Dynamic Peer-to-Peer Construction of Clusters

by

Pranith Reddy Kadaru
Advisor: Dr. Peter Graham

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Department of Computer Science
University of Manitoba

Copyright © 2009 by Pranith Reddy Kadaru
Advisor: Dr. Peter Graham

Abstract

Dynamic Peer-to-Peer Construction of Clusters

Pranith Reddy Kadaru

Advisor: Dr. Peter Graham

Master of Science

Department of Computer Science

University of Manitoba

2009

The use of parallel computing is increasing with the need to solve ever more complex problems. Unfortunately, while the cost of parallel systems (including clusters and small-scale shared memory machines) has decreased, such machines are still not within the reach of many users. This is particularly true if large numbers of processors are needed. A largely untapped resource for doing some, simpler, types of parallel computing are temporarily idle machines in distributed environments. Such environments range from the simple (identical machines connected via a LAN) to the complex (heterogenous machines connected via the Internet).

In this thesis I describe a system for dynamically clustering together similar machines distributed across the Internet. This is done in a peer-to-peer (P2P) fashion with the goal of ultimately forming useful compute clusters without the need for a heavily centralized software system overseeing the process. In this sense my work builds on so-called "volunteer computing" efforts, such as SETI@Home but with the goal of supporting a different class of compute problems.

I first consider the characteristics that are necessary to form good clusters of shared machines that can be used together effectively. Second, I exploit simple clustering algorithms to group together appropriate machines using the identified characteristics. My system assembles workstations into clusters which are, in some sense, “close” in terms of bandwidth, latency and/or number of network hops and that are also computationally similar in terms of processor speed, memory capacity and available hard disk space. Finally, I assess the conditions under which my proposed system might be effective via simulation using generated network topologies that are intended to reflect real-world characteristics. The results of these simulations suggest that my system is tunable to different conditions and that the algorithms presented can effectively group together appropriate machines to form clusters and can also manage those clusters effectively as the constituent machines join and leave the system.

Dedication

Dedicated to my parents.

Acknowledgements

I thank my advisor Dr. Peter Graham for all his support through out my thesis work. Without Dr. Graham my thesis would not have been possible. I am also grateful to my thesis committee members, Dr. Rasit Eskicioglu and Dr. Paul card, for having provided a good environment during my defense, and for all the valuable suggestions on my thesis document.

I thank my family mother, father, and brother for their constant love and support. I would also like to thank my friends for extending their support at all times.

Above all, I would like to thank the God for giving this wonderful life.

Contents

1 Introduction **1**

2 Related Work **5**

2.1 Resource Discovery and Management 5

2.2 Election Algorithms 11

2.3 Bidding Techniques 15

2.4 Clustering Algorithms 16

2.5 Multicasting 19

3 Problem and Motivation **22**

4 Solution Strategy **26**

4.1 Definitions 30

4.2 Choosing a cluster to join 36

4.3 Leader Processing 36

4.4 Cluster Splitting 38

4.4.1 Capacity-based Splitting 39

4.4.2 Proximity-based Splitting 40

4.4.3 Improved Proximity-based Splitting 44
4.5 Node Departure 44
4.6 Cluster Merging 45
4.7 Node Failure 46
4.8 Use of Resulting Clusters 46

5 Implementation And Simulation Results 49

5.1 Implementation of the architecture 50
5.1.1 Key Data Structures 50
5.1.2 Joining of a node 53
5.1.3 Primary Leader Message Processing 53
5.2 Simulation Setup 55
5.2.1 Network Topology Generation Using BRITE 56
5.2.2 Assigning Hard Disk Space, Processor Speeds, and Memory Sizes to Each Node 58
5.2.3 Simulation Parameters 59
5.3 Simulation Results 60
5.3.1 Controlling the Weighting Factors 61
5.3.1.1 Controlling the Weighting Factors of Network Characteristics 62
5.3.1.2 Controlling the Weighting Factors of Node Characteristics 67
5.3.2 Impact of Available Resources 71
5.3.2.1 Connectedness of Nodes 71

5.3.2.2 Number of Nodes 74

5.3.3 Assessment of Different Splitting Algorithms 83

6 Conclusions and Future work 90

6.1 Conclusions 90

6.2 Future work 92

List of Tables

5.1 Default Parameters 60

5.2 Default Discovered Clusters 61

5.3 Varying the bandwidth weighting factor 62

5.4 Varying the latency weighting factor 64

5.5 Varying the number of hops weighting factor 65

5.6 Varying the proximity weighting factor 66

5.7 Varying the processor speed similarity weighting factor 68

5.8 Varying the memory size similarity weighting factor 69

5.9 Varying the hard disk similarity weighting factor 71

5.10 Number of Edges of Each node vs Weighted average Proximity of all clusters 72

5.11 Number of Edges of Each node vs Weighted average bandwidth of all clusters 73

5.12 Number of Edges of Each node vs Weighted average latency of all clusters 74

5.13 Number of Edges of Each node vs Weighted average number of hops of all clusters 75

5.14 Total number of nodes vs Weighted average bandwidth of all clusters . . 77

5.15 Total number of nodes vs Weighted average latency of all clusters 77

5.16 Total number of nodes vs Weighted average number of hops of all clusters 79

5.17 Total number of nodes vs Weighted average proximity of all clusters . . . 80

5.18 Total number of nodes vs Weighted average processor speed similarity of all clusters 81

5.19 Total number of nodes vs Weighted average memory size similarity of all clusters 83

5.20 Total number of nodes vs Weighted average hard disk space similarity of all clusters 84

5.21 Effect of different splitting algorithms on weighted average proximity . . 85

5.22 Effect of different splitting algorithms on weighted average bandwidth . . 86

5.23 Effect of different splitting algorithms on weighted average latency 88

5.24 Effect of different splitting algorithms on weighted average number of hops 88

List of Figures

2.1 NEVRLATE Server Organization (adapted from [9]) 7

2.2 Example of a very simple resource description and its AVTree (adapted from [6]) 8

2.3 STORM architecture (adapted from [15]) 11

2.4 Example Dendogram (adapted from [24]) 17

2.5 Unicasting 20

2.6 Multicasting 20

4.1 Joining of a Node 27

4.2 Cluster Splitting 48

5.1 Flowchart for joining of a node 54

5.2 Flowchart for Primary Leader Message Processing 55

5.3 Top-down hierarchical topology (adapted from [31]) 57

5.4 β_1 vs Weighted Average Bandwidth of all clusters 63

5.5 β_2 vs Weighted Average Latency of all clusters 64

5.6 β_3 vs Weighted Average Hops of all clusters 65

5.7 α_1 vs Weighted Average Proximity of all clusters 67

5.8 α_2 vs Weighted Average Processor Speed Similarity of all clusters 68

5.9 α_3 vs Weighted Average Memory Size Similarity of all clusters 70

5.10 α_4 vs Weighted Average Hard Disk Space Similarity of all clusters 70

5.11 Number of Edges of Each node vs Weighted average Proximity of all clusters 72

5.12 Number of Edges of Each node vs Weighted average Bandwidth of all clusters 73

5.13 Number of Edges of Each node vs Weighted average Latency of all clusters 75

5.14 Number of Edges of Each node vs Weighted average Hops of all clusters . 76

5.15 Total number of nodes vs Weighted Average Bandwidth of all clusters . . 78

5.16 Total number of nodes vs Weighted Average Latency of all clusters . . . 78

5.17 Total number of nodes vs Weighted Average Hops of all clusters 79

5.18 Total number of nodes vs Weighted Average Proximity of all clusters . . 80

5.19 Total number of nodes vs Weighted Average Processor Speed Similarity of all clusters 82

5.20 Total number of nodes vs Weighted Average Memory Size Similarity of all clusters 82

5.21 Total number of nodes vs Weighted Average Hard Disk Space Similarity of all clusters 84

5.22 Effect of different splitting algorithms on weighted average proximity . . 85

5.23 Effect of different splitting algorithms on weighted average bandwidth . . 87

5.24 Effect of different splitting algorithms on weighted average latency 87

5.25 Effect of different splitting algorithms on weighted average number of hops 89

Chapter 1

Introduction

In recent years, there has been rapid improvement in computation, storage and networking technologies in concert with the development of demanding new applications in industry and academia. Ever faster computers continue to support the development of new applications unforeseen a decade earlier. Large increases in storage capacity coupled with decreasing costs have also made it practical to store massive amounts of data. Finally, the growing prevalence of high-speed network interconnection between machines facilitates both cooperation in computation and the sharing of ever larger volumes of data. This growth in computing capabilities has significantly increased user expectations to be able to solve even more complex problems and has fueled a hunger for the computational resources needed to do so.

