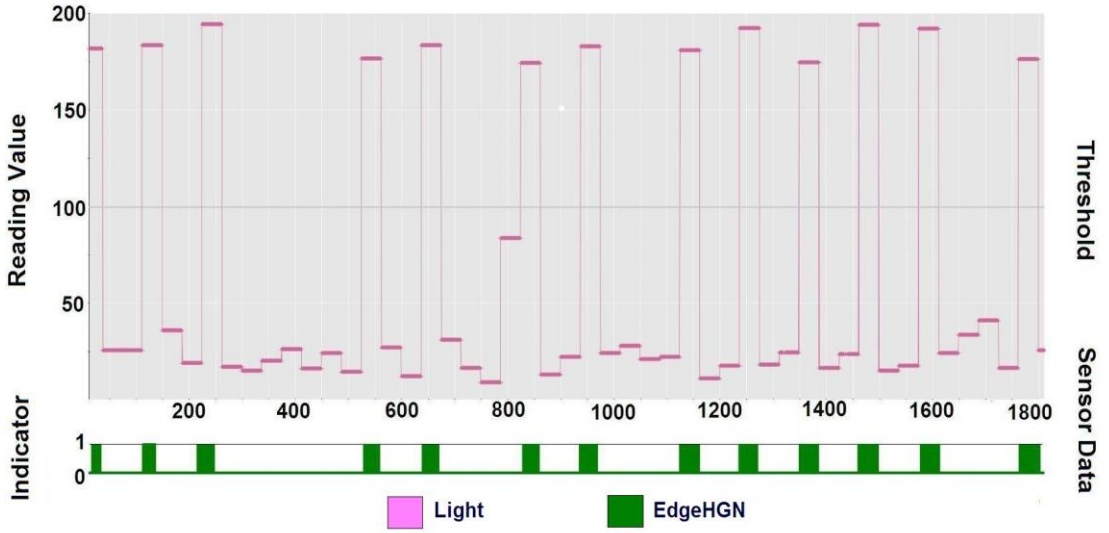

Figure 5.3: EdgeHGN event detection result for a test using 1800 light sensor datasets (Smart-It 1) (x-axis) with a threshold of 100 (Basirat & Khan, 2013)

For performance benchmarking, similar experiments are conducted using support vector machine (SVM) and self-organizing map (SOM). For this comparison test, SVMLight implementation (Joachims, 2008) with both linear-type and 2-degree polynomial kernel and SOM Toolbox with default configuration (Vesanto, et. al., 2010) are used. For the purpose of this exercise, we have calculated *precision*, *recall*, *accuracy* and *error value* parameters as a comparative basis for the classification process. Table 5.2 shows how each of these parameters are defined and represented.

Table 5.2: Recognition parameters with their respective definitions

Recognition Parameters	Definitions
Precision	$\frac{\text{True Positive}}{\text{True Positive+False Positive}}$
Recall	$\frac{\text{True Positive}}{\text{True Positive+False Negative}}$
Accuracy	$\frac{\text{True Positive+True Negative}}{\text{True Positive+True Negative+False Positive+False Negative}}$
Error Value	$\frac{\text{False Positive+False Negative}}{\text{Total Number of Test Objects}}$

In this experiment, SVM and SOM classifiers use a set of six readings in their training set, while EdgeHGN only uses datasets with two entries. For both SVM classifiers (linear SVM and poly-2 SVM), six support vectors have been initialized and implemented for classification purposes. Table 5.3 shows the results of the recognition test performed on Smart-It 1, 2, 3 datasets using different performance parameters for sensory data. The value of the best result (selected feature) for each parameter is underlined. From the results obtained, *EdgeHGN shows high recognition accuracy with very low error value and very high recall rate*. Both SVM and SOM classifiers have also demonstrated reasonably high precision and accuracy. However, their recall value is comparatively low when compared against EdgeHGN. This low recall value is mainly due to the low true positive values obtained during the classification process. In addition, both schemes produce higher error values.

Table 5.3: Comparative analysis on recognition accuracy parameters between EdgeHGN and other classifiers for event recognition using three sensory data obtained from Catterall et al. (2003) (Smart-It 1, Smart-It 2, and Smart-It 3).

Smart-It	Classifier	Precision	Recall	Accuracy	Error
1	EdgeHGN	1.0000	0.8851	0.9973	0.0023
	Linear SVM	0.9375	0.7419	0.9888	0.0112
	Poly-2 SVM	1.0000	0.7538	0.9905	0.0095
	SOM	1.0000	0.8367	0.9953	0.0047
2	EdgeHGN	0.9486	0.7825	0.9652	0.0011
	Linear SVM	0.9268	0.7218	0.9475	0.0198
	Poly-2 SVM	0.9430	0.7082	0.9528	0.0189
	SOM	0.9392	0.6852	0.9564	0.0148
3	EdgeHGN	0.9388	0.8692	0.9750	0.0038
	Linear SVM	0.8932	0.8485	0.9442	0.0116
	Poly-2 SVM	0.9072	0.7836	0.9644	0.0072
	SOM	0.9218	0.8206	0.9672	0.0064

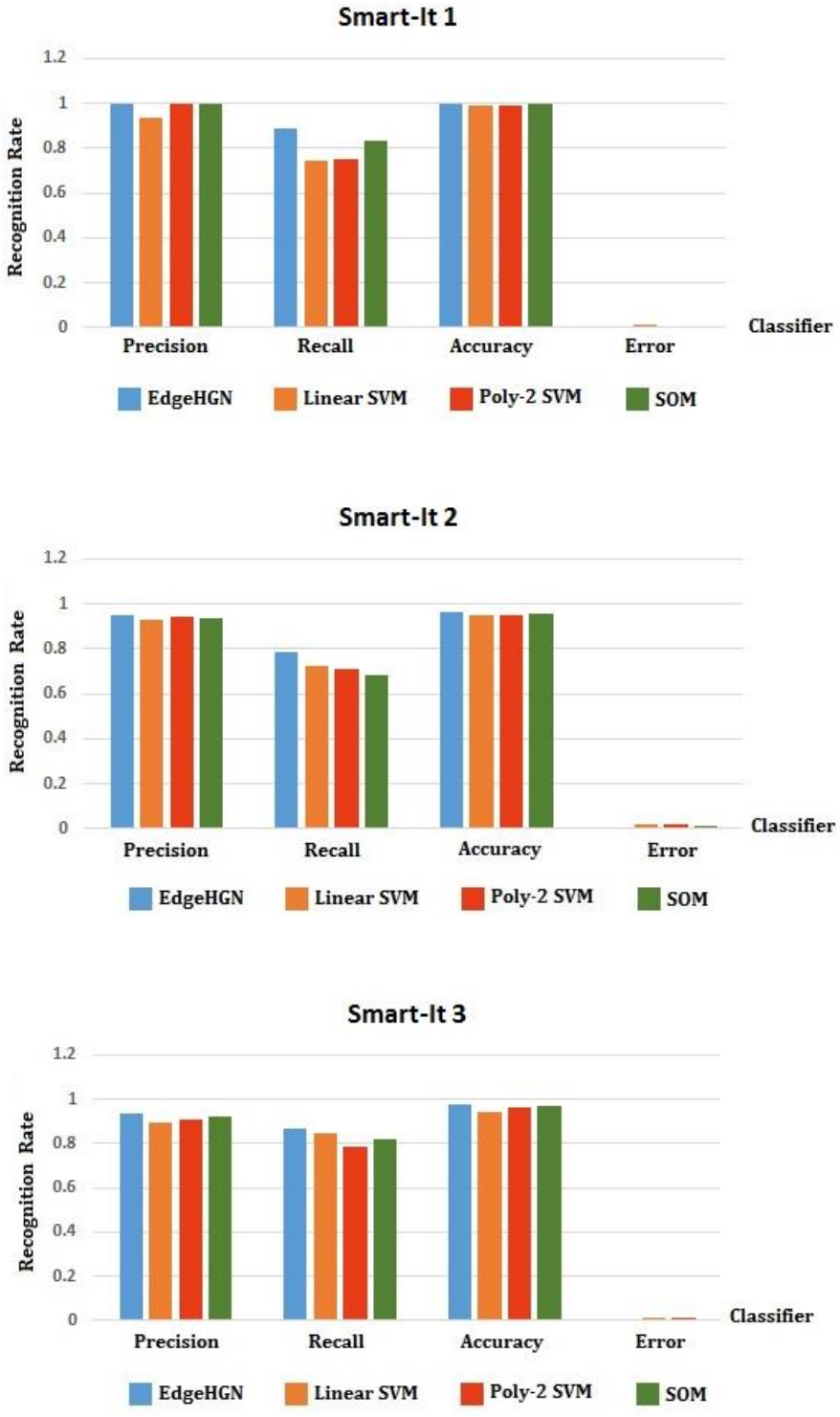

Figure 5.4: Comparative analysis on recognition parameters rates between EdgeHGN and other classifiers for event recognition using three sensory data obtained from Catterall et. al. (2003) (Smart-It 1, Smart-It 2, and Smart-It 3).

Overall, as shown in Figure 5.4, *EdgeHGN scheme offers better recognition accuracy than the other two algorithms*. In regards to the precision value, SOM and polynomial SVM also exhibit rather similar performances for Smart-It 1 and Smart-It 2. However, for Smart-It 3 SOM precision rate outperforms polynomial SVM. The polynomial SVM however produces slightly better results than the linear approach (for Smart-It 1, Smart-It 2 and Smart-It 3). This shows that SVM scheme performance is heavily reliant on the type of kernel that is being implemented (either linear or polynomial) as well as the nature of data being used. This data dependency problem restricts the flexibility of SVM approach for event classification in WSN. Although SOM shows better overall performance when compared with SVM, but its iterative algorithm is resource intensive and hence not practically feasible to be deployed in the resource-constrained wireless sensor networks. *EdgeHGN on the other hand, is a lightweight, single-cycle learning algorithm that is shown to offer high detection accuracy when used in conjunction with a simple threshold binary signature technique*. As part of this experiment, a performance analysis to measure the recognition time incurred for each sensor dataset is also conducted. The simulation was implemented using a fully-distributed MPI configuration using MPICH-2 package under GNU C program. Figure 5.5 shows the recall/store time for EdgeHGN recognition scheme dealing with three wireless sensor datasets (Smart-It 1, Smart It 2 & Smart-It 3).

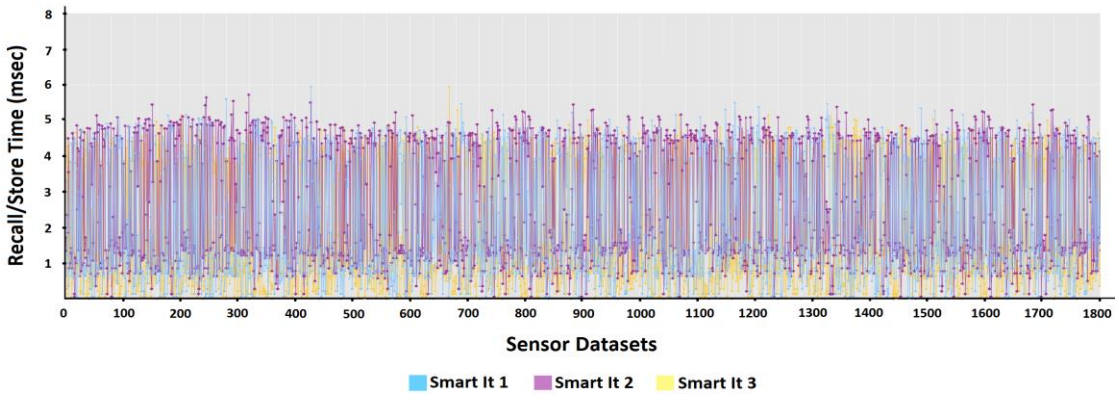

Figure 5.5: EdgeHGN Recognition time for 1800 sensor data (x-axis) taken from Smart-It 1, Smart It 2 and Smart It 3 datasets.

The recognition scheme only takes up to 6msec (average result of less than 3msec) for a sensor data to be recalled or memorised. As it is shown in the figure, *the average recall time for each sensor data remains highly consistent for the entire sensor data collection that makes this approach a suitable candidate for applications involving large amount of data.* The performance results also indicate a potential for real-time pattern recognition schemes within resource-constrained WSNs.

5.3.1 EdgeHGN-WSN Memory Utilization

Memory utilisation estimation for the EdgeHGN algorithm is calculated by performing an analysis of the bias array capacity for all of the GNs in the distributed architecture setup as well as the storage capacity of the SI module node. A detailed analysis of the bias array capacity for EdgeHGN algorithm was previously discussed in Chapter 3 (section 3.6.2.1). As concluded from that analysis and considering that EdgeHGN only uses memory to store newly discovered patterns rather than storing all input patterns, the storage/recall mechanism of EdgeHGN offers efficient memory utilisation. Figure 5.6 shows the estimated maximum memory usage for EdgeHGN when processing different pattern sizes and it compares that estimated value with the maximum memory size of a typical Berkeley Mica Mote senor node (128KB Flash and 4KB RAM).