This improvement in technologies alone, however, is not sufficient to meet the demands of many resource-intensive applications. Researchers working in areas that range from scientific simulation to engineering design and that include applications such as Monte Carlo simulation, image processing, aircraft modeling, genome analysis, etc. re-

quire computing platforms with capabilities that cannot be provided using a conventional single processor computer. This has lead to the creation of a range of parallel and distributed systems including very expensive large-scale shared memory multiprocessor parallel computers, relatively inexpensive compute clusters and the increasingly common multi-core desktop machines. Each of these types of machines meets the needs of a specific class of applications and, correspondingly, their users.

The most demanding applications are solved using dedicated computing facilities. Such systems cost millions of dollars to purchase and operate, and while very effective, are outside of the price range for many potential users. Smaller scale systems are also increasingly common (e.g. department-level compute clusters), but have correspondingly less capabilities and are commonly targeted at relatively small scale problems. For some specific problems, it may be possible to exploit the large collective capabilities offered by underutilized machines scattered throughout organizations around the world. This is already being done for so-called “volunteer computing” efforts, such as SETI@Home [34]. An interesting question that arises is whether or not this pool of resources can be harnessed for other loosely coupled computing problems and, if so, on what scale this is possible.

Emerging resource-intensive applications that require large scale computational services, but which have modest communication demands, could potentially benefit greatly from exploiting distributed computing resources. In this case, locating and assembling appropriate sets of available computational resources is of great importance, particularly to organizations or researchers that may have inadequate access to powerful computation capabilities using expensive, large scale parallel machines. The idea is to build loosely

coupled clusters of available machines that can be used effectively together to solve a reasonable range of problems. The challenge is to be able to identify collections of machines that can be used together in this way. This process will be complicated by the heterogeneous and dynamic nature of the machines typically found in such distributed environments.

Today's Internet consists of many very capable workstations connected by relatively high speed interconnection networks. Assembling these workstations into a cluster-like infrastructure offers the promise of a cost-effective solution to meet the high-end computational capabilities required by researchers and organizations. Because the available computational resources are autonomous, however, their ongoing participation cannot be assured. In terms of identifying and assembling the machines into useful groups (or *clusters*) of machines, this suggests coupling these workstations in a peer-to-peer fashion. The entire process is also burdened with challenges introduced by the heterogeneous nature of the machines and their geographic distribution.

In this thesis, I first consider the characteristics that should be considered to form good clusters of shared machines that can be used effectively together. Second, I consider the use of simple clustering algorithms to group together appropriate machines using the identified characteristics. I then describe a peer-to-peer resource discovery system, that I have developed, that will enable users to incorporate workstations, distributed over potentially large geographic areas, into a cluster-like computing infrastructure. The system described facilitates the assembling of workstations into clusters which are, in some sense, "close" in terms of bandwidth, latency and/or number of network hops and that are also computationally similar in terms of processor speed, memory capacity and

hard disk size. Finally, I assess the conditions under which my proposed system might be effective via simulation using generated network topologies that are intended to reflect real-world characteristics. The results of these simulations are discussed and some basic conclusions are drawn about the effectiveness of the technique and where and when it might be useful.

Chapter 2

Related Work

This section presents a survey of related work, which serves as background to my thesis research. This related work falls into five categories, which are resource discovery and management, election algorithms, bidding algorithms, clustering techniques and, to a lesser extent, multicasting.

2.1 Resource Discovery and Management

Resource discovery is the process of locating resources in computer-based networks or, the recently popular term, "grids". For the purpose of this thesis, a grid may be considered to simply be a collection of machines that are coordinated for resource sharing and problem solving to form one or more "virtual organizations" in which users can work collaboratively. Resources can be any real or conceptual object, including CPU cycles, hard disk space, software, etc. that is of interest to the users. While, for convenience, we discuss grids as an example of a large-scale distributed computing environment, the

work presented later is not restricted to grid-based systems.

A Grid Directory is a registry system that is commonly used for recording information about the location and availability of resources as well as other useful information about a grid (its structure, status, etc.) Registration is the process of registering a resource in a directory. Resource lookup is the process of finding needed resources using a directory.

Globus [13, 14] , one of the most widely used grid computing systems, facilitates modular deployment of grid services using the Globus Metacomputing Toolkit, which consists of a set of several software components that collectively provide mechanisms for resource discovery, resource management, fault tolerance, communication, security, etc. Among these, the Globus Resource Allocation Manager (GRAM) is responsible for resource allocation as well as process creation and management services while the Nexus communication library provides the underlying communication services. The Metacomputing Directory Services (MDS) provides information about the structure and status of the grid and the Grid Security Infrastructure (GSI) provides authentication and related security services.

The MDS [13, 14] itself has two components: the Grid Resource Information Service (GRIS) and the Grid Index Information Service (GIIS). The Grid Resource Information Service provides resource discovery services. Resource information providers periodically update the GRIS. The Grid Index Information Service provides information about the structure of the entire grid by collecting information from multiple GRIS providers. Resource discovery is done by querying the MDS as a whole.

Nevrlate [9] is a scalable resource discovery system that organizes directories (on separate servers) containing resource location information in a logical two-dimensional

structure. The directories are organized into sets such that registration is performed in the horizontal dimension and lookup in the vertical dimension as shown in Figure 2.1. When a server joins or leaves the network, set splitting or set absorption is performed to maintain acceptable dimensions for the network.

Every resource is registered in one server in every set so lookup can be done in any set. When a node wants to join, it can send a join request to any NEVRLATE node. The receiving node, based on its current estimates of each set, determines which set the joining node should be assigned to (to maintain the best network balance) and then forwards the joining node's id to the corresponding set. During the process of node joining and leaving, the receiving node determines whether set splitting or set joining should be performed to maintain load balance.

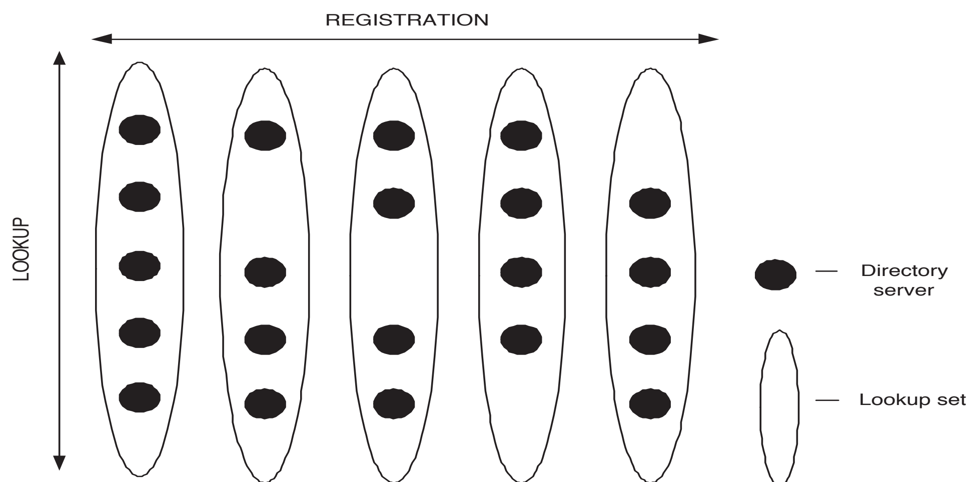
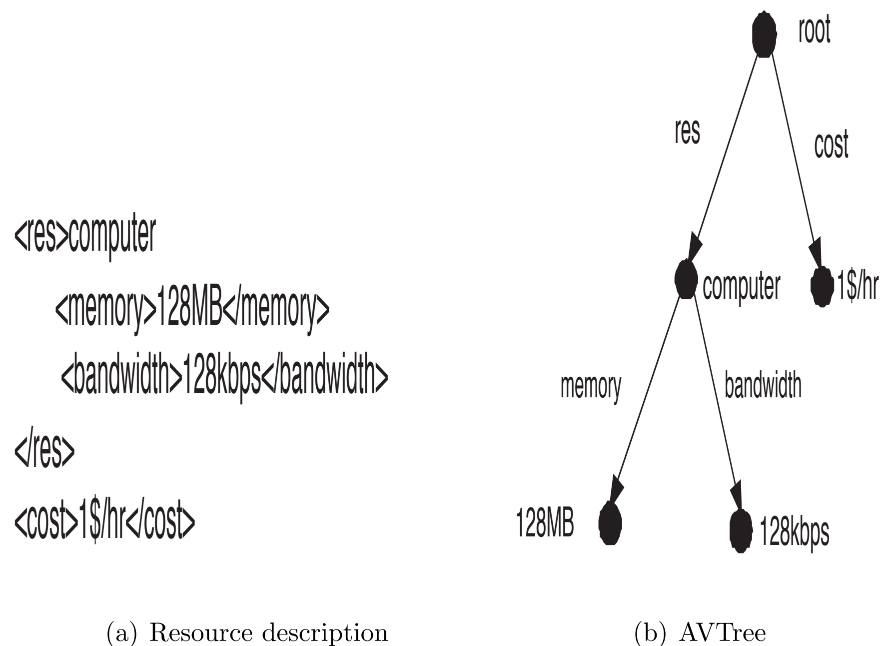


Figure 2.1: NEVRLATE Server Organization (adapted from [9])

INS/Twine [6] is a scalable peer-to-peer resource discovery system that uses the intentional naming system [1] from MIT [4]. This system is designed to organize resource

[1]A naming system designed for naming and discovering resources in networks based on the attributes of resources rather than simply their name or location

resolvers (directories that store resource information) in a peer-to-peer fashion (similar to Gnutella [40]). Resources are described in INS/Twine using hierarchies of attribute-value pairs represented using a convenient language (e.g., XML) as shown in Figure 2.2(a) that can be converted into an attribute-value tree (AVTree) as shown in Figure 2.2(b), which is structured for efficient lookup processing. Resolvers store resource information and resolve queries from the clients by matching an AVtree describing a resource to the query.



(a) Resource description (b) AVTree

Figure 2.2: Example of a very simple resource description and its AVTree (adapted from [6])

Neptune [35] is a policy driven resource management system that dynamically reconfigures resources in a computing cluster based on active usage metrics and a set of service level performance goals. Neptune defines a resource model that consists of service domains and resources. Each service domain represents resource ownership corresponding

to a set of applications and data and also specifies a domain policy. Each domain policy defines a set of resource constraints and performance objectives. Neptune allocates resources based on each domain's policies.

When resource request events or threshold events (e.g. overload) occur, Neptune reconfigures its resources and coordinates resource allocations through "plan formulation". Plan formulation consists of a set of tasks needed to change the current allocation of resources, which is accomplished by a series of interactions with appropriate resource managers. A resource request event is a request for a change in resource assignment, for example, to request additional servers. Threshold events occur when unacceptable performance, as defined by service domain policies, is detected. Neptune monitors its resource managers for threshold events and carries out its decisions by locating and reallocating appropriate resources. Resource reconfiguration plans are composed of a hierarchy of tasks. Each task within a plan may therefore itself launch dependent tasks (sub-plans). Individual tasks set up event listeners to await completion of their sub-tasks. Plan formulation is driven by application specific resource allocation policies.

The Condor-G [23] system uses the protocols from Globus and Condor [29] (a system that looks for idle workstations and schedules jobs onto them) to allow users to control and use grid resources as if all the resources were available locally. Condor-G makes use of protocols from the Globus toolkit for remote resource access and it makes use of protocols from Condor for computation management and remote execution control.

Cactus-G [5] is a grid-enabled computational framework that combines the Cactus system [1] and the MPICH-G2 [25] grid enabled message passing library. The top level in Cactus-G consists of the grid-aware application itself which is built on several

lower layers. At the top of these lower layers lie the various cactus application "thorns" (used to perform the necessary calculations) followed by grid-aware infrastructure thorns, which provide all the features and drivers that the application thorns need. The lowest layer is the grid-enabled communication library (MPICH-G2), an implementation of the MPI [33] parallel programming standard capable of running MPI programs across widely distributed heterogeneous computing resources.

STORM [15] is a scalable resource management tool that makes uses of three types of daemons to handle job launching, scheduling and monitoring in an environment of networked multiprocessors. STORM has been implemented over the Quadrics network [36] which is built using the Elan network interface and the Elite switch. The Elan network interface links Quadrics network nodes to a computing node that contains multiple processing elements (PEs) as shown in Figure 2.3. The three types of daemons in STORM are the Machine Manager (one per management node), the Node Manager (one per compute node) and the Program Launcher (many per compute node, one per PE). The Machine Manager allocates resources for jobs. Node managers are responsible for managing resources on a single computing node. When a new job arrives, the Machine Manager queues the job and allocates the job to processing elements as soon as they become available. After job allocation is completed, the Machine Manager broadcasts a job-launch message to all the Node Managers (on nodes which are assigned to the job) and those Node managers will launch the job. The Machine Manager can also broadcast binary image and data files to the nodes before the execution of a job so that the nodes can access the files locally. The Program Launcher handles individual application processes for the Node Manager.

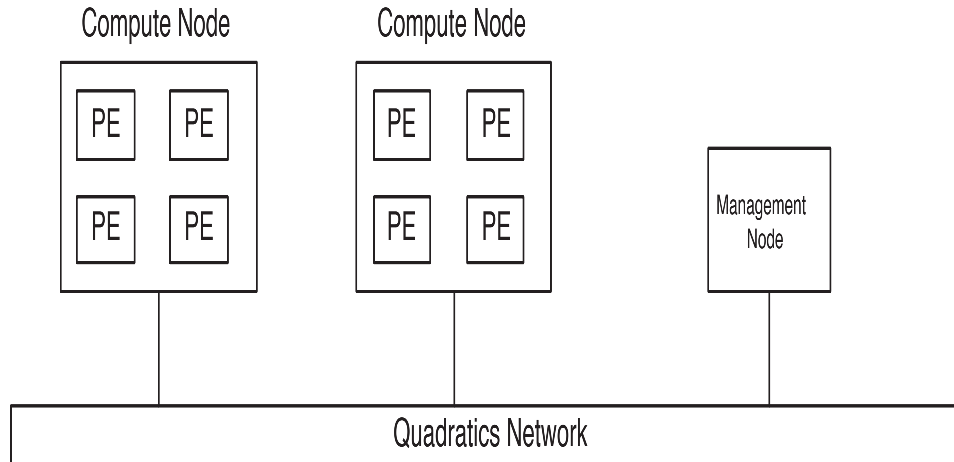


Figure 2.3: STORM architecture (adapted from [15])

Resource discovery and management are vital components in grid systems. While Globus is widely used, Nevrlate organizes its resource directories in a logical two-dimensional grid to achieve $O(\sqrt{n})$ message complexity for both registration and lookup. INS/Twine organizes resource resolvers in a peer-to-peer fashion and uses attribute-value trees to provide highly flexible resource description based resolving. Neptune can dynamically reconfigure its resources through plan formulation when unacceptable performance, as defined by domain policies, is detected and STORM's computational model directly supports increasingly common multiprocessors.

2.2 Election Algorithms

Election algorithms are used to elect a leader in a group of entities in a distributed system so that the leader can perform a special role such as being a coordinator, initiator, etc. in the group. An entity could be a process, computer, etc. Perhaps the best known election algorithm is Garcia-Molina's "Bully" algorithm [17] upon which most election schemes

are based. Much work has been done over the years on election algorithms and the field is now largely mature with more recent contributions (such as work by Kim et al. [26]) being applicable to specific application domains only. As elections are fundamental to what will be proposed in this thesis, some representative techniques are now reviewed.

The Bully election algorithm [17] elects the coordinator, which has the highest identification number of all the competing nodes in the system. Each node in the algorithm has a unique identification number. A node with identification number i that is trying to become leader contacts the nodes that have identification numbers greater than i. If any of the nodes respond, node i discontinues its attempt to become leader. If none of the nodes respond, node i sends “I am elected messages” to all nodes with identification number less than i.