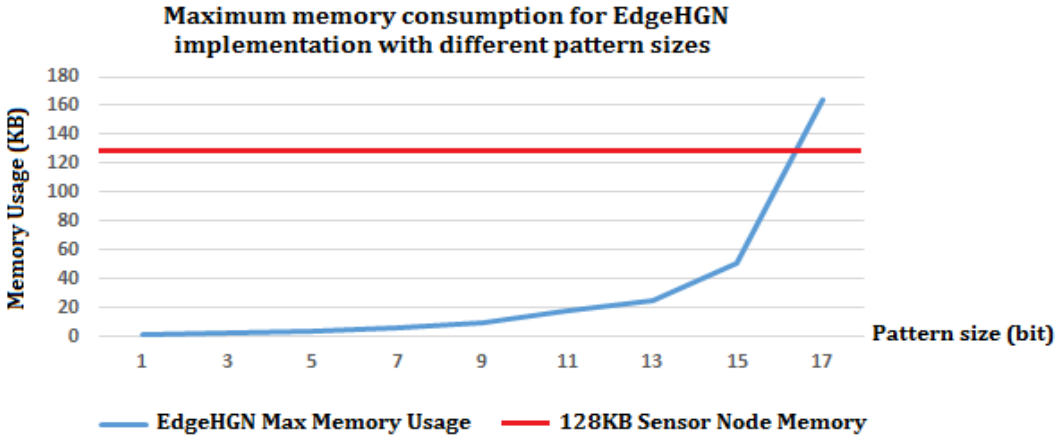

Figure 5.6: Maximum memory consumption for each EdgeHGN subnet for different pattern sizes. EdgeHGN uses minimum memory space with small pattern size

The EdgeHGN memory capacity estimation shown in Figure 5.6 is derived from equations (3.30) to (3.37) in Chapter 3. *As the size of input sub-pattern increases for more than 13 bits, the requirement for memory space increases significantly. It is worth noting that small sub-pattern sizes (sub-pattern sizes of 9 bits or smaller) only consume less than 10% of the total memory space available. Hence, EdgeHGN scheme for WSN is best suited for processing inputs with small sub-pattern sizes*.

5.4 Conclusions

The proposed use of EdgeHGN for event detection in WSN, outlines a new type of the WSN that detects macroscopic events by collating diverse sensor data, locally and in real-time, into meaningful patterns. As such, the research in this chapter performed a detailed study on EdgeHGN pattern recognition for event detection within WSN. The performance of EdgeHGN recognition approach in WSN was evaluated and benchmarked against SVM (linear and poly-2) and SOM in relation to its precision, recall, accuracy and error rates. From performance results, EdgeHGN shows high recognition accuracy with very low error value and very high recall rate which demonstrates the capabilities of EdgeHGN to implement a light-weight distributed event detection scheme within a resource-constrained network such as WSN. Furthermore, EdgeHGN approach offers very low memory consumption for event data storage mainly due to its simple bias array representation. This memory efficiency is best achieved when dealing with small input sub-pattern sizes, as EdgeHGN utilises only a small portion of the memory space in a typical physical sensor node in WSN network. The distributed nature of the scheme also lowers storage capacity requirements per node. In addition to this efficient memory usage, EdgeHGN reduces complexity of the WSN by eliminating the need for complex computations for event classification which increases its potential for wide-spread use. By adopting single-cycle learning and using adjacency comparison method, EdgeHGN offers a non-iterative and light-weight computational framework for event recognition and classification. EdgeHGN is a distributed computational model in nature and hence it can be readily deployed over a distributed network setup such as

WSN to provide an effective front-end recognition scheme for event detection within a WSN network.

Despite all the benefits that EdgeHGN offers, the scheme also suffers from certain limitations. EdgeHGN in its current form uses a simple data representation and for this matter the approach requires significant pre-processing to be conducted at the front-end system. This may not practically feasible for strictly resource-constrained sensor networks, where computing resources are very limited. Furthermore, EdgeHGN single-hop communication model may not suit WSN networks which cover a large geographical area or areas which are error prone due to potentially high packet loss rate during transmission phase. The current EdgeHGN implementation is also more focused on supervised classification while there is certainly a need for unsupervised classification for addressing requirements of rapid event detection. Addressing these limitations can all be the path for future research (i.e. providing unsupervised classification or utilising a multi-hop communication strategy).

This Page Intentionally Left Blank

Chapter 6

Case Study: Applying EdgeHGN based MapReduce Approach to Real World Big Data Processing Scenarios

Transforming big data into valuable information requires a fundamental rethink of how future data management models will need to be developed on the Internet. The efficiency of the cloud system in dealing with data-intensive applications through parallel processing essentially lies in how data are partitioned and processing is divided among nodes. As a result, data access schemes must be able to efficiently handle this partitioning automatically and support the collaboration of nodes in a reliable manner. This automatic data partitioning is what was previously known as domain decomposition problem, where data was divided on the basis of either geometrical or algorithmic considerations (Toselli & Widlund, 2005). Geometric considerations were efficient for spatial datasets e.g. finite element meshes (image-like) and algorithmic considerations were for data being partitioned in such a way that the computational load was equally divided in terms of equations to be solved (i.e. complexity). In cloud the consideration is data access rather than CPU parallelism. However, the fundamental data partitioning considerations remain the same.

In this regard, Google's MapReduce, was designed for large-scale data processing in a massively parallel manner that could solve issues involving the parallelisation of computational processes and data distribution across heterogeneous networks (Dean & Ghemawat, 2004). The MapReduce implementation also addresses load balancing, network performance and fault tolerance issues, and it has achieved greater scalability than parallel databases. However, this comes at a cost; time-consuming analysis and code customisations are required when dealing with complex data inter-dependencies. Moreover, existing large-scale data processing schemes such as MapReduce involve isolating basic operations within an application for data distribution and partitioning. This excludes their applicability to many applications with complex data dependency considerations. MapReduce models when used with complex data requirements generally entail additional difficult and error-prone application-level customisations. Adding higher and complex data representations within the model will vastly improve its usability and provide an important – pattern recognition based – data analysis option. To date, all implementations of MapReduce, including the Hadoop version, have interpreted data in a relational model. Utilising a single-cycle associative memory based method, which have so far not been investigated for MapReduce, will provide means to deliver efficient data processing. In light of the above issues, in chapter 4 we explored possibilities to evolve a novel processing scheme that could efficiently partition and distribute data for clouds. In this regards, loosely coupled associative techniques could be pivotal in effectively partitioning and distributing data in the clouds. Thus, the aim in this research was to develop a distributed data access scheme that could enable data access to be conducted effectively by means of the distributed pattern recognition (DPR) approach.

To achieve this, a distributed data access scheme referred to as EdgeHGN, is first developed to circumvent the partitioning issue experienced within referential data access mechanisms. In this model, data records are treated as patterns and as a result, data storage and retrieval are performed using a distributed pattern recognition approach. Furthermore, to reconcile MapReduce with associated memory concepts, in particular for adaptive and fast data access, an associative-memory-based MapReduce

is introduced to elevate the MapReduce key-value scheme to a higher level of functionality by replacing the purely quantitative key-value pairs with scalable associative-memory-based data structures that will improve parallel processing of data with complex relations. By having an associative key-value model, we can deal with data in any form and in any representation simply by using a pattern-matching model that treats data records as patterns and provides a distributed data access scheme that enables data storage and retrieval by association, thereby circumventing the scaling issue experienced within referential data access mechanisms.

6.1 EdgeHGN based MapReduce – High Level Framework

As discussed in detail in chapter 3, EdgeHGN allows the recognition process to be conducted in a smaller sub-pattern domain, hence minimising the number of processing nodes, which in turn reduces the complexity of pattern analysis. In addition, the recognition process performed using the EdgeHGN algorithms is unique in a way that each subnet is only responsible for memorising a portion of the pattern (rather than the entire pattern). A collection of these subnets is able to form a distributed memory structure for the entire pattern. This feature enables recognition to be performed in parallel and independently. The decoupled nature of the sub-domains is the key feature that brings scalability to our data management approach for the cloud. Moreover, EdgeHGN provides a capability for a recognition process to be deployed as a composition of sub-processes executed in parallel across a distributed network. Sub-processes execute mutually independently which makes this approach less cohesive compared to other pattern recognition schemes.

In chapter 4, we demonstrated that the effectiveness of MapReduce parallelism as a scalable scheme for data processing in the cloud can be improved by transforming the data-processing operation into a single-cycle distributed pattern matching approach in which distributed computations are performed in-network, thereby enabling data storage and retrieval by association rather than deploying a referential data access mechanism. In this context, processing the database and handling the dynamic load could be performed using a distributed pattern recognition approach. In

EdgeHGN_MR approach, the principle of associative-memory-based learning is implemented through the use of connected layers in a hierarchical fashion; with local feature learning happening at the lowest layer while features are combined to form higher representations at upper layers. This approach envisages data retrieval being implemented as a distributed pattern recognition process that is implemented through the integration of associative memory based computational networks. In a high-level EdgeHGN_MR design framework, the map function takes EdgeHGN subnets as the key and the object itself as the value, performs sub-pattern matching, calculates the bias index and emits a set of intermediate key-value pairs as output. Intermediate keys are EdgeHGN subnets and intermediate values are GN bias arrays, holding store or recall decisions for each subnet. It is worth noting that all map functions can be run and implemented in parallel. The class reducer then works on EdgeHGN subnets as keys and intermediate GN bias associative arrays as values, calculates the final decision and then emits the final store or recall decision. Algorithm 6.1 shows a high-level framework for our proposed EdgeHGN_MR scheme. A practical way to test the usefulness of our approach is to apply the EdgeHGN_MR processing scheme to real-world big data processing problems. For this reason, in the remaining sections of this chapter, we present the results of a 6-month AMSI internship project conducted at a major pharmaceutical company to showcase a study on the adoption of EdgeHGN_MR distributed data processing scheme for analysing large-scale environmental monitoring data and IT service management data.

The remaining part of this chapter has been structured as follows. Section 6.2 discusses the overall setup and design model for our case study. Section 6.3 presents the results of processing merged Solarwinds datasets using MapReduce scheme. Section 6.4 discusses the results of analysing ITSM data using MapReduce approach. Section 6.5 shows the results of pattern matching between ITSM and Solarwinds datasets utilising EdgeHGN_MR algorithm and finally section 6.6 concludes this chapter by offering some comparative performance benchmarking results between MR and EdgeHGN_MR implementations.

Algorithm 6.1: EdgeHGN based MapReduce – High level framework

Class Mapper

method Map(EdgeHGNSubnet $\mathcal{S}$, Obj $\mathcal{O}$)

BAA ← new GN Bias Associative Array

for all term $\mathfrak{T}$ ∈ EdgeHGNSubnet $\mathcal{S}$ **do**

Calculate Adjacency Comparison Function (algorithm 3.3)

Calculate Bias Index (algorithm 3.4)

Update **BAA**

Emit(EdgeHGNSubnet $\mathcal{S}$, BiasAssociativeArray **BAA**)

Class Reducer

method Reduce(EdgeHGNSubnet $\mathcal{S}$, BiasAssociativeArray **BAA**)

for all b ∈ BiasAssociativeArray [b_1, b_2, . . .] **do**

Calculate SI Module function (algorithm 3.1)

Calculate Voting function (algorithm 3.2)

Calculate result ← Store/Recall

Emit (Obj $\mathcal{O}$, Boolean **result**)

6.2 Case Study: Solarwinds and ITSM Big Data Processing using MapReduce and EdgeHGN based MapReduce

Data alignment in a high-tech complex environment is critical for the productivity and profitability. The case study project work was conducted at a high-tech pharmaceutical company in Melbourne with data being produced at each step of the process, but in disparate systems where data in scope is of excessive size and joining the dots across all the data sources is critical for success. The company senior leadership was striving to improve productivity by readily converting the data into information. The success of the project had to be measured on the actionable insights. Participating in this real world large data analysis could also help management team

to learn from their data and make informed decisions to deliver tangible results. In summary, the purpose of the research project was to develop proof points allowing to:

- Perform some sort of data processing and data alignment to uncover hidden patterns, unknown correlations and other usefull information that could be used to make better decisions, improve monitoring and increase efficiency.
- Trend environmental monitoring data (Solarwinds) against IT service management data (ITSM) to find correlations between the two to aid engineering team take a pro-active approach and minimise production failures.