A leader election algorithm for broadcast networks has been proposed by Israel et al. [12]. During the operation of this election algorithm, the nodes are partitioned into fragments. Each fragment consists of a candidate node (interim leader) and its supporting nodes. During the initial phase of the algorithm, each node is a fragment of size 1. The fragment's nodes report back to the candidate node the size of the fragment and the identity of its largest (in terms of fragment size) neighbor. If the size of the candidate node's fragment is not X times bigger [12] than the maximal neighbor's fragment (optimal value of X=3), then the fragment joins its maximal neighbor's fragment. The algorithm terminates with a single leader when the candidate node learns that the fragment it represents has no neighbors. This algorithm elects a leader but is not fault tolerant.

A distributed, t-resilient algorithm that elects a leader when at most t nodes are faulty is presented by Alon et al. [22]. In the basic algorithm, one or more nodes may

start the algorithm. These nodes are candidates for leadership, and are called kings. Each node, at any given time, knows who its king is. Each king tries to annex other nodes to its kingdom. An annexed king stops being a king but all the nodes previously annexed by it remain in its domain. Each king knows the size of its kingdom. To annex a neighbor X, a token is sent from king Y to node X. This token is forwarded to X's current king, Z. If the size of the kingdom of Z is smaller than X, then node X joins the kingdom of Y. If the size of the kingdom of Z is greater than X, then the token, with a reject message attached, is returned. To make this algorithm t-resilient, t+1 tokens are sent by each node to ensure that at least one of them is processed.

Huang et al. [20] proposed a leader election protocol for a uniform ring of processors. A ring of processors is considered uniform if all the processors are similar in characteristics and run the same program. Initially, each processor has an arbitrary label in the range (0, 1 . . . ,n-1) where n is the number of processors. During the protocol, each process has several rules, where each rule has two parts: the guard and the move. The guard is a Boolean function [20] of the state of the processor and its neighbors. A processor makes a corresponding move if any of its rules can be applied to enter a new state, which is a function of its old state and the new states of its neighbors (as the neighbors can also make moves if any of their rules can be applied). A previously applicable rule may not be applied in the new state because of the change of state of its neighbors. At the end of the election protocol, none of the rules can be applied, and the processor with label 0 is the elected leader in the ring of processors.

Frederickson et al. [16] proposed a leader election algorithm for synchronous ring of processors. Each processor has a unique Id, which is an integer. Initially, some

processors (called participating processors) autonomously wake up to initiate the leader election algorithm. Each participating processor creates a message process that stores the processor Id of the initiating processor and moves around the ring of processors at the rate of one message transmission every 2^k rounds where k is the Id of the initiating processor. The processors that have received a message process cannot wake up. A faster message process that overtakes a slower message process kills the slower message process and also a message process carrying processor Id k is killed at a participating processor with Id j if $j < k$. The arrival of a message process at its initiator results in electing the initiator as the leader of the synchronous ring of processors.

A leader election algorithm for a mobile ad hoc network has been proposed by Malpani et al. [30] that is highly adaptive to frequent network topology changes and makes use of diffusing computation to elect the "most valued node" (based on some system-specific performance characteristic such as remaining battery life, etc.). The algorithm makes use of three types of messages: election, ack and leader. Initially a source node (node that initiates the algorithm) sends an election message to all its neighbors. The neighboring nodes propagate the received election message to their neighbors except to the node from which it received the message. If a node receives an election message from its neighbors, it responds with an ack message that contains leader election information (such as the identifier and value of the most valued node that the node has from its neighbors). A node responds with an ack message to the node it recieved the election message from. After receiving the acks message, the source node broadcasts a leader message that contains the identity of the most valued node.

A similar leader election algorithm for bounded-degree networks where the number of

communication links for each node is bounded is presented by Chow et al. [11]. Instead of the most valued node, the algorithm elects the node that initiated the algorithm first based on a timestamp stored in its "Campaign-For-Leader" message.

A leader election algorithm for n processors that elects a processor as the leader has been presented by Gummadi et al. [3]. This algorithm elects the processor with the highest identification number as the leader. All the processors start the algorithm simultaneously. In each round, a processor broadcasts a message comprising of its identification number to its neighbors up to a certain depth. When a processor receives a message from its neighbors, it stops being a leader (dies) if the identification number received is greater than its identification number. If a processor is alive after a round, it proceeds to the next round.

2.3 Bidding Techniques

Auctioning is the process of putting an item on sale where the item will be sold to the highest bidder. Bidding is the process of submitting an offer (bid) to buy an item at an auction. Bidders are prospective purchasers who place a bid (offer value) to buy the desired item. The bidding process may be repetitive and is normally competitive.

A typical, simple, auction process consists of bid submission, bid evaluation (winner determination) and feedback to the bidders. In multidimensional auctions, all "dimensions" (corresponding to a number of different items, the qualitative attributes and quantity for each item) of a collection of goods are negotiable.

Bidding [10] has been used for resource allocation in a "computational market"

(CM). In such a computational market architecture, a wide-area network is divided into regions corresponding to local markets. Each local market consists of an auction server (which manages resource transactions within the local market), local brokers (that group resources into virtual resource clusters), supplier agents (which represent the resource providers) and consumer agents (that represent the resource consumers that bid for resources). Two heuristics for winner determination for multi-unit combinatorial auctions 2 (MUCAs) are proposed by Chunming et al. [10]. In the co-bids first approach (CFA), co-bids 3 are allocated first because they have greater constraints than other bids. In the no preference approach (NPA) all bids are treated alike and any inconsistencies that arise with co-bids are fixed after the initial allocation has been completed.

The design of an object framework that allows specifying the preferences of buyers, the rules of allocation and supplier offerings through a declarative interface for developing different winner determination algorithms is presented by Juhnyoung et al. [28]. This framework hides the complexities of the underlying bidding algorithms.

2.4 Clustering Algorithms

In general, a cluster is a group of similar objects in a particular domain (e.g. all students in first year might form a cluster). The process of grouping similar objects is called clustering. The "similarity" varies according to the properties of the objects and the resultant clusters. The most important step in clustering is to choose a subset of all available features, which are appropriate to the problem being solved, upon which to base the clustering. Choosing the best clustering algorithm depends on the preferred

properties of the resultant clusters. Normally, k features are selected and clusters are in some way determined by a measure of "nearness" in the resulting k-space.

Clustering can be either agglomerative or divisive. In agglomerative clustering, initially each object is a cluster and these clusters are merged until k clusters are formed, where k is the desired number of clusters. In divisive clustering, initially all the objects are contained in a single cluster and this cluster is repeatedly split until k clusters are formed.

Clustering algorithms can also be classified as being hierarchical or non-hierarchical (or partitionable). Hierarchical clustering [5, 24] generates a dendogram (hierarchy of nested clusterings) as shown in Figure 2.4, whereas non-hierarchical clustering generates a single, non- hierarchical clustering.

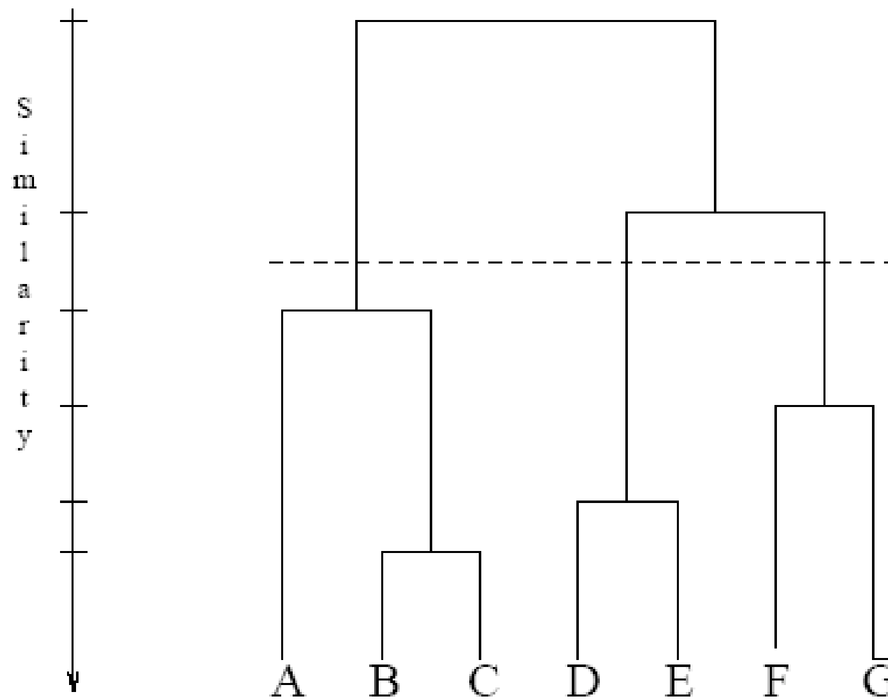


Figure 2.4: Example Dendogram (adapted from [24])

Using the Minimum-Cost Spanning Tree Clustering algorithm [2, 19, 39], a minimum-cost spanning tree is constructed by removing the k-1 largest (in terms of distance reflecting dissimilarity between the nodes connected by an edge) edges, where k is the desired number of clusters. Each of the k clusters created by removing k-1 edges has a path that connects all the nodes (vertices) within that cluster and the sum of the cost of the edges (paths between the nodes) in the k clusters is a minimum.

In the K-means Clustering algorithm [19, 24], k nodes are initially chosen randomly in a graph where each node will represent the centroid of an eventual cluster and k is the desired number of clusters. At each step, a node is added to the nearest cluster and the centroid is recalculated every time a node is added to a given cluster. This continues until all the nodes fall into the k clusters and a convergence criterion (for example, no reassignment of nodes to a new cluster) is met. This algorithm depends heavily on the k initially chosen nodes and may converge to a local rather than a global optimum if the initial k nodes are not properly chosen.

In the Maximum-cut Clustering Algorithm [2, 19, 41], a given graph is split into k clusters, where a vertex represents a node, an edge represents the path between the nodes and the weight of an edge represents the relative distance between the nodes. At each step, the vertices of the graph are partitioned into two subsets such that the sum of the weight of the edges between the two subsets is a maximum.

In BIRCH [42], clustering is performed by constructing a hierarchical structure called CF-Tree. A CF-Tree is a height balanced tree where the leaf nodes represent the current clusters and the parent nodes represent the cluster formed by merging its child nodes. Each cluster has a threshold radius and cluster is split if it exceeds this threshold.

CURE (Clustering Using REpresentatives) [18] uses a certain number of representative points rather than a single centroid for clustering. The algorithm starts with each point as a separate cluster and ends when the desired number of clusters in the system are attained. Each representative point is chosen such that they are far away from the center as well as each other. The distance between two clusters is the minimum distance between any pair of representative points. At each step, the clusters with the shortest distance between them are merged and the representative points are calculated for the new cluster formed.

2.5 Multicasting

Multicasting [37, 38] is a method of efficiently sending the same information to multiple recipients using a single transmission as shown in Figure 2.6. Unlike broadcasting, in multicasting, the recipients that are to receive the data can be restricted. Multicasting has several benefits over unicasting (shown in Figure 2.5) and broadcasting. The most important benefit of multicasting is reducing the overall bandwidth used by the transmission (compare Figures 2.6 and 2.5).

A multicast tree is initially constructed, which is a spanning tree of network routers connecting all senders and recipients in a multicast group (consisting of all the participants in a specific multicast communication). The path traversed by the datagrams from the sender to the multiple recipients in the multicast group follows this tree with datagrams being duplicated as needed at internal tree nodes (i.e. at routers). This in-network duplication makes multicasting bandwidth-efficient.

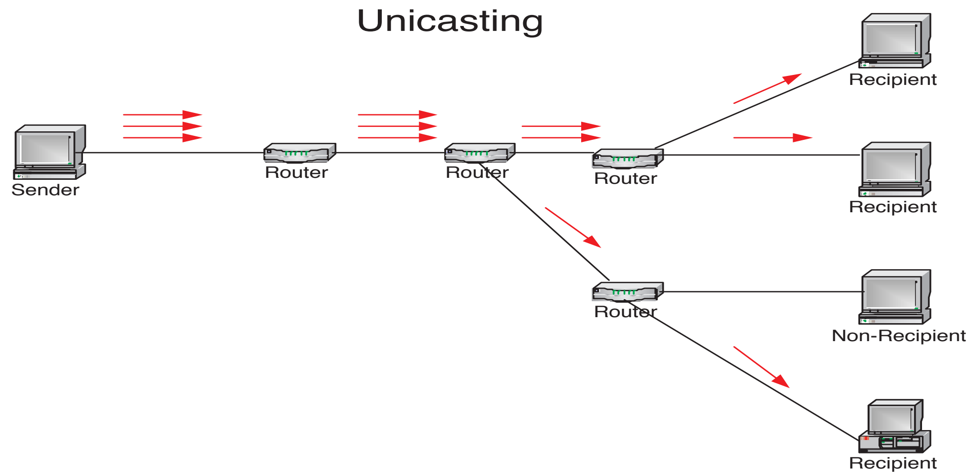


Figure 2.5: Unicasting

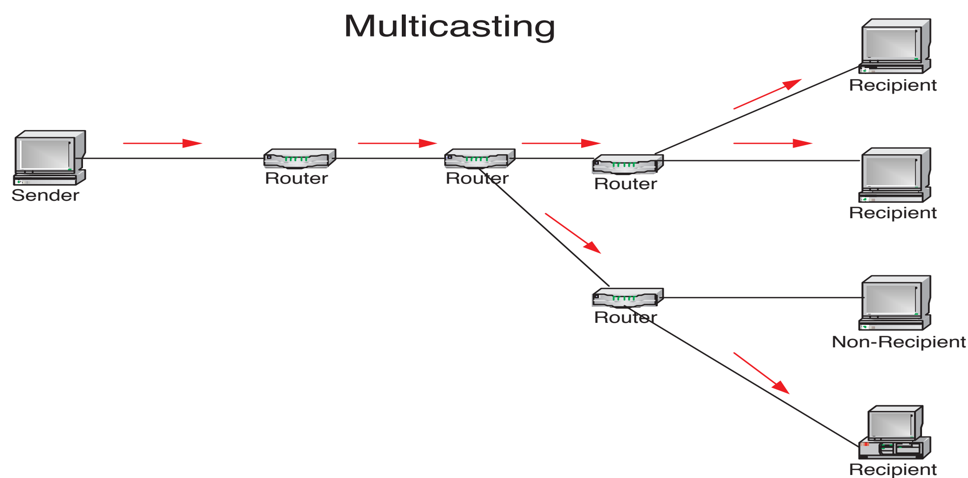


Figure 2.6: Multicasting

In IP Multicasting [7] (i.e. multicasting in a IP-based network), a multicast group is identified by a single multicast IP address. If a message is sent to the multicast address, all the members of the multicast group receive it. To join a multicast group, a machine informs the nearest router on the multicast tree that it wants to join. This is done using the Internet Group Management Protocol (IGMP) [8]. The multicast routers keep track of the multicast group members in their network. When a multicast message arrives, the

router checks the group id of the message and forwards the message only if a member of the multicast group exists in its network. Multicasting is not been extensively used in practise. Multicasting can be replaced with with several unicasting messages based on an agreed way of communication.

Chapter 3

Problem and Motivation

Many emerging scientific problems require a lot of computational power and this computational power cannot be provided by a single machine. High performance, parallel machines are very expensive and, hence, their use may be cost-prohibitive for some potential users. The problem of getting access to sufficient compute power can be addressed for some applications by utilizing the unused computational power available in idle workstations around an office, organization, metropolitan area, or even the world. Clusters built from idle workstations can provide a cost-effective solution to the problem of providing large amounts of computational power for applications not needing frequent communication between processes. This research deals with how to group these idle workstations to dynamically form a "grid" of clusters which is more peer-to-peer in nature rather than hierarchical (as is the current norm).

There are thousands of idle workstations available at any given time in any of a number of geographical areas. So far many ad-hoc grid-like collections have been constructed using machines from institutions consisting of everything from supercomputers to work-

stations but all under a single administrative domain. The computing power of all the idle workstations (especially desktop computers) around the world is a large source of untapped potential computing resources. Further, desktop computers now typically have fast Internet connections and their capabilities (processor speed, memory and hard disk space etc.) have also increased significantly making them more capable participants.