Solarwinds offers infrastructure management software to monitor various components like network performance, application performance and database performance, storage and disk performance. IT service management data or ITSM data on the other hand mostly captures requests, problems and incidents reported mainly by end users or clients. It is mainly a manual data capture process but in some cases automated tickets can be raised as the result of Solarwinds alerts. To perform data analysis, both MapReduce and EdgeHGN based MapReduce implementations are setup within a Hadoop based framework to evaluate their respective performance when dealing with ITSM and Solarwinds dataset examples. Table 6.1 lists characteristics of the given database snapshots for ITSM and Solarwinds for this data processing exercise.

Table 6.1: ITSM and Solarwinds data snapshots for data processing exercise

Data Sources	Solarwinds: Production environment monitoring data
	ITSM: Production incident/request data
Data Volume	ITSM: 116 MB Solarwinds: 382 MB
History	35 days (08/02/2015 to 11/03/2015)

The experimental Hadoop cluster was configured with four *DataNodes* and one *NameNode*. The *NameNode* machine acts as both *JobTracker* and *NameNode* while each of the four *DataNodes* act as both *TaskTracker* and *DataNode*. The Hadoop cluster configuration details are listed in Table 6.2.

Table 6.2: Hadoop 4-node cluster details for implementing MR and EdgeHGN_MR

NameNode apaubmwapp12 172.21.92.38	4 Virtual CPU (VP) Memory: 8GB, SSD: 1.5TB OS: SUSE Linux 11
4 DataNodes apaubmwapp13/14/15/16 172.21.92.39/172.21.92.40/172.21.92.41/172.21.92.42	4 Virtual CPU (VP) Memory: 8GB, SSD: 1.5TB OS: SUSE Linux 11
Network bandwidth	1Gbps
Hadoop version	2.5.2, 64 bits
Java Version	OpenJDK 1.6
JVM Heap Size	16GB

6.2.1 Solarwinds and ITSM Data Correlation Design Model

The end goal of this project was to perform data analysis on ITSM and Solarwinds datasets in isolation and in conjunction to find correlations and common patterns. As Solarwinds monitoring data was captured and stored across a few separate database tables, merging and linking relevant data from different sources could provide us with meaningful insights. Moreover, searching for common patterns and finding correlations between ITSM and Solarwinds data could provide monitoring team with meaningful insights that were not achievable before due to their lack of proper big data processing tools in place. To perform this data correlation exercise between ITSM and Solarwinds data we could search for some shared keys or similar patterns. To perform this task, one of the possibilities was to do some sort of text analysis on free-text entry fields in ITSM data logs. In fact, for ITSM data we had a few free-text entry fields like subject and incident summary that users could use to put their

comments in. By writing some simple text-mining scripts we could extract some unique identifiers like object names to correlate ITSM and Solarwinds data using them utilizing MapReduce (MR) and EdgeHGN based MapReduce (EdgeHGN_MR) schemes. Figure 6.1 illustrates the SPSS modelling process of extracting such unique identifiers (patterns) from ITSM data and searching for those patterns within Solarwinds data using MR and EdgeHGN_MR approaches. It should be noted that Solarwinds alert log database identifies each alert with an *AlertDefID* tag, but does not explicitly identify the Object Type/Object Name that each alert belongs to. To solve this issue as shown in Figure 6.1, a reference table was created by extracting the AlertDefID from the AlertLog, AlertStatus and AlertDefinition tables to have a complete view of all Solarwinds alert data entries.

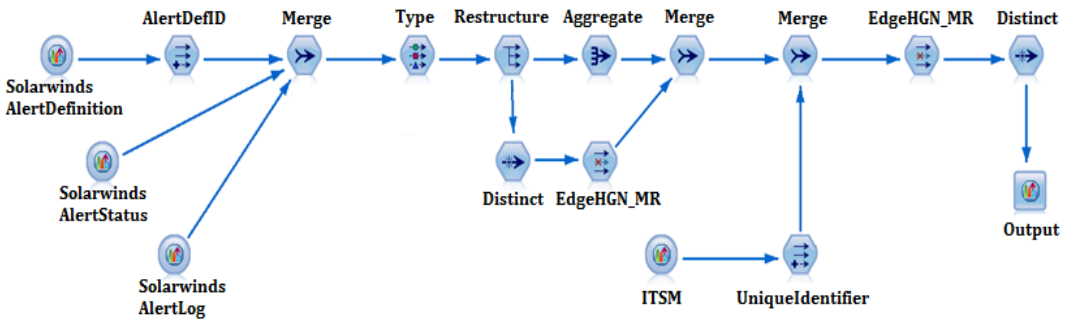

Figure 6.1: SPSS modelling process of linking ITSM and Solarwinds using EdgeHGN_MR scheme

In the next step, distinct identifiers are extracted for the Solarwinds object name data fields and EdgeHGN network is trained using those identifiers. Unique identifiers are also extracted from ITSM free-text entry fields (subject and incident summary) where these identifiers are used by EdgeHGN_MR approach to search through Solarwinds datasets to find possible correlations between ITSM and Solarwinds data. Figure 6.2 illustrates an architectural overview of the project setup. ITSM and Solarwinds data are extracted from relevant databases and fed into Hadoop distributed file system (HDFS) using scoop database connectivity technology. Scoop is simply a tool designed for efficiently transferring bulk data between Apache

Hadoop and structured data stores such as relational databases to enable MapReduce functions. To build and develop MapReduce functions we also use Apache Hive which gives us required tools for doing query and analysis on top of HDFS. Apache Hive is the preferred choice here because it provides HiveQL, a SQL-like query language that we can utilise to convert queries to Map and Reduce.

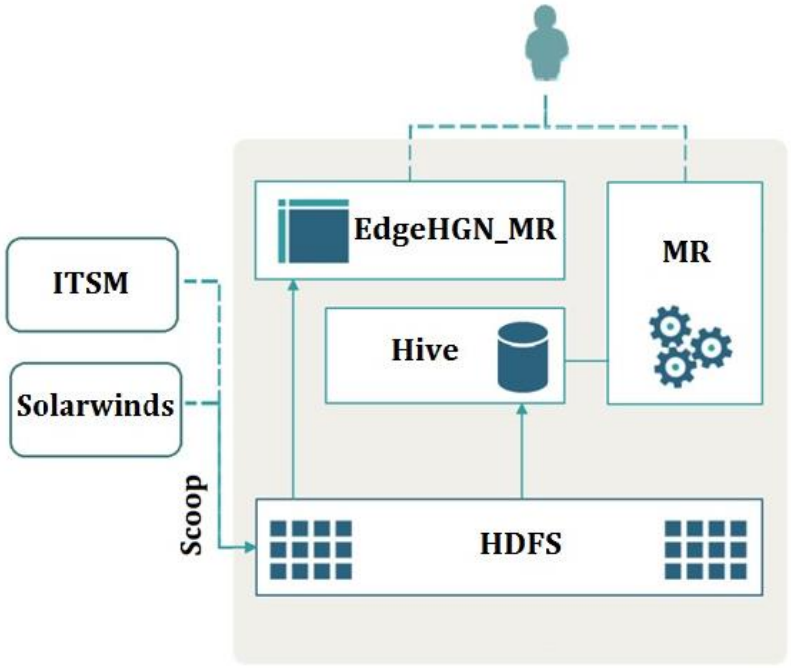

Figure 6.2: Architectural overview of ITSM and Solarwinds data correlation project

6.3 ITSM & Solarwinds Data Correlation Using EdgeHGN_MR

In this section, the data analysis results of pattern matching between Solarwinds and ITSM datasets using both MR and EdgeHGN_MR schemes are presented. For this particular exercise, we look at Solarwinds alerts from 8th of February to 11th of March 2015 which are just a snapshot of Solarwinds data stored in the database. We have about 20 thousand alerts generated for that 35-day period. We also look at ITSM requests/incidents for the same period of time to perform data correlation task

between the two. We have about 5110 incident/request tickets raised for that 35-day period. It should be noted that each ITSM ticket has 19 entry fields *(IncidentNumber, CreatedDateTime, CreatedBy, Department, IsVIP, LastModBy, LastModDateTime, OwnerDisplay, OwnerTeam, Priority, Category, Service, ProfileFullName, Source, Status, Subject, IncidentSite, ResolvedBy, and resolution*) from which "*subject*" and "*resolution*" fields are free-text entry forms that users of the system can utilise to put their comments in. In summary, the following steps are followed to perform ITSM and Solarwinds data correlation:

i. Cleansing and pre-processing ITSM and Solarwinds data to remove outliers (observation points that are distant from other observations).
ii. ITSM and Solarwinds data ingestion and integration into Hadoop platform (HDFS) using scoop database connectivity technology.
iii. Creating a merged dataset of Solarwinds alerts from *AlertLog*, *AlertStatus* and *AlertDefinition* datasets using *AlertDefID* tag value.
iv. Developing simple text mining extractor scripts to find unique identifiers in the free-text entry fields of ITSM data (subject and incident summary).
v. Initializing eight Mapper functions where each Mapper constructs an EdgeHGN subnet.
vi. Training EdgeHGN network with those unique identifiers extracted from ITSM datasets.
vii. Input data (merged Solarwinds dataset) is split among eight data chunks so that later they can be processed by Mapper functions in parallel where each Mapper constructs the same EdgeHGN classifier using similar set of training data.
viii. Each Mapper inputs a subset of Solarwinds testing instances. When the network starts performing recognition test, each Mapper starts classifying only a subset of the entire testing dataset. This approach improves efficiency through parallelism.
ix. Upon completion of all Mapper tasks, the Reducer starts processing and merging all the outputs of Mappers using the same key (calculating the output for each unique identifier) and writing the result back into HDFS.

Algorithm (6.2) depicts the pseudo-code for EdgeHGN_MR scheme and Figure (6.3) shows the architecture of EdgeHGN_MR approach used in this experiment.

Algorithm 6.2: EdgeHGN_MR scheme for implementing pattern matching between ITSM & Solarwinds

---

Input: Solarwinds Dataset ($\mathbb{T}$) **Output:** Pattern matching result (AA)

(1) **eight** Mappers and **one** Reducer, each Mapper constructs an EdgeHGN subnet

(2) Train EdgeHGN network with unique identifiers from ITSM data

(3) Divide $\mathbb{T}$ into $\{t_1, t_2, t_3 \ldots t_8\}$, $\cup_{i=1}^{8} t_i = \mathbb{T}$

(4) Each Mapper builds an EdgeHGN subnet and inputs t_i where $t_i \in \mathbb{T}$

BAA ← new Boolean Associative Array

For all term $t_i \in \mathbb{T}$ do

Calculate Adjacency Comparison Function (algorithm 3.3)

Calculate Bias Index (algorithm 3.4) & Update BAA_i

(5) Mapper outputs (, BAA_i)

(6) Reducer collects and merges all (, BAA_i)

For all term BAA $(i = 1, \ldots 8)$ do

Calculate SI Module function (algorithm 3.1)

Calculate Voting function (algorithm 3.2)

Calculate AA ← Store/Recall

(7) Repeat (3), (4) and (5) until $\mathbb{T}$ is traversed and all testing data are processed

(8) Reducer outputs (AA) and writes it back into HDFS

By implementing this approach, we could identify 206 tickets in ITSM as the result of Solarwinds alerts (See Figure 6.4). However, due to the poor data quality for ITSM, there is a good chance that we have more tickets in ITSM system due to Solarwinds alerts.