Until now, personal computers have typically been used primarily for file sharing using peer-to-peer (P2P) technology. With the deployment of high-speed networks, incorporating these machines into a grid-like architecture has become feasible. Peer-to-Peer systems offers much robustness (no single point of failure in the system), scalability and fault tolerance than centralized systems. In this research I extend and adapt work already done for volunteer computing [1] with the hope that a wider range of problems may be solved using collections of desktop machines. My research goal is to design and assess a resource discovery system, which can enable such desktop computers to be easily and dynamically grouped as parts of a grid of clusters.

Parallel programming tools allow users to run parallel programs on a cluster of machines as a parallel computer. To run parallel programs on such clusters of machines, the machines should be nearby for efficient communication. Also, the machines in a cluster need to have similar characteristics (processor speed, memory and storage) to help ensure consistent progress of all parallel processes across the machines. When the machines in a cluster are identical, it makes it easier to partition a program into parallel tasks. Also

[1] Volunteer Computing is a variation of metacomputing (where many machines cooperate to act as a single resource) that focuses on easing the effort required by people to incorporate their personal machines to be a part of a metacomputer used to solve problems of general interest to society at large (e.g. SETI@HOME [34])

there should be no bottlenecks (slow points) when the parallel tasks are running on these machines due to one task running slower than others.

The problem of constructing a grid of clusters which is more peer-to-peer rather than hierarchical is burdened with many challenges. The grid must be able to support resource sharing across distributed and heterogeneous environments. Also in a peer-to-peer environment, each workstation may exist in its own domain and have its own administrative policies. Further, peer workstations may join and leave grids unpredictably. These factors complicate the construction and use of peer-to-peer grids.

Resource discovery is, of course, a fundamental problem that has to be addressed in peer-to-peer grids. Peer-to-peer environments [21] lack global central authority, are strongly diverse in resource types and have unpredictable resource participation. These factors complicate resource discovery since it is impossible to rely on predefined hierarchies (with well known "representatives" coordinating access to a large number of machines) to support the process. Election algorithms [2] can, however, provide a mechanism to dynamically select a leader which can act as a dynamic, coordinating authority to manage resources in a peer-to- peer fashion. Further multicasting can be used to support efficient communication between initially unknown and changing entities (the peers) for management purposes. Finally, bidding and clustering provide a basis upon which resource grouping (and later selection) decisions may be made.

The construction of potentially large scale peer-to-peer grids is attractive in terms of cost effectiveness. It may also offer benefits in terms of robustness to failures, congestion

[2] In many distributed computing systems, the nodes need to cooperate with each other to perform a certain task. For this purpose, a unique node or a leader may be "elected"/chosen to perform some coordination in the system.

and possible overload. I will describe a peer-to-peer resource discovery system that eases the effort required by users to exploit unused machines that may be distributed over a potentially wide area network. I will also assess the usefulness of such a system for a range of networks and application characteristics. To the best of my knowledge, no such system currently exists. My system should offer significant improvements in cost effectiveness, scale, ease of use and robustness of operation relative to many existing parallel computing systems.

Chapter 4

Solution Strategy

The primary challenge in resource discovery for peer to peer grid environments is to gather and organize resource availability information in a form that is useful. The focus of this thesis is on the discovery of useful clusters (i.e. on "cluster construction") not on the later use of the discovered clusters. In this chapter, I describe a mechanism to do this based on techniques drawn from the areas of related work reviewed in Chapter 2.

Initially, I assume that a single, well known multicast address, used to identify a multicast group of existing clusters (of distributed computers that can work together effectively), exists. Each cluster elects a leader and has a maximum "capacity", which is a measure of the number of computers/nodes that can be effectively used and managed in the cluster[1]. At any given time, a computer/node can offer its services (resources) for use or fail or withdraw its services.

When a node wants to join the system (see Figure 4.1), it sends a 'Bid Request'

[1]While no concrete formula for determining the management overhead (incurred by a cluster leader) is discussed, it is necessary to incorporate such a measure in any cluster construction algorithm.

message to the associated multicast group. Initially, when no cluster exists, the node forms an initial cluster and becomes the single leader of that cluster. As a leader, it also becomes a recipient of the multicast group messages. If one or more clusters already exist, then a new node normally joins whichever cluster returns the highest bidding value (reflecting suitability) in response to its 'Bid Request' message (see below). At this time, a new leader may be elected because the new node may be better able to manage the cluster's nodes than the current leader.

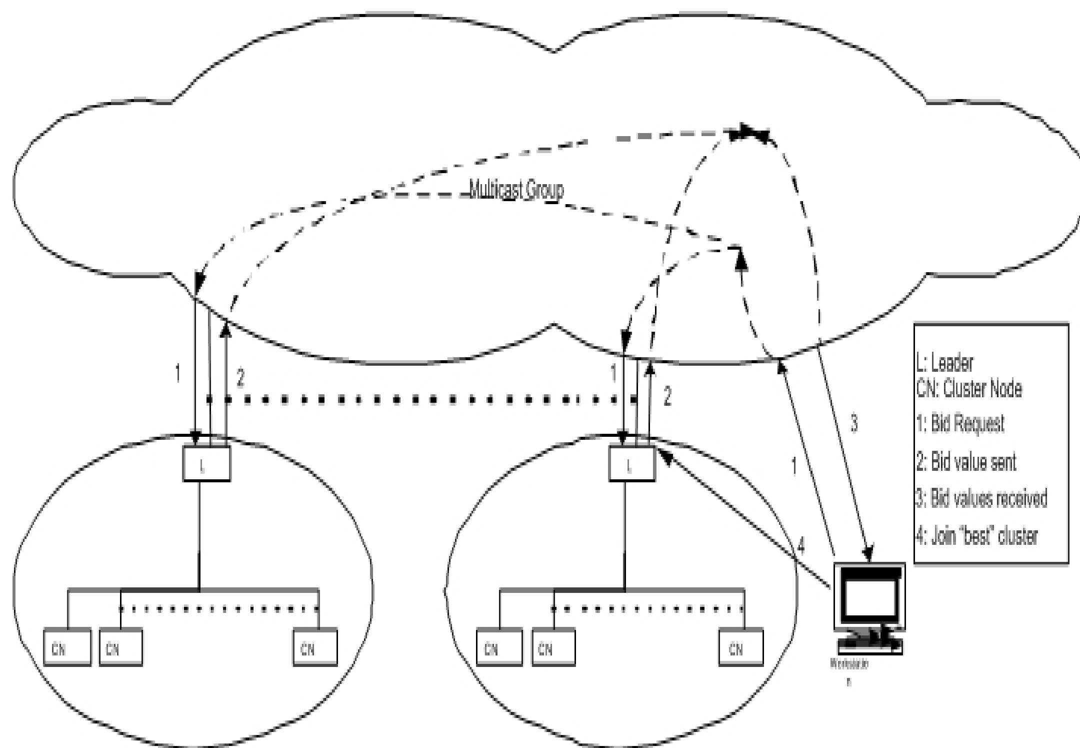


Figure 4.1: Joining of a Node

Bidding, by the existing cluster leaders, takes place when a new node conveys its desire to join the grid by sending a 'Bid Request' message to the multicast group, as just described. During the bidding process, the leader in each cluster will send its bidding

value (characterizing the cluster it leads and the cluster's suitability for incorporating the new node) to the requesting node. The new node may join the cluster that returns the highest bidding value or become a leader of a new cluster if all the returned bidding values are below a minimum acceptable threshold value. As discussed earlier, the primary characteristics determining the compatibility of two nodes to form a group/cluster are the distance between them (to ensure relatively efficient communication) and the similarity of the machines (to ensure effective coordinated use). These factors, possibly among others, therefore also determine the suitability of a cluster to host a new node. Accordingly, in this thesis, the bidding values are determined considering some combination of the following factors:

- Bandwidth of the connection from the new node to the corresponding cluster
- Latency of the connection from the new node to the corresponding cluster
- Number of hops on the connection from the new node to the corresponding cluster
- Similarity of hard disk capacity of the new node to nodes in the corresponding cluster
- Similarity of speed of the processor of the new node to nodes in the corresponding cluster [2]
- Similarity of memory capacity of the new node to nodes in the corresponding cluster

[2]Assuming homogenous machine types. Heterogeneity could be supported by another component in each bid.

These factors are effectively the features (dimensions) chosen for a clustering problem and the bidding values will reflect the "nearness" between the joining node and the corresponding cluster in the resulting k-space of the clustering dimensions.

The capacity of a single node is derived using the following factors:

- *Available* hard disk capacity of the node
- Approximate speed of the processor of the node [3]
- Memory capacity of the node

When a cluster's "management capacity" [4] has been reached (given the arrival of a new node), cluster splitting must take place to ensure the continued effective management and use of the cluster nodes (see example in Figures 4.2(a) and 4.2(b)). The maximum management capacity of a cluster is defined to be the management capacity of its leader.

Each cluster leader will maintain a ranking of all its nodes based on their management capacity. After a split, two clusters will be formed. In the first resulting cluster, the leader from the original cluster will become the new leader. In the second cluster, the "second best" node (i.e. the node with the second highest management capacity) in the former cluster will become the new leader as shown in Figure 4.2(c). These pairings are chosen because each leader node should be able to effectively manage the nodes in their corresponding clusters. The proportion of nodes assigned to each new cluster will be based on the management capacities of the newly determined leaders. (The management

[3] The exact speed of a shared processor cannot be determined. So an approximate value based on machine type is used. This could be determined via benchmark.

[4] Management capacity reflects the ability of a cluster's leader to manage nodes in the cluster as well as the desirability of having a cluster of the corresponding size.

capacities of the newly formed clusters will, again, be determined by the management capacity of their respective leaders.)

Just as cluster splitting is required when a cluster becomes too large, it may be desirable to merge/join very small clusters. This would, potentially, result in fewer but more useful clusters. The details of cluster merging, however, are not discussed in this thesis. Both cluster merging and the handling of the failure of nodes are left for future work.

4.1 Definitions

The following are the key definitions used in describing the algorithms presented.

Proximity

The proximity between two nodes is technically not a measure of their closeness but, instead, a measure of their relative effectiveness at communication and is defined as follows:

Proximity = $\beta_1[NBW] + \beta_2[\frac{1}{NL}] + \beta_3[\frac{1}{H}]$, where:

$NBW = \frac{BW}{BW_{max}}$, where:

BW = Bandwidth of the connection between the nodes

BW_{max} = Maximum available bandwidth in the network

$NL = \frac{L}{L_{min}}$, where:

L = Latency of the connection between the nodes

L_{min} = Minimum potential latency in the network

H = The number of hops on the connection between the nodes ($H_{min} = 1$)

N = Normalized

The weighting factors for the various terms, $\beta_1, \beta_2, \beta_3$, can be selected either statically or dynamically, as needed.

Processor Speed Similarity

The Processor Speed Similarity (PSS) between nodes A and B $= \frac{PS_A}{PS_B}$ if $PS_A \leq PS_B$

$= \frac{PS_B}{PS_A}$ if $PS_B < PS_A$,

where:

PS_A = Processor speed of node A,

PS_B = Processor speed of node B, and

$0 < PSS \leq 1$.

Memory Size similarity

Memory Size similarity(MSS) between nodes A and B $= \frac{MSS_A}{MSS_B}$ if $MSS_A \leq MSS_B$

$= \frac{MSS_B}{MSS_A}$ if $MSS_B < MSS_A$,

where:

MSS_A = Memory Size of node A,

MSS_B = Memory Size of node B, and

$0 < MSS \leq 1$.

Hard Disk Space Similarity

Hard Disk Space Similarity(HSS) between nodes A and B $= \frac{HSS_A}{HSS_B}$ if $HSS_A \leq HSS_B$

$= \frac{HSS_B}{HSS_A}$ if $HSS_B < HSS_A$,

where:

HSS_A = available Hard Disk Space on node A,

HSS_B = available Hard Disk Space on node B, and

$0 < HSS \leq 1$.

Bid

A bid response (message) is sent by the cluster leader in response to a 'Bid Request' message from a new node. The encapsulated bid value is computed as follows:

Bid = $\alpha_1[AP] + \alpha_2[APSS] + \alpha_3[AMSS] + \alpha_4[AHSS] + \alpha_5[CN]$, where:

AP = Average Proximity,

APSS = Average Processor Speed Similarity $0 < APSS \leq 1$,

AMSS = Average Memory Size Similarity $0 < AMSS \leq 1$,

AHSS = Average Hard Disk Space Similarity $0 < AHSS \leq 1$, and

CN = Cluster Need, either 0 or 1. (See Section 5.3).

The weighting factors for the various terms, $\alpha_1, \alpha_2, \alpha_3, \alpha_4$, and α_5 may, again, be selected statically or dynamically, as needed.

Average Proximity

The Average Proximity is the average of the proximity values between each node in the cluster and the new node.

Average Processor Speed Similarity

The Average Processor Speed Similarity (APSS) is the average of the processor speed similarity between all nodes in the cluster and the new node. An exact match between all the processors results in a value of 1.

Average Memory Size Similarity

The Average Memory Size Similarity is the average memory size similarity between all nodes in the cluster and the new node. An exact match between all the memory sizes results in a value of 1.

Average Hard Disk Space Similarity

The Average Hard Disk Space Similarity is the average of the hard disk space similarity between all nodes in the cluster and the new node. An exact match between all the hard disk available spaces results in a value of 1.

Minimum Bid Value

Recall that there is a minimum bid value, Bid_{min}, that a new node requires to join a cluster otherwise the new node will form a new, single node, cluster. This value is computed as:

$Bid_{min} = \gamma_1 \times Bid_{max}$, where:

$Bid_{max} = \beta_1 + \beta_2 + \beta_3 + \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4$, and

γ_1= another weighting factor which may be selected statically or dynamically.

The maximum possible bid value, Bid_{max}, reflects the best possible match between a new node and a given cluster. It is computed as the sum of the weighting factors of a bid (ignoring α_5 – the weight reflecting cluster need).

Minimum Proximity

There is also a minimum proximity value, $Proximity_{min}$, that a new node requires to join a cluster otherwise the new node should form a new single node cluster. This is necessary to prevent new nodes being added to otherwise suitable but very distant clusters.

$Proximity_{min} = \gamma_2 \times Proximity_{max}$ where:

$Proximity_{max} = \beta_1 + \beta_2 + \beta_3$, and

γ_2= weighting factor which may be selected statically or dynamically.

The maximum proximity value is simply the sum of the weighting factors as the ideal maximum values of Average Bandwidth, Average Latency, and Average Number of Hops are all one.

Weighted Average Bandwidth of all the Clusters

Assuming there are n clusters in the system, the Weighted Average Bandwidth of all Clusters = $\frac{\sum_{i=1}^{n} n_i ABW_i}{\sum_{i=1}^{n} n_i}$, where:

n_i = number of nodes in cluster i, and

ABW_i = Average Bandwidth of cluster i

Weighted Average Latency of all Clusters

Assuming there are n clusters in the system, the Weighted Average Latency of all clusters = $\frac{\sum_{i=1}^{n} n_i AL_i}{\sum_{i=1}^{n} n_i}$, where:

n_i = number of nodes in cluster i, and

AL_i = Average Latency of cluster i

Weighted Average Number of Hops of all Clusters

Assuming there are n clusters in the system, the Weighted Average Number of Hops of all Clusters = $\frac{\sum_{i=1}^{n} n_i AN_i}{\sum_{i=1}^{n} n_i}$, where:

n_i = number of nodes in cluster i, and

AN_i = Average Number of Hops in cluster i

Weighted Average Proximity of all Clusters

Assuming there are n clusters in the system, the Weighted Average Proximity of all clusters = $\frac{\sum_{i=1}^{n} n_i AP_i}{\sum_{i=1}^{n} n_i}$, where:

n_i = number of nodes in cluster i, and

AP_i = Average Proximity in cluster i

Weighted Average Processor Speed Similarity of all Clusters

Assuming there are n clusters in the system, the Weighted Average Processor Speed Similarity of all clusters = $\frac{\sum_{i=1}^{n} n_i APSS_i}{\sum_{i=1}^{n} n_i}$, where:

n_i = number of nodes in cluster i, and

$APSS_i$ = Average Processor Speed Similarity in cluster i

Weighted Average Memory Size Similarity of all Clusters

Assuming there are n clusters in the system, the Weighted Average Memory Size Similarity of all clusters = $\frac{\sum_{i=1}^{n} n_i AMSS_i}{\sum_{i=1}^{n} n_i}$, where:

n_i = number of nodes in cluster i, and

$AMSS_i$ = Average Memory Size Similarity in cluster i

Weighted Average Hard Disk Space Similarity of all Clusters

Assuming there are n clusters in the system, the Weighted Average Hard Disk Space Similarity of all clusters = $\frac{\sum_{i=1}^{n} n_i AHSS_i}{\sum_{i=1}^{n} n_i}$, where:

n_i = number of nodes in cluster i, and

$AHSS_i$ = Average Hard Disk Space Similarity in cluster i

Management Capacity of a Node

The Management Capacity of a Node i= $min(\frac{PS_i}{PS_{min}}, \frac{MS_i}{MS_{min}}, \frac{HS_i}{HS_{min}})$, where:

PS_i = Processor Speed of node i,

PS_{min} = Assumed Minimum Processor Speed required to manage a node in a cluster,

MS_i = Memory Size of node i,

MS_{min} = Assumed Minimum Memory Size required to manage a node in a cluster,

HS_i = available Hard Disk Space on node i, and

HS_{min} = Assumed Minimum Hard Disk Space required to manage a node in a cluster.