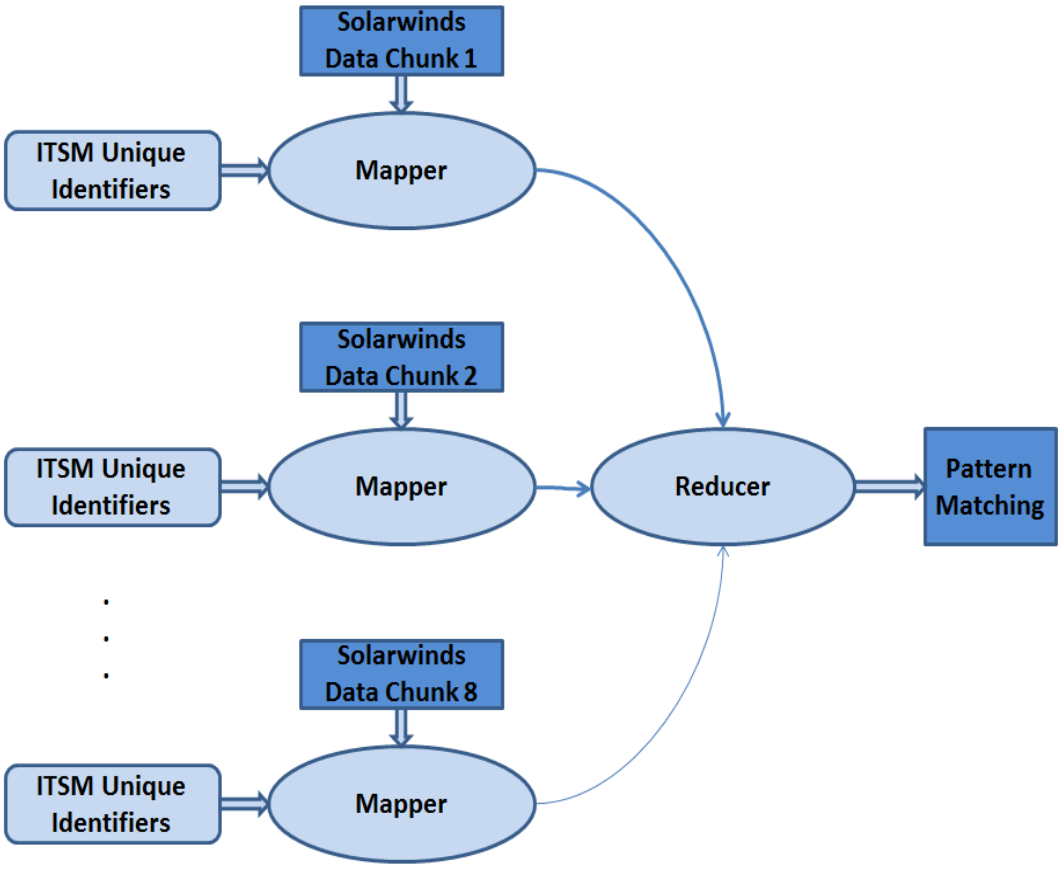

Figure 6.3: EdgeHGN_MR architecture for pattern matching between ITSM and Solarwinds datasets

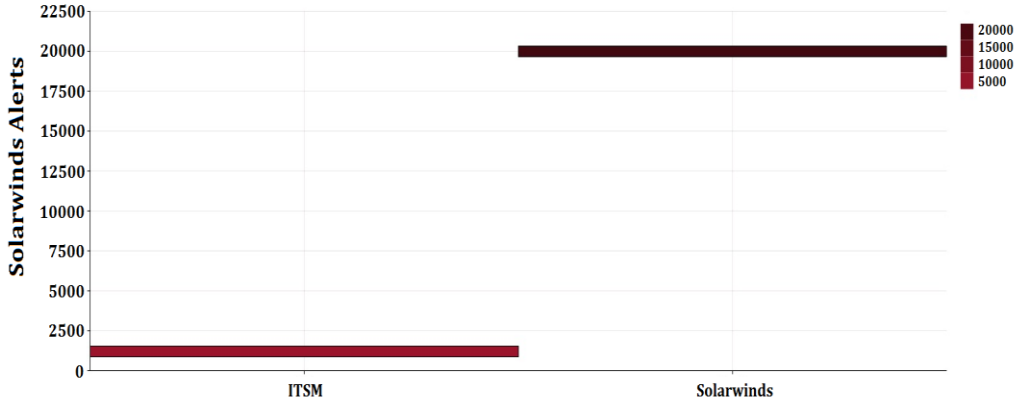

Figure 6.4: ITSM tickets raised due to Solarwinds alerts

Figure 6.5 shows that out of 206 ITSM tickets raised due to Solarwinds alerts, 53% have their service listed as data management.

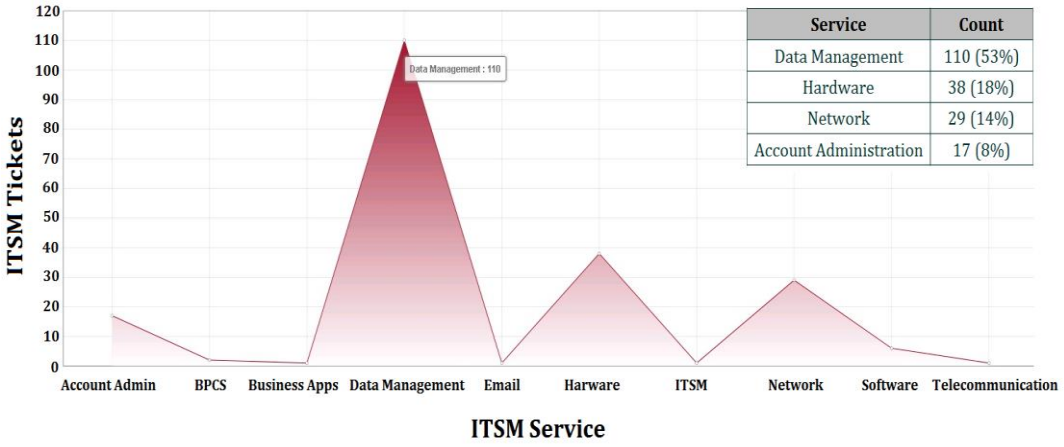

Figure 6.5: Service field for ITSM tickets raised due to Solarwinds alerts

Figure 6.6 illustrates the main causes of alert. As shown in this figure, almost 43% of alerts are due to high disk I/O latency. The second most common cause of alerts is Node down alert. It can be safely assumed that some of these node down alerts are happening during a maintenance window where for instance a server goes down due to a scheduled change but for some reason Solarwinds alerts are not properly blacked out, hence there is some opportunity to reduce noise for improvement here.

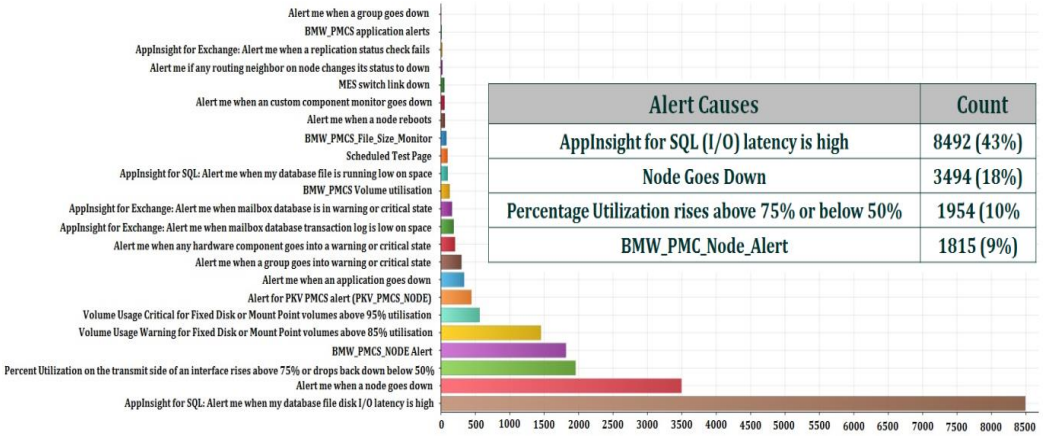

Figure 6.6: Main causes of Solarwinds alerts

Figure 6.7 shows average distribution of Solarwinds alerts during day. This graph is interesting as it shows most number of alerts are generated around 1pm (2057 alerts over that 35-day period) and almost 10% around mid-day. The other interesting observation here is that 7% of alerts are generated around 11pm which is normally due to weekly bounces or scheduled changes.

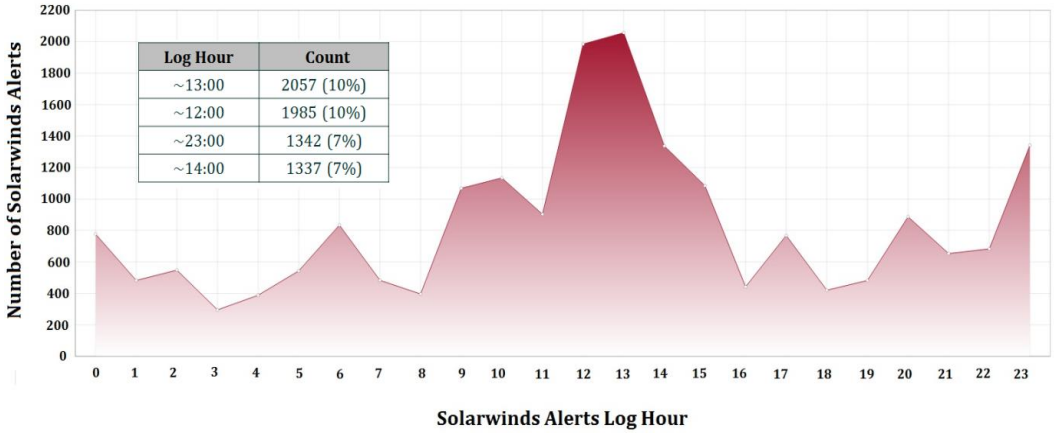

Figure 6.7: Average distribution of Solarwinds alerts during day

6.4 Data Correlation Results

Table 6.3 illustrates the processing time of performing data correlation between ITSM and Solarwinds datasets using both MR and EdgeHGN_MR schemes. The version of 4-node Hadoop cluster implementation was 2.5.2 at the time of conducting this test (see Table 6.2).

Table 6.3. ITSM & Solarwinds data processing time using MR & EdgeHGN_MR

Solarwinds (MB)	MR	EdgeHGN_MR
50MB	18 secs	16 secs
100MB	31 secs	26 secs
200MB	55 secs	48 secs
300MB	81 sec	72 secs
382MB	103 secs	92 secs

As shown in Figure 6.8, *EdgeHGN_MR outperforms MR for both small and large data volumes*. In fact, processing small data with MR operation is undesirable because the time it takes to collect distributed data during a Reduce operation within the same processing node outweighs the advantages of distributing data between Map functions. On the other hand, *EdgeHGN_MR works well in dealing with both small to large size data counts due to its parallel single-cycle learning mechanism where the size of input data has minimal effect on the time of its single-cycle in-network processing*.

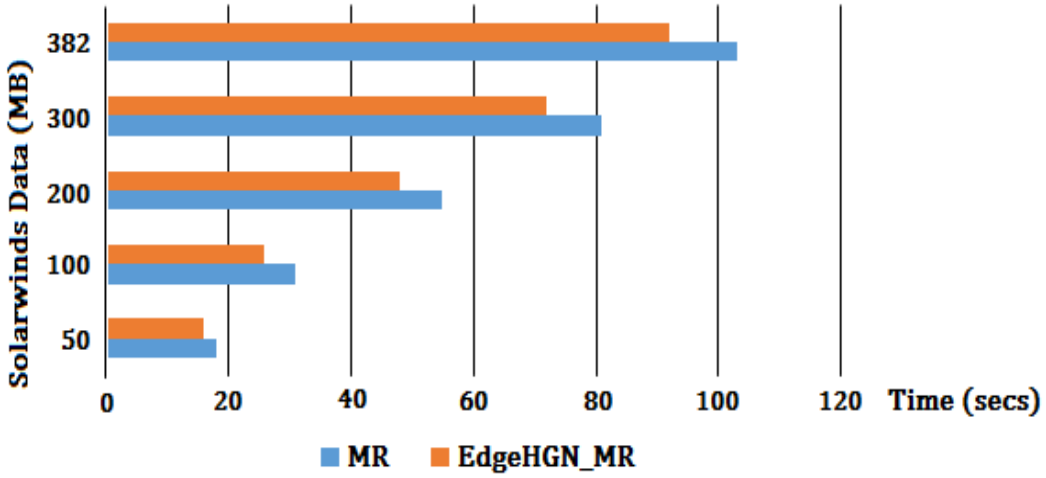

Figure 6.8: Processing time of performing data correlation between ITSM and Solarwinds datasets using both MR and EdgeHGN_MR schemes

6.5 Conclusion

In this chapter, the results of a 6-month big data AMSI internship project at a major pharmaceutical company was presented as a case study to evaluate performance of EdgeHGN_MR when applied to real-world big data processing scenarios. Both MR and EdgeHGN_MR schemes were utilised to perform data correlation between two sets of environmental monitoring data and IT service management data. Comparative results demonstrate the improved response time when using EdgeHGN_MR approach. Nevertheless, this comparison was not intended for a replacement of MapReduce with EdgeHGN_MR, but more of comparative indication of

EdgeHGN_MR ability for large-scale data processing. EdeHGN_MR online-learning AM scheme is conceived on the principle that '*moving computation is much cheaper than moving data*'. Hence, it will provide methods for automatic aggregation and partitioning of associated data and redefines the reduction phase by forming a hierarchical adjacency based computing model to produce the final result. Furthermore, when using EdgeHGN_MR programmers can work at a higher level of abstraction without having to know the structural details of every data item. This is due to the fact that EdgeHGN model uses a universal structure for all data types. In fact, information about the logical structure of the data – metadata – and the rules that govern it can be stored alongside the data. The approach is not only scalable and supports single-cycle learning, but it is also generic where large and complex data sets from a variety of sources and representing diverse pattern recognition requirements can be analysed in real-time. The scheme thus demonstrates that large-scale pattern recognition is possible through the distributed processing.

Chapter 7

Conclusion

Supporting data intensive applications is an essential requirement for the clouds. However, dynamic and distributed nature of cloud computing environments makes data management processes very complicated, especially in the case of real-time data processing/database updating. With emerging interest to leverage massive amounts of data available in open sources such as the Web for solving long standing information retrieval problems, the question as how to effectively incorporate and efficiently exploit immense data sets is an open one. To cope with today's intensive data workloads, initial proposed schemes include distributed databases for update intensive application workloads and parallel database systems for descriptive and deep analytics. Although distributed data management has been the vision of the database research community for a long period of time, but much of this research has been focussed on designing scalable schemes for intensive workloads in traditional large-scale data processing settings, and lesser impetus on re-designing the processing architecture to keep up with big data. While the opportunities for parallelisation and distribution of data in clouds make storage and retrieval processes very complex, especially in facing with real-time data processing, the challenge of processing voluminous data sets in a scalable and cost-efficient manner has rendered traditional

database solutions prohibitively expensive. Due to changes in the data access patterns of applications and necessity to use thousands of compute nodes, major cloud computing companies have started to integrate frameworks for parallel data processing in their product portfolio; making it easier for customers to access these services and to deploy their applications. Thus efficiencies through widespread use of multi-core CPUs, cost reduction for commodity hardware, enhanced performance, and higher reliability in use are derived from an architectural paradigm which favours a massively distributed data processing framework running on a large number of inexpensive compute nodes. Large data operations such as processing crawled documents or regenerating a web index are split into several independent subtasks, distributed among the available nodes, and computed in parallel within the network.

To simplify the development of distributed applications on top of such highly distributed architectures, customised data processing frameworks such as MapReduce are developed and deployed on large clusters of shared-nothing compute nodes rather than relying on traditional database management systems (DBMSs). Although these schemes differ in structure, their design concepts share similar objectives, namely hiding complexity of parallel programming, fault tolerance, and execution optimisation issues from the developer. In fact, developers can typically proceed with writing sequential programs and it is the processing framework which takes care of distributing the program among the available compute nodes and executing each instance of the program on the appropriate fragment of data set. Nevertheless, MapReduce have achieved greater scalability than parallel databases at the cost of avoiding complex transaction support but these still require customisation of the analysis code. It is also worth noting that in MapReduce computational model, not only the maximum parallelism of the parallel map phase is limited by the number of input pairs, but also the parallelism in the reduction phase is also limited by the number of different output keys of the map phase, which in turn highly depends on the implemented algorithm and the input data. These limitations require that MapReduce model is significantly enhanced in a way that preserves the strength of the model and eliminates these constraints.

7.1 Research Summary

Our proposed scheme in this research does so in a well-integrated manner where there is no outward change in way of its deployment and use. The research conducted in this thesis has focused on evolving a new type of data processing approach that will efficiently partition and distribute data for clouds. For this matter, loosely-coupled associative techniques, not considered so far, can be the key to effectively partitioning and distributing data in the clouds. Unlike the existing relational schemes, associative models of data can analyse data in similar ways to which our brain links information. Such interactions when implemented in voluminous data clouds can assist in finding overarching relations in large and complex data sets. In this context our proposal in this research investigated an associative memory model for use with the MapReduce based search schemes for uncovering new patterns. In order to achieve this, an initial step taken was to develop a distributed data access scheme that enables data storage and retrieval by association, and thereby circumvents the partitioning issue experienced within referential data access mechanisms. In our proposed scheme, data records are treated as patterns. As a result, data storage and retrieval can be performed using a distributed pattern recognition approach that is implemented through the integration of loosely-coupled computational networks, followed by a divide-and-distribute approach that allows distribution of these networks within the cloud dynamically. Our proposed approach is based on a special type of Associative Memory (AM) model, which is readily implemented within distributed architectures. Hierarchical structures in associative memory models are of interest as these have been shown to improve the rate of recall in pattern recognition applications. As we know, existing data access mechanisms for cloud computing such as MapReduce has proven the viability of parallel access approach in cloud infrastructure. Thus, our aim in this thesis was to apply an access scheme that enables data retrieval across multiple records and data segments within a single-cycle, utilising a parallel approach. It should be noted that a framework that meets this goal will provide vast improvements to MR model, further reduced costs, improved functionality, and will extend the application space.

To achieve our research objectives in this thesis, the below steps are followed:

- Redesigning data management architecture to treat data records as patterns, and thus, enabling data storage and retrieval by association over and above the existing simple data referential mechanisms.
- Processing the database and handling the dynamic load using a distributed pattern recognition approach that is implemented through the integration of loosely-coupled computational networks, followed by a divide-and-distribute approach that allows distribution of these networks within the cloud dynamically.
- Developing a scalable MapReduce framework that allows complex data representations to be used as keys for Map and Reduce operations; allowing content-association based data retrieval and storage within cloud.
- Validation of results and finding asymptotical limits of the technique through rigorous testing.

A summary of the contents of this thesis is as follows:

i. In chapters 1 and 2, a comprehensive review has been conducted on current implementations of scalable pattern recognition. The study determined three fundamental approaches towards addressing the scalability concerns within PR schemes, namely data, learning, and distributed approaches. As described in section 1.4, the data approach performs reduction or modification of data, while the learning approach aims to reduce the complexity of memorisation and recognition phases. Nevertheless, both the data and learning approaches do not fully meet the scalability requirements, mainly due to loss of data integrity, low recognition accuracy and high computational costs. On the other hand, the distributed approach, due to its ability to distribute data and processes within computational networks, has been shown to exhibit high scalability to scale up with today's outgrowth of data, that involves large and complex datasets. Nevertheless, some of the existing models are extremely complex and highly cumbersome to parallelise.

ii. Furthermore, in chapter 2, a comprehensive study on the current data-parallel frameworks for cloud data processing has been presented and different kinds of approaches to large-scale data processing have been explored. The pros and cons of each approach are also examined in relation to scalability and adaptability requirements of big data processing. This chapter has also presented a detailed analysis on how neural network approaches can open a new pathway for accessing data in highly distributed environments by discussing some major schemes presented in the literature. The investigation carried in chapter 2 revealed the fact that existing neural network techniques in their current forms are far from providing a suitable scalable framework for large scale recognition purposes.

iii. In chapter 3, we established the thesis position by proposing an associative memory based scheme, referred to as edge detecting hierarchical graph neuron (EdgeHGN) that utilises single-cycle learning and implements a bottom-up approach. Our proposed distributed pattern recognition approach remains scalable for any given size or dimensions of data, if sufficient computational resources are available. It is not only applicable to numerical and textual data processing problems but also it is effectively capable of processing complex patterns, such as high-dimensional images. EdgeHGN allows the recognition process to be conducted in a smaller sub-pattern domain, hence minimising the number of processing nodes, which in turn reduces the complexity of pattern analysis. In addition, the recognition process performed using the EdgeHGN algorithm is unique in a way that each subnet is only responsible for memorising a portion of the pattern (rather than the entire pattern). A collection of these subnets is able to form a distributed memory structure for the entire pattern. This feature enables recognition to be performed in parallel and independently. The decoupled nature of the sub-domains is the key feature that brings dynamic scalability to our data processing approach for the cloud. Moreover, EdgeHGN provides a capability for a recognition process to be deployed as a composition of sub-processes executed in parallel across a distributed network. Sub-processes execute mutually

independently. This approach is less cohesive compared to any other pattern recognition scheme. Furthermore, the conducted complexity analysis of the proposed scheme indicates that the approach is highly scalable and incurs low computational costs as part of its recognition procedure.

iv. In chapter 4, we formulated a distributed data management approach, referred to as EdgeHGN based MapReduce, that enables seamless data access and distribution using single-cycle learning associative memory-based algorithms. This is achieved by designing a scalable MapReduce framework that allows complex data representations to be used as keys for Map and Reduce operations; allowing content-association based data retrieval and storage within cloud. EdgeHGN_MR elevates the MapReduce key-value scheme to a higher level of functionality by replacing the purely quantitative key-value pairs with more complex data structures that empowers the parallel processing of data with complex associations (or dependencies). By having an associative key-value framework, we can deal with data in any form and in any representation simply by using a pattern matching model (including fuzziness) that treats data records as patterns and provides a distributed data access scheme that enables balanced data storage and retrieval by association. Our proposed scheme preserves the strength of the MapReduce model and eliminates/alleviates most of its constraints in a well-integrated manner where there is no outward change to the way in which MapReduce models are deployed and used. The experimental results conducted in chapter 4 demonstrate that EdgeHGN_MR provides comparable performance benchmarks (high accuracy rate with low response time) when tested against well-known Pregel-like graph processing systems such as Giraph, GPS, Mizan and GraphLab.

v. In chapter 5, a study on capability of the proposed EdgeHGN scheme to perform pattern recognition in fine-grained wireless sensor networks (WSNs) was established through the use of simple recognition procedure with low processing and memory requirements. From performance results, EdgeHGN shows high

recognition accuracy with very low error value and very high recall rate which demonstrates the capabilities of EdgeHGN to implement a lightweight distributed event detection scheme within a resource-constrained network such as WSN. By adopting single-cycle learning and using adjacency comparison method, EdgeHGN offers a non-iterative and scalable computational framework to provide an effective front-end recognition scheme for event detection within a WSN network.

vi. In chapters 6, the results of a 6-month big data processing AMSI internship project at a major pharmaceutical company are provided as a proof of concept that our approach will indeed work when applied to real-world large scale data processing problems. For this exercise, EdgeHGN_MR scheme was implemented to perform data correlation between IT service management data and environmental monitoring data. The conducted experimental results show that EdgeHGN_MR approach incurs lower processing time when compared with standard MapReduce implementation. This can firmly establish the credentials of our technique as an innovative and viable approach for addressing real-world big data processing problems.

Our proposed data processing scheme in this thesis is primarily focused for use within the MapReduce framework and is fundamentally different from all published approaches in data management. It has the potential of wider applicability, provided we can benchmark its characteristics against some of the discipline specific techniques, e.g. parallel finite elements and more broadly, the Parallel Dwarfs Project. The large heterogeneous data sets already gathered from various experiments and case studies provide an excellent resource to compare and contrast the one-shot learning, scalability, and accuracy of our approach with a number of well-established data management techniques. In this regard, performance results already derived from various experiments are very promising.

7.2 Research Contributions

In this thesis, four key contributions have been made that are recapitulated below, in the sequence that they appear in the thesis.

i. A distributed data access scheme, referred to as EdgeHGN, is proposed that enables data storage and retrieval by association where data records are treated as patterns; hence, finding overarching relationships among distributed datasets becomes easier for a variety of pattern recognition and data-mining applications. EdgeHGN does not require definition of rules or manual interventions by the operator for setting of thresholds to achieve the desired results, nor does it require heuristics entailing iterative operations for memorisation and recall of patterns. In addition, our approach allows induction of new patterns in a fixed number of steps. Whilst doing so it exhibits a high level of scalability i.e. the performance and accuracy do not degrade as the number of stored pattern increases over time. Its pattern recognition capability remains comparable with contemporary approaches. Furthermore, all computations are completed within the pre-defined number of steps and as such the approach implements one shot, i.e. single-cycle or single-pass, learning. The proposed scheme will be suitable for the operational requirements of clouds and will enable relevant data to be readily available for large-scale computations.

   EdgeHGN imposes low computational complexity. Comparative analysis of recognition accuracy and complexity between EdgeHGN and other recognisers/classifiers such as Hopfield Network and Kohonen SOM (Section 3.6) demonstrate EdgeHGN's low complexity in performing recognition processes. In terms of Big-O complexity estimation, EdgeHGN only imposes linear complexity as low as $\mathbf{O}(\boldsymbol{n})$ in its recognition process, where $\boldsymbol{n}$ represents a single executable instruction within the procedure (section 3.6). The results obtained from a number of experiments have demonstrated that EdgeHGN recall accuracy is considerably high for distorted pattern recognition. In evaluating the accuracy of EdgeHGN recogniser, it was determined that the proposed approach is capable of producing perfect recall for up to 20% distortion on binary character

images (Section 3.7.1). At this scale of distortion, even human eye can barely associate the original pattern with distorted ones. Our tests show that an increase in the number of sub-patterns stored within the network does not have any adverse effect on the recall/store time of EdgeHGN approach (section 3.7.2). In fact, the scalability of EdgeHGN scheme will not affected by the number of stored patterns within the EdgeHGN network. Our experimental results demonstrate the fact that EdgeHGN is able to achieve very low error rate of ($\sim$2.7%) in classifying 50 facial image classes of 1000 test images (section 3.7.3). This high accuracy rate is accompanied by remarkable scalability features. In contrast to the rest of hierarchical models already proposed in the literature, EdgeHGN's pattern matching capability and the small response time, that remains insensitive to the increases in the number of stored patterns, can make this approach remarkably suitable for clouds.

ii. A distributed scalable cloud data management, referred to as EdgeHGN_MR, is formulated that enables seamless data access and distribution using single-cycle learning associative memory-based algorithms. EdgeHGN_MR scheme allows complex data representations to be used as keys for Map and Reduce operations; allowing content-association based data retrieval and storage within cloud. This will improve MapReduce-based parallel processing by replacing referential data access mechanisms with more versatile and distributable associative functions that allow complex data relations such as images to be easily encoded into the keys as patterns. These patterns can be applied in a variety of applications that require content recognition, such as image databases, searches within large multimedia files and data mining. In this regards, three extensions of EdgeHGN based MapReduce, referred to as EddgeHGN_MRv1, EdgeHGN_MRv2 and EdgeHGN_MRv3, are introduced that each utilise EdgeHGN network processing model in conjunction with MapReduce computational framework to effectively deal with data-intensive scenarios in the face of excessive amount of classification data to process, voluminous training datasets or massive number of processing

neurons in the network, respectively (section 4.3). The algorithmic strengths of the MapReduce approach are investigated for the first time in regards to the effectiveness of one-shot learning-based parallelism provisioned via our distributed pattern recognition approach, EdgeHGN. The principle of associative-memory-based learning was implemented through the use of hierarchically connected layers, with local feature learning at the lowest layer and upper layers combining features into higher representations. The EdgeHGN-based MapReduce approach to cloud-based data processing is unique.

Our experimental results show that EdgeHGN based MapReduce works exceptionally well in dealing with both small to large size data counts due to its parallel one-shot learning mechanism where the size of input data has minimal effect on the time of its single-cycle in-network processing (section 4.4.1). Performance evaluation results against state-of-the-art parallel processing techniques such as Giraph, GPS, Mizan and GraphLab demonstrate that the performance of MapReduce parallelism as a scalable scheme for data processing in clouds can be significantly improved by transforming the data processing operations into one-shot distributed pattern matching sub-tasks, in which distributed computations are performed in-network, enabling data storage and retrieval by association (instead of pre-set referential data access mechanisms) (section 4.5.2). Moreover, EdgeHGN_MR demonstrates efficient memory usage across all experiments as memory requirements per GN node to maintain the bias array do not increase disproportionately with the increase in the number of stored patterns (section 4.5.2).

iii. The capabilities of the proposed EdgeHGN scheme is investigated in the context of distributed data processing in wireless sensor networks (WSNs). We demonstrated the ability of the proposed recognition technique to learn and recognise complex patterns using minimal information and resources to effectively perform classification tasks. This distributed pattern matching approach outlines a new type of the WSN that detects macroscopic events by

collating diverse sensor data, locally and in real-time, into meaningful patterns. EdgeHGN reduces complexity of the WSN by eliminating the need for complex computations for event classification which increases its potential for wide-spread use. EdgeHGN is a distributed computational model in nature and hence it can be readily deployed over a distributed network setup such as WSN to provide an effective front-end recognition scheme for event detection.

The performance of EdgeHGN recognition approach in WSN was evaluated and benchmarked against SVM (linear and poly-2) and SOM in relation to its precision, recall, accuracy and error rates. Experimental results show that EdgeHGN scheme offers better recognition accuracy and higher recall value with very low error rate (section 5.3). Moreover, EdgeHGN demonstrates fast recognition performance, where it takes only up to 6msec (average result of less than 3msec) for a sensor data to be recalled or memorised. This in turn makes this approach a feasible strategy for implementing real-time even detection within WSNs (section 5.3). Furthermore, EdgeHGN approach offers very low memory consumption for event data storage mainly due to its simple bias array representation (section 5.3.1). This memory efficiency is best achieved when dealing with small input sub-pattern sizes, as EdgeHGN utilises only a small portion of the memory space in a typical physical sensor node in WSN network. The distributed nature of the scheme also lowers storage capacity requirements per node and incurs lesser communication cost, thus improving the response-time characteristic.

7.3 Future Research

With the current technological advancements under the label of Internet-of-Things (IoT) and System-of-Systems (SoS), fully integrated large-scale systems are possible. However, the question of discovering the knowledge potentials of such systems must be addressed effectively. One solution can be achieved using distributed pattern recognition schemes when dealing with such Internet-scale environments. In this

regard, the research conducted in this PhD thesis has mainly aimed at developing a capability for large-scale pattern recognition involving complex and large-scale data. As a result, EdgeHGN_MR approach has been proposed as a scalable scheme for distributed pattern recognition, that incorporates in-network processing mechanism along with single-cycle learning capability. EdgeHGN demonstrated high recall accuracy with low computational complexity for distributed pattern recognition. To further improve upon EdgeHGN_MR approach, we can propose the following future research directions with main focus on algorithm and application improvements:

7.3.1 Algorithm-Specific Research

The following two potential future improvements on EdgeHGN algorithmic design have been identified:

i. *Bias array design*: The current bias array architecture is heavily reliant upon storing unique entries within the bias array to address memory efficiency requirements. Further research on bias array design can lead to better memory management as the size of bias array will grow in size as more and more unique patterned are being stored.
ii. *Structural representation*: EdgeHGN reduces the required number of processing GN nodes significantly by utilising Dropfall scheme within a hierarchical structure. However, the number of GN nodes can still increase dealing with large pattern sizes and dimensions. As a result, further study can be carried out on finding potential structural representations, other than the existing hierarchical form.

7.3.2 Application-Specific Research

The following four potential future improvements on EdgeHGN application related design have been identified:

i. *Movement and tracking*: In the evaluation of the EdgeHGN distributed pattern recognition, this research did not look at object movement and tracking. Potential

future works can be carried out on the recognition of patterns that may involve some structural changes, such as rotation, transformation or relocation.

ii. *Heterogeneous clouds*: Extension of the proposed EdgeHGN_MR distributed data management scheme to heterogeneous clouds (with different processing capacity nodes). This work will investigate the proposed distributed data management scheme at different levels of granularity. Improving upon the existing cloud data management models for fault-tolerance and scalability and reducing MapReduce communication overheads by introducing data locality. Moreover, investigating innovative cloud applications that benefit from such schemes, benchmarking and validating the results to find asymptotical limits of the technique through rigorous testing and simulation.

iii. *Combinational logic*: MapReduce is only one of a dozen or so patterns of parallel processing, massive communication and data distribution. Asanovic (2006) classifies 13 such patterns or dwarfs. Another important pattern of these thirteen is Combinational Logic, so named after networks of logical functions implemented in electronic designs, however generalised to networks of stream processing functions for software architectures dealing with massive amounts of data and large processing tasks of varying duration. Reconciling combinational logic with associated memory concepts, such as EdgeHGN, in particular for adaptive and fast data access, aggregation and movement can be another potential field of future research.

iv. *Spatio-temporal event detection*: Distributed event classification within resource-constrained WSNs has been proposed in chapter 5 by offering a front-end recognition mechanism at the sensory level. To further extend this for WSNs, potential future research can be carried out to include spatio-temporal analysis of events by observing the frequency and distribution of events within the network.

This Page Intentionally Left Blank

Vita

This thesis was proofread by a professionally qualified editor from Elite Editing on the 22nd of August 2016.

Academic Editing Services provided by Elite Editing comply with the National Standards for Editing. Services are also conducted within the guidelines of the policy developed collaboratively by the Deans and Directors of Graduate Studies with the Council of Australian Societies of Editors entitled ‘The editing of research theses by professional editors (2008)’. Compliance with this policy ensures that the student’s thesis retains its integrity as entirely the work of the student.

Further information about editing services provided by Elite Editing are available at http://www.eliteediting.com.au.

This Page Intentionally Left Blank

References

Abdi. H. and Williams, L.J. (2010). Principal component analysis, Wiley Interdisciplinary Reviews, Computational Statistics, pp. 433 – 459.

Agrawal, D., Bernstein, P., Bertino, E., Davidson, S. (2012). Challenges and Opportunities with Big Data: A community white paper developed by leading researchers across the United States. Whitepaper, Computing Community Consortium, [Online Document]. http://cra.org/ccc/docs/init/bigdatawhitepaper.pdf

Alfehaid, W.M. (2013). Distributed Pattern Recognition Schemes for Wireless Sensor Networks, PhD thesis, Faculty of Information Technology, Monash University.

Alham, N.K. (2011). Parallelizing support vector machines for scalable image annotation [Ph.D. thesis], Brunel University, Uxbridge, UK.

Alham, N. K., Li, M., Liu, Y., and Qi, M. (2013). A MapReduce-based distributed SVM ensemble for scalable image classification and annotation, Computers and Mathematics with Applications, volume 66, no. 10, pp. 1920 – 1934.

Alpaydin, E, (2004). Introduction to Machine Learning, Second Edition, MIT Press, 584 pp.

Amazon Elastic Cloud Computing (2011), [Online Document]. http://aws.amazon.com/ec2/

Anderson, C. (2008). The end of theory: The data deluge makes the scientific method obsolete. [Online Document]. http://www.wired.com/science/discoveries/magazine/16-07/pb_theory

Apache Giraph (2013). [Online Document]. http://giraph.apache.org/

Apache Hadoop (2010). [Online Document]. http://hadoop.apache.org/

Apache Hadoop MapReduce (2011). [Online Document]. http://hadoop.apache.org/docs/r1.2.1/index.html

Apache Hadoop YARN (2013). [Online Document]. http://hadoop.apache.org

Apache Lucene Core (2008). [Online Document]. http://lucene.apache.org/core/

Apache Mahout Canopy Clustering (2012). [Online Document]. https://mahout.apache.org/users/clustering/canopy-clustering.html

Apache Mahout Software Foundation (2012). [Online Document]. https://mahout.apache.org/

Apache Spark (2013). [Online Document]. https://spark.apache.org/

Asanovik, K. (2006). The Landscape of Parallel Computing Research: a view from Berkeley, EECS Department, University of California, Berkeley.

Assuncao, L., Goncalves, C. and Cunha, J.C. (2012). Data analytics in the cloud with flexible MapReduce workflows, In Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom), Washington, DC, USA.

Baig Z.A., Baqer M., Khan A.I., (2006). A Pattern Recognition Scheme for Distributed Denial of Service (DDOS) Attacks in Wireless Sensor Networks, In Proceedings of the 18th International Conference on Pattern Recognition.

Baqer, M. (2008). Energy Ecient Event Recognition for Wireless Sensor Networks, PhD thesis, Faculty of Information Technology, Monash University.

Baqer, M., Khan A.I. (2007), Energy-efficient pattern recognition approach for wireless sensor network, In M. Palaniswami, S. Marusic & Y.W. Law (Eds.), Proceedings of the 2007 International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Melbourne, Australia, pp. 509-514.

Baqer M., Khan A.I, and Baig Z.A., (2005). Implementing a graph neuron array for pattern recognition within unstructured wireless sensor networks, in Proceedings of EUC Workshops, pp. 208 – 217.

Banerjee, T., Xie, B. and Agrawal, D. P. (2008). Fault tolerant multiple event detection in a wireless sensor network, J. Parallel and Distributed Computing, 68(9): 1222–1234.

Basirat, A. H., & Khan, A. I. (2013). Scalable event detection in wireless sensor networks using a novel content-based pattern recognition scheme. Proceedings of the 3rd International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering (PARENG 2013), Civil-Comp Press, Stirlingshire, UK, pp. 53–59.

Battiti, R. and Colla, A. M. (1994). Democracy in neural nets: voting schemes for classification, Neural Networks. 7(4): pp. 691–707.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks 5(2), pp. 157 - 166.

Bishop, C.M. (2006). Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag.

Blazejewski, A. and Coggins, R. (2004). Application of self-organizing maps to clustering of high-frequency financial data, ACSW Frontiers '04: Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalization, Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 85–90.

Brewer, E.A. (2005). Combining Systems and Databases: A Search Engine Retrospective, Database Systems, 4th Edition, Cambridge, MA.

Bondi, A. B. (2000). Characteristics of scalability and their impact on performance, Proceedings of the 2nd international workshop on Software and performance, ACM, New York, NY, USA, pp. 195–203.

Breiman, L. (2001). Random forests, Technical report, Department of Statistics, University of California, Machine Learning 45(1), pp. 5 – 32.

Brin, S., and Page. L., (1998). The anatomy of a large-scale hyper-textual Web search engine, Computer Networks, pp. 107 – 117.

Casali, D., Costantini, G., Perfetti, R. and Ricci, E. (2006). Associative memory design using support vector machines, Neural Networks, IEEE Transactions on 17(5): pp. 1165 – 1174.

Catterall, E., Van Laerhoven, K. and Strohbach, M. (2003). Self-organization in adhoc sensor networks: an empirical study, ICAL 2003: Proceedings of the eighth international conference on Artificial life, MIT Press, Cambridge, MA, USA, pp. 260–263.

Chaiken, R., Jenkins, B., Larson, P.A., Ramsey, B., Shakib, D., Weaver, S., and Zhou, J. (2008). "SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets", In Proceedings of Very Large Database Systems (VLDB), 1(2):1265 - 1276.

Chen, S., Schlosser, S.W., (2008). Map-Reduce Meets Wider Varieties of Applications, Technical Report, Intel.

Chen, J., Chen, Y., Du, X., Li, C., Lu, J., Zhao, S., and Zhou, X. (2013). Big data challenge: a data management perspective. Frontiers of Computer Science, 7(2):157–164, doi: 10.1007/s11704-013-3903-7.

Cheng, J. and Wang, K. (2007). Active learning for image retrieval with co-svm, Pattern Recognition 40(1): 330 – 334.

Cheung, Y.-M. and Law, L. (2007). Rival-model penalized self-organizing map, IEEE Transactions on Neural Networks, 18(1), pp. 289 – 295.

Chih Yang, H., Dasdan, A., Hsiao, R.L., and Parker, D. S. (2007). "Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters", In Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pp. 1029 – 1040, New York, NY, USA.

Chisvin L. and Duckworth J.R., (1989). Content-addressable and associative memory: alternatives to the ubiquitous RAM, IEEE Computation., 22, pp. 51– 64.

Chow, T. W. S. and Huang, D. (2008). Data Reduction for Pattern Recognition and Data Analysis, Springer, Berlin / Heidelberg, pp. 81–109.

Congedo, G., Dimauro, G., Impedovo, S. and Pirlo, G. (1995). Segmentation of Numeric Strings, pp. 1038 – 1041.

Connor, J.T., Martin R.D., and Atlas, L.E. (1994). Recurrent neural networks and robust time series prediction, IEEE Neural Networks, volume 5, pp. 240 – 254.

Cortes, C. and Vapnik, V. (1995). Support-vector networks, Mach. Learn. 20(3): 273–297.

Cruz, B., Sossa, H. and Barr´on, R. (2007). A new two-level associative memory for efficient pattern restoration, Neural Process. Lett. 25(1): pp. 1–16.

Culler, D. E., Estrin, D. and Srivastava, M. B. (2004). Guest editor's introduction: Overview of sensor networks, IEEE Computer 37(8): 41–49.

Daintith, J., and Wright. E., (2008). A Dictionary of Computing, 6th edition, Oxford, UK, Oxford University Press, pp. 14 – 15.

Dasarathy, B.V. (2002). Data mining tasks and methods, classification: Nearest-neighbor approaches, in handbook of data mining and knowledge discovery, pp. 288 – 298.

Dean, J., and Ghemawat, S. (2004). "MapReduce: Simplified Data Processing on Large Clusters", In OSDI'04: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation, Berkeley, CA, USA.

DeWitt. D., and Gray. J., (1992). Parallel Database Systems: The Future of High Performance Database Systems. Communications of the ACM, 35(6): 85–98, ISSN 0001-0782. doi:10.1145/129888.129894.

Dimauro, G., Impedovo, S., Pirlo, G., and Salzo, A, (2009). Automatic Bank check processing: A New Engineered System, International Journal of Pattern Recognition and Artificial Intelligence, pp. 467 – 504.

Donoho, D.L. (2000). High-dimensional data analysis: The courses and blessings of dimensionality, Lecture delivered at the mathematical Challenges of the 21st Century conference of The American Math Society.

Dong, J.-X., Krzyzak, A. and Suen, C. (2005). Fast svm training algorithm with decomposition on very large data sets, Pattern Analysis and Machine Intelligence, IEEE Transactions on 27(4): pp. 603 – 618.

Duda, R.O., Stork, D. G., and Hart, P.E. (2001). Pattern Classification, 2nd edition, New York, Wiley.

Duin, R. P. W. and Tax, D. M. J. (2000). Experiments with classifier combining rules, MCS ’00: Proceedings of the 1st International Workshop on Multiple Classifier Systems, Springer-Verlag, London, UK, pp. 16–29.

Elaine R. M. (2008). The impact of next-generation sequencing technology on genetics. Trends in Genetics, 24(3):133–141.

EBI (2013). The European Bioinformatics Institute bank, [Online Document] http://www.ebi.ac.uk/about/background

Faloutsos, M., Faloutsos, P., and Faloutsos. C. (1999). On power-law relationships of the Internet topology. In SIGCOMM, pages 251–262.

Feed-forward neural network, (2011). [Online Document]. http://en.wikipedia.org/wiki/Feedforward_neural_network

Fei-Fei, L., Fergus, R. and Perona, P. (2006). One-shot learning of object categories, IEEE Transactions on Pattern Analysis and Machine Intelligence 28(4): 594–611.

Figueiredo, R., Dinda, P., Fortes, J., (2003). A Case for Grid Computing on Virtual Machines, In Proceedings of 23rd International Conference on Distributed Computing Systems, pp. 550–559.

Fox, G. C., Aktas, M. S., Aydin, G., Donnellan, A., Gadgil, H., Granat, R., Pallickara, S., Parker, J., Pierce, M. E., Oh, S., Rundle, J., Sayar, A. and Scharber, M. (2005). Building sensor filter grids: Information architecture for the data deluge, Proceedings of the First International Conference on Semantics, Knowledge and Grid, IEEE Computer Society, Washington, DC, USA, p. 2.

Frank, A. and Asuncion, A. (2010). UCI machine learning repository. [Online Document] http://archive.ics.uci.edu/ml\

Freund, Y., Iyer, R., and Schapire, R.E. (1998). An efficient boosting algorithm for combining preferences, in Proceedings of the 15th International Conference on Machine Learning.

Freund, Y., and Schapire, R. E. (1999). Large Margin Classification Using the Perceptron Algorithm. Machine Learning, 37 (3): 277.

Friedman, N., Geiger, D., and Goldszmit, M. (1997). Bayesian network classifiers, Machine Laming, volume 29, pp. 131- 163.

Gamma, E., Helm, E., Johnson R. (1995). Design Patterns: Elements of Reusable Object-oriented Software, Addison Wesley, Reading, MA, USA.

Gantz, J., and Reinsel, D., (2012). The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. Study report, IDC, [Online Document] www.emc.com/leadership/digital-universe/index.htm

Gashler, M. and Martinez, T. (2011). Temporal Nonlinear Dimensionality Reduction, In Proceedings of the International Joint Conference on Neural Networks, pp. 1959 – 1966.

Gelernter D., and Carriero, N., (1985). Generative Communication in Linda, ACM Transactions on Programming Languages and Systems, 7(1):80–112.

Giorgetti, G., Gupta, S. K. S. and Manes, G. (2007). Wireless localization using self-organizing maps, IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks, ACM, New York, NY, USA, pp. 293 – 302.

GoGrid Cloud Hosting, (2011). [Online Document]. (Accessed on 22/05/2011) http://www.gogrid.com

Gonzalez, J., Low, Y., Gu, H., Bickson, D. and Guestrin C. (2012). PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs, Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation.

GraphLab Open Source, (2009). [Online Document]. http://graphlab.org/projects/index.html

Gray, J., Bell, G., and Szalay, A. (2006). Petascale computational systems. IEEE Computer, 39(1):110–112.

Gropp, W., Thakur, R. and Lusk, E. (1999). Using MPI-2: Advanced Features of the Message Passing Interface, MIT Press, Cambridge, MA, USA.

Grzegorz Malewicz, Matthew H. Austern, Aart J.C. Bik, James Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. (2010). Pregel: A System for Large-Scale Graph Processing. In Proceedings of ACM SIGMOD International Conference on Data Management, 135-146.

Gu, R., Shen, F., and Huang, Y. (2013). A parallel computing platform for training large scale neural networks. in Proceedings of the IEEE International Conference on Big Data, pp. 376 – 384.

Hagan, M.H, Demuth, H.B., and Beale, M.H. (1996). Neural Network Design, PWS Publishing.

Halevy, A., Norvig, P., and Pereira. F. (2009). The Unreasonable Effectiveness of Data. IEEE Intelligent Systems, 24(2):8–12, ISSN 1541-1672. doi: 10.1109/MIS.2009.36.

Harrop, P. & Das, R. (2010). Wireless Sensor Networks 2010-2020 [Online Document] http://www.sbdi.co.kr/cart/data/info/2049291810_54e75de0_IDTechEx_Wireless_Sensor_Networks_2010-2020_Pamphlet.pdf?ckattempt=1

Hebb, D. O. (1988). The organization of behavior, pp. 43–54.

Helland, P. (2011). If You Have Too Much Data, then Good Enough Is Good Enough. Communications of the ACM, 54(6):40–47, doi: 10.1145/1953122.1953140.

Hey, A. J. G. and Trefethen, A. E. (2003). The data deluge: An e-science perspective. [Online Document] http://eprints.ecs.soton.ac.uk/7648/

Hey, T., Tansly, S., and Tolle, K. (2009). The Fourth Paradigm: Data-Intensive Scientific Discovery. Microsoft Research.

Hopfield network, (2012). [Online Document]. http://en.wikipedia.org/wiki/Hopfield_network

Hopfield, J. and Tank, D. (1985). Neural computation of decisions in optimization problems, Biological Cybernetics 52: 141–152.

Huang, G.-B., Mao, K., Siew, C.-K. and Huang, D.-S. (2005). Fast modular network implementation for support vector machines, Neural Networks, IEEE Transactions on 16(6): 1651–1663.

Huqqani, A. A., Schikuta, E., and Mann, E., (2014). Parallelized neural networks as a service, in Proceedings of the International Joint Conference on Neural Networks (IJCNN ’14), pp. 2282 – 2289.

Ikram, A.A., Ibrahim, S., Sardaraz, M., Tahir, M., Bajwa, H., and Bach, C. (2013). Neural network based cloud computing platform for bioinformatics, in Proceedings of the 9th Annual Conference on Long Island Systems, Applications and Technology (LISAT ’13), pp. 1 – 6, Farmingdale, NY, USA.

Introduction to Support Vector Machines, (2012). [Online Document]. http://docs.opencv.org/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html

Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. (2007). "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks". In Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, pp. 59 – 72, New York, NY, USA.

Jiang, J., Zhang, J., Yang, G., Zhang, D., and Zhang, L. (2010). Application of back propagation neural network in the classification of high resolution remote sensing image: take remote sensing image of Beijing for instance. in Proceedings of the 18th International Conference on Geo-informatics, pp. 1 – 6.

Jain, A.K., Duin, R.P., and Mao, J. (2000). Statistical pattern recognition: A review, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 1, pp. 4–37, 2000.

Joachims, T. (2008). Making Large-scale Support Vector Machine Learning Practical, pp. 169 – 184.

Kaisler, S., Armour, F., Espinosa, J.A., Money. W. (2013). Big Data: Issues and Challenges Moving Forward. In Proceedings of the 46th Hawaii International Conference on System Sciences, pp. 995–1004.

Kalos, A. (2005). Automated heuristic growing of neural networks for nonlinear time series models, in proceedings of 2005 IEEE International Joint Conference on Neural Networks, Vol. 1, pp. 320–325.

Kamath, C. and Musick, R. (1998). Scalable pattern recognition for large-scale scientific data mining.

Kanan, H., and Khanian, M. (2012). Reduction of neural network training time using an adaptive fuzzy approach in real time applications, International Journal of Information and Electronics Engineering, volume 2, no. 3, pp. 470 – 474.

Kasabov, N. K. (1996). Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering, MIT Press, Cambridge, MA, USA.

Kbir, M. A., Maalmi, K., Benslimane, R. and Benkirane, H. (2000). Hierarchical fuzzy partition for pattern classification with fuzzy if-then rules, Pattern Recognition Letter 21(6-7): 503–509.

Khan, A.I. (2002). A Peer-to-Peer Associative Memory Network for Intelligent Information Systems, In Proceedings of the 13th Australasian Conference on Information Systems, Vol. 1.

Khan A.I. (2007), System infrastructure design for an end-to-end E-application for lab data acquisition to repository deposition, In Proceedings of the 9th International Conference on Information Integration and Web-Based Applications and Services, Jakarta, Indonesia, pp. 287-296.

Khan, A. I. and Mihailescu, P. (2004). Parallel pattern recognition computations within a wireless sensor network, ICPR (1), pp. 777–780.

Khan, A.I. and Muhamad Amin, A.H. (2007). One-shot Associative Memory Method for Distorted Pattern Recognition Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Australia, pp. 705 - 709.

Khan, A. I. and Muhamad Amin, A. H. (2009). Integrating Sensory Data within a Structural Analysis Grid, Saxe-Coburg Publications, Stirlingshire, UK.

Khan A.I., Isreb M., Spindler R.S., (2004). A parallel distributed application of the wireless sensor network, In Proceedings of the 7th International Conference on High Performance Computing and Grid in Asia Pacific Region.

Khayyat, Z., Awara, K., Alonazi, A., Jamjoom, H., Williams, D., and Kalnis. P. (2013). Mizan: A system for dynamic load balancing in large-scale graph processing, In EuroSys '13, pages 169 - 182.

Khoa, N. L. D., Sakakibara, K., and Nishikawa, I. (2006). Stock price forecasting using back propagation neural networks with time and profit based adjusted weight factors. in Proceedings of the SICE-ICASE International Joint Conference, pp. 5484 – 5488, IEEE, Busan, Republic of korea.

Kim, J. H., Yoon, S. H., Kim, Y. H., Park, E. H., Ntuen, C. A. and Sohn, K. (1992). Efficient matching algorithm by a hybrid Hopfield network for object recognition, in S. K. Rogers (ed.), Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 1709 of Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference, pp. 908–916.

Kimmel, R., Shaked, D., Elad, M. and Sobel, I. (2005). Space-dependent color gamut mapping: a variational approach, IEEE Transactions on Image Processing, 14(6): 796–803.

Kohonen, T. (2000). Self-Organizing Maps, 3rd edition, Springer.

Kopetz, H. (2011). Internet of things,in Real-Time Systems, Real-Time Systems Series, pp. 307–323, Springer US.

Kosko, B., (1992). Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Kosko, B. (1988). Bidirectional associative memories, IEEE Transaction on Systems and Cyber Networks. 18(1): 49–60.

Kraska, T. (2013). Finding the Needle in the Big Data Systems Haystack, IEEE Internet Computing, 17(1):84–86, ISSN 1089-7801. doi: 10.1109/MIC.2013.10.

Kulakov, A. and Davcev, D. (2005). Tracking of unusual events in wireless sensor networks based on artificial neural-networks algorithms, ITCC ’05: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC’05) - Volume II, IEEE Computer Society, Washington, DC, USA, pp. 534 – 539.

Kumar, V., Grama, A., Gupta, A., and Karypis, G. (2002). Introduction to Parallel Computing, Benjamin Cummings / Addison Wesley, San Francisco, California, USA.

Kuncheva, L. I. (2004). Combining Pattern Classifiers: Methods and Algorithms, Wiley-Interscience.

Laney, D. (2001). 3D Data Management: Controlling Data Volume, Velocity and Variety. Technical report, META Group, Inc (now Gartner, Inc.).

LeCun, Y. and Bengio, Y. (1995). Convolutional networks for images, speech, and timeseries, in M. A. Arbib (ed.), The Handbook of Brain Theory and Neural Networks, MIT Press.

Li, Y. and Parker, L. E. (2008). Detecting and monitoring time-related abnormal events using a wireless sensor network and mobile robot, IROS, pp. 3292–3298.

Li, Y., Tang, Z., Xia, G. and Wang, R. (2005). A positively self-feedbacked Hopfield neural network architecture for crossbar switching, IEEE Transactions on Circuits and Systems, Regular Papers, 52(1): 200–206.

Lin, X., Soergel, D. and Marchionini, G. (1991). A self-organizing semantic map for information retrieval, SIGIR '91: Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval, ACM, New York, NY, USA, pp. 262–269.

Liu, Z., Li, H., and Miao, G. (2010). MapReduce-based back propagation neural network over large scalemobile data, in Proceedings of the 6th International Conference on Natural Computation (ICNC '10), pp. 1726 – 1730.

Long. L.N. and Gupta, A. (2008). Scalable massively parallel artificial neural networks. Journal of Aerospace Computing, Information and Communication, volume 5, no. 1, pp. 3 – 15.

Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J.M. (2010). GraphLab: a new framework for parallel machine learning, in proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI).

Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin C., and Hellerstein J.M. (2012). Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. PVLDB.

Lowe, M. (1999). On the storage capacity of the Hopfield model with biased patterns, IEEE Transactions on Information Theory 45(1): 314–318.

Malewicz, G., Austern, M.H., Bik, A.J.C, Dehnert, J., Horn, I., Leiser, N., and Czajkowski, G., (2010). Pregel: A System for Large-Scale Graph Processing. In Proceedings of ACM SIGMOD international Conference on Management of Data, Indianapolis, IN, USA, pp. 135-146

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., and Hung Byers, A., (2011). Big data: The next frontier for innovation, competition, and productivity. Analyst report, McKinsey Global Institute, [Online Document]. http://www.mckinsey.com/insights/mgi/research/technology_and_innovation/big_data_the_next_frontier_for_innovation

Martinez, A. M. and Kak, A. C. (2001). PCA versus LDA, IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 228 –233. doi:10.1109/34.908974.

Mavroforakis, M., and Theodoridis, S. (2006). A geometric approach to support vector machine (svm) classification, IEEE Transactions on Neural Networks, volume 17, pp. 671–682.

Miller, E., Matsakis, N. and Viola, P. (2000). Learning from one example through shared densities on transforms, In proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Vol. 1, pp. 464 – 471.

MNIST Dataset (2012). [Online Document] http://yann.lecun.com/exdb/mnist/

Muhamad Amin A.H., and Khan A.I. (2008). Commodity-Grid Based Distributed Pattern Recognition Framework, 6th Australasian Symposium on Grid Computing and e-Research, Wollongong, NSW, Australia.

Muhamad Amin, A.H., Khan A. I., and Mahmood R. A., (2009), A distributed event detection scheme for wireless sensor networks, In Proceedings of the 7th

International Conference on Advances in Mobile Computing and Multimedia, ACM Press, New York NY USA, pp. 295-299.

NameNode Performance. (2008). Compare Name-Node Performance When Journaling Is Performed into Local Hard-Drives or NFS, [Online Document]. http://issues.apache.org/jira/browse/HADOOP-3860

Nadal, J.-P. (1989). Study of a growth algorithm for a feed forward network, Int. J. Neural Syst. 1(1): 55–59.

NASA Kennedy Space Center, (2014). WWW Server logs. [Online Document]. http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html

Nascimento, M. A. and Chitkara, V. (2002). Color-based image retrieval using binary signatures, in proceedings of the 2002 ACM symposium on Applied computing, ACM, New York, NY, USA, pp. 687–692.

Nasution, B.B. and Khan A.I., (2008). A Hierarchical Graph Neuron Scheme for Real-Time Pattern Recognition, IEEE Transactions on Neural Networks, pp. 212-229.

Nguyen, D. and Ho, T. (2006). A bottom-up method for simplifying support vector solutions, Neural Networks, IEEE Transactions on 17(3): 792–796.

Nguyen, D., and Widrow, B. (2010). Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights, in Proceedings of the International Joint Conference on Neural Networks, volume 3, pp. 21 – –26.

Ohkuma K., (1993). A Hierarchical Associative Memory Consisting of Multi-Layer Associative Modules, In Proceedings of 1993 International Joint Conference on Neural Networks (IJCNN'93), Nagoya, Japan.

OpenSource Forum. (2011), MapReduce: More Power, Less Code. [Online Document].
http://www.opensourceforu.com/2011/03/mapreduce-more-power-less-code-hadoop/

Pal, S. K. and Mitra, P. (2004). Pattern Recognition Algorithms for Data Mining: Scalability, Knowledge Discovery, and Soft Granular Computing, Chapman & Hall, Ltd., London, UK, UK.

Percolator, Dremel and Pregel: Alternatives to Hadoop. (2012). [Online Document].
http://www.rosebt.com/blog/percolator-dremel-and-pregel-alternatives-to-hadoop

Ratnaparkhi A. (1997). A simple introduction to maximum entropy models for natural language processing, Technical Report, Institute for Research in Cognitive Science, University of Pennsylvania.

Recurrent Neural Networks in Ruby (2012). [Online Document].
http://blog.josephwilk.net/ruby/recurrent-neural-networks-in-ruby.html

Resnick, P. and Varian, H. (1997). Recommender systems, Communications of the ACM, 40(3):56-58.

Ricci, F., Rokach, L., and Shapira, B. (2011). Introduction to Recommender Systems Handbook, Recommender Systems Handbook, Springer, pp. 1-35.

Ritter, G., Sussner, P., and Diaz-de Leon, J. (1998). Morphological associative memories, IEEE Transactions on Neural Networks, volume 9, pp. 281– 293.

Roman-Godinez, I., Lopez-Yanez, I. and Yanez-Marquez, C. (2009). Classifying patterns in bioinformatics databases by using alpha-beta associative memories, pp. 187–210.

Rueda, L. and Herrera, M. (2008). Linear dimensionality reduction by maximizing the chernoff distance in the transformed space, Pattern Recognition 41(10): 3138–3152.

Rumelhart, D. E. and Zipser, D. (1988). Feature discovery by competitive learning, Connectionist models and their implications: readings from cognitive science, Ablex Publishing Corp., Norwood, NJ, USA, pp. 205 – 242.

Russom, P., (2011). Big Data Analytics, Best practices report, The Data Warehousing Institute.

Saha, S. and Bajcsy, P. (2003). System design issues in single-hop wireless sensor networks, The IASTED International Conference on Communications, Internet and Information Technology 2003 (CIIT 2003), Scottsdale, AZ, USA, 17-19 November 2003.

Salihoglu, S., and Widom. J. (2013). GPS: A Graph Processing System. In SSDBM '13, pages 22:1 - 22:12.

Schlegel, D. (2011). Categorization of High-Dimensional Knowledge-Based Representations of File Data Using Abstract Self-Organizing Maps. [Online Document]. http://cs.oswego.edu/~dschlege/sitev2/courses/468/Cog468%20ASOM%20Presentation.htm

Schlimmer, J. C. and Granger, Jr., R. H. (1986). Incremental learning from noisy data, Machine Learning 1(3): 317–354.

Shiers, J. (2009). “Grid today, clouds on the horizon”, Computer Physics Communications, pp. 559-563.

Shih, K.P., Wang, S.S., Chen, H.C. and Yang, P.H. (2008). Collect: Collaborative event detection and tracking in wireless heterogeneous sensor networks, Computer Communications 31(14): 3124 – 3136.

Stanford Network Analysis Project, (2015). [Online Document]. http://snap.stanford.edu/data/

Stonebraker, M., Madden, S., and Dubey. P. (2013). Intel Big Data Science and Technology Center Vision and Execution Plan. SIGMOD Record, 42(1):44–49.

Sussner P. and Valle M.E., (2006). Gray-Scale Morphological Associative Memories, IEEE Transactions on Neural Networks, vol. 17, pp. 559-570.

Szalay, A., Bunn, A., Gray, J., Foster, I., Raicu, I. (2006). “The Importance of Data Locality in Distributed Computing Applications”, In Proceedings of the NSF Workflow Workshop.

Tao, Y., Papadias, D., and Sun, J. (2003). TPR-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries, VLDB, pp. 790 – 801.

Theodoridis, Y., Vazirgiannis, M., and Sellis, T.K. (1996). Spatio-Temporal Indexing for Large Multimedia Applications. pp. 441 – 448.

Toselli A., and Widlund, O.B. (2005): Domain Decomposition Methods – Algorithms and Theory. Springer-Verlag, Berlin and Heidelberg.

Trajan R.E. and Trojanowski A.E., (1984). Finding a maximum independent set. SIAM journal of Computing, 25(3): 537 – 546.

Valiant. L.G., (1990). A bridging model for parallel computation, CACM, 33(8), pp. 103-111.

Vesanto, J., Himberg, J., Alhoniemi E, and Parhankangas, E. (2010). Self-organizing Map in Matlab: the SOM Toolbox, In Proceedings of the Matlab DSP Conference, Espoo, Finland, pp. 35 – 40.

Vivanco, R. A., Demko, A. and Pizzi, N. J. (2005). Scopira: A pattern recognition application framework for biomedical datasets, Proceedings of the Fourth International Conference on Machine Learning and Applications, IEEE Computer Society, Washington, DC, USA, pp. 165–170.

Wang, H., Zheng, H. and Azuaje, F. (2007). Poisson-based self-organizing feature maps and hierarchical clustering for serial analysis of gene expression data, IEEE/ACM Transactions on Computational Biology, 4(2): 163 – 175.

Wang, Y., Li, B., Luo, R., Chen, Y., Xu, N., and Yang, H. (2014). Energy efficient neural networks for big data analytics, in Proceedings of the 17th International Conference on Design, Automation and Test in Europe (DATE 2014).

Wilson, R. C. (2009). Parallel Hopfield networks, Neural Computations. 21(3): 831–850.

Wythoff, B. J. (1993). Backpropagation neural networks: A tutorial, Chemometrics and Intelligent Laboratory Systems 18: 115–155.

Yahoo Developer Network (2008). [Online Document]. https://developer.yahoo.com/blogs/hadoop/scaling-hadoop-4000-nodes-yahoo-410.html

Yang, F. and Paindavoine, M. (2003). Implementation of an RBF neural network on embedded systems: real-time face tracking and identity verification, Neural Networks, IEEE Transactions on 14(5): 1162–1175.

Zaremba, M., St-Laurent, L., Niemann, O. and Richardson, D. (2000). Integration of self organizing maps with spatial indexing for efficient processing of multi-dimensional data, GIS '00: Proceedings of the 8th ACM international symposium on Advances in geographic information systems, ACM, New York, NY, USA, pp. 77 – 82.