4.2 Choosing a cluster to join

After starting, each new node becomes a member of the multicast group and sends a 'Bid Request' message to the multicast group. The node waits for a pre-determined period of time to receive any incoming bid values. After waiting, the node processes the bid messages received. In this process, the highest bid value is calculated from all the bid messages [5].

If the highest bidding value is greater than the minimum bid value and not equal to zero (which means no other clusters or only single node clusters exist), then the node stops being a member of the multicast group and sends a 'Join' message to the leader of the cluster that sent the highest bid value thereby joining that cluster.

If the highest bid value is less than the minimum bid value or equal to zero, the new node becomes a single node cluster.

4.3 Leader Processing

The leader of each cluster is the node in that cluster with the greatest capacity to manage the cluster. The leader of each cluster processes the messages sent to the cluster. Leaders receive two types of messages as already discussed: 1) 'Bid Request' messages and 2) 'Join' messages. A 'Bid Request' message is sent to the primary leaders of all existing clusters when a new node arrives and wants to join the system. Each 'Join' message is sent only to the primary leader of the cluster that sent the highest bid value.

[5] We exclude messages with average proximity = 0 because these messages correspond to those from single node clusters and are of no importance in determining the average proximity.

When a leader receives a 'Bid Request' message, the leader checks whether

- The number of nodes in its cluster is greater than or equal to a threshold (e.g. eighty percent of its capacity),
- The second best node (the node that has the second largest management capacity in the cluster including the leader) in its cluster has a capacity less than a threshold (e.g. 30 percent of its capacity),
- The proximity between the node and the leader is greater than a minimum proximity, and
- The new joining node has a management capacity greater than a threshold (e.g. 30 percent of its capacity).

If all these conditions are met (Cluster Need = 1), then a high bidding value is returned to the requesting node. These conditions ensure that two appropriate clusters (clusters with reasonable capacities) can be formed should cluster splitting later take place.

When a leader receives a 'Join' message, the leader checks whether the new node has a capacity greater than the leader's capacity. If the new node's capacity is greater than the leader's capacity, then:

- The present leader stops being a member of the multicast group,
- The new node becomes the leader of the cluster, and
- The new leader becomes a member of the multicast group

If the new node's capacity is less than or equal to the leader's capacity, the leader checks whether the number of nodes in its cluster is equal to its capacity. If so, then cluster splitting takes place. If the number of nodes in its cluster is less than its capacity, then the new node simply becomes another member of the cluster.

When a new node becomes a member of a cluster, its leader's id is stored by the new node, the leader also stores the new node's information, and the number of nodes in the cluster is incremented by one.

4.4 Cluster Splitting

When the leader of a cluster receives a 'join message' from a new node, the number of nodes in the cluster is equal to the leader's capacity and the new node has capacity less than or equal to the leader's capacity, cluster splitting takes place. In cluster splitting, the present leader becomes the leader of one new cluster and the second best node in the cluster (the node, which has the highest management capacity excluding the present leader) becomes the leader of a second new cluster. The number of nodes assigned to each cluster is determined in proportion to their leaders' capacities as follows:

Total capacity = capacity of leader + capacity of second best node

Number of nodes to be assigned to second cluster = capacity of second best node × capacity of leader / total capacity.

Number of nodes to be assigned to first cluster = capacity of leader - number of nodes to be assigned to second cluster + 1.

The remaining nodes can then be split using either of the following two approaches:

4.4.1 Capacity-based Splitting

After determining the number of nodes to be assigned to each cluster as just described, the nodes (ordered according to capacity) are assigned to each cluster in an alternating fashion. This roughly balances the aggregate management capacity in each of the resulting clusters.

For example, let us assume a "full" cluster having 10 nodes with management capacities as follows:

Node	Capacity
A	10
B	7
C	6
D	5
E	5
F	4
G	4
H	4
I	2
J	2

When a new node, K, with capacity 3 sends a 'join' message, then two clusters are formed. The second cluster will consist of $10 \times 7 / 17 \cong 4.11 = 4$ nodes as follows:

Node	Capacity
B	7
D	5
F	4
H	4

The first cluster will have 10 - 4 + 1 = 7 nodes as follows:

Node	Capacity
A	10
C	6
E	5
G	4
K	3
I	2
J	2

The leader of the second cluster becomes a member of the multicast group and each node in the second cluster updates its leader information.

4.4.2 Proximity-based Splitting

Using proximity-based splitting, after determining the number of nodes to be assigned to each cluster, the primary leader and the second best node (node with second largest management capacity) order the remaining nodes in descending order based on their proximity to each of the leaders using two separate lists. The primary leader then selects

the best node from its list (first node in its list). This selected node is removed from the ordered list of the second best node as well. The second best node then selects the best node from its list. This selected node is also removed from the ordered list of the primary leader. This process is repeated until the second best node has the necessary number of nodes assigned to it. The remaining nodes are assigned to the original leader. This approach produces clusters whose nodes are more closely clustered by proximity.

For example, let us assume a “full” cluster having 10 nodes as follows:

Node	Capacity
A	10
B	7
C	6
D	5
E	5
F	4
G	4
H	4
I	2
J	2

When a new node, K, with capacity 3 sends a ‘join’ message, the number of nodes in the cluster is greater than the capacity of the primary leader. So, cluster splitting takes place.

The number of nodes assigned to the second cluster is $10 \times 7 / 17 \cong 4.11 = 4$ and the number of nodes assigned to the first cluster is therefore $10 - 4 + 1 = 7$.

The ordered list (of the remaining nodes) for the primary leader might be:

Node	Capacity	Proximity
C	6	2.9
H	4	2.75
J	2	2.66
K	3	2.57
E	5	2.55
F	4	2.50
I	2	2.47
G	4	2.3
D	5	2.2

and the ordered list (of the remaining nodes) for the second best node might be:

Node	Capacity	Proximity
E	5	2.87
G	4	2.8
H	4	2.6
K	3	2.54
F	4	2.53
I	2	2.52
D	5	2.3
C	6	2.27
J	7	2.15

Note that the proximity of a node in the cluster to its primary leader is different from the proximity to its second best node because one or more of the bandwidth, the latency, and the number of hops between the node and its primary leader may not be equal to the corresponding bandwidth, latency, and number of hops between the node and its second best node, respectively. In this example, after splitting, the final list of nodes for the primary cluster would be:

Node	Capacity	Proximity
A	10	
C	6	2.9
H	4	2.75
J	2	2.66
F	4	2.50
I	2	2.47
D	5	2.2

and the final list of nodes for the secondary cluster would be:

Node	Capacity	Proximity
B	7	
E	5	2.87
G	4	2.8
K	3	2.54

4.4.3 Improved Proximity-based Splitting

An improvement to proximity-based splitting can be achieved as follows. After the primary leader and the second best node (node with second largest management capacity) choose a node from their respective proximity-based sorted lists, the primary leader and the second best node re-order the remaining nodes in descending order based on their proximity to *all* the nodes in the clusters so far, again using two separate lists.

The difference between this approach and the basic Proximity-based Splitting just described is that after each leader chooses a node, the remaining nodes are re-sorted based on the proximity to all the nodes of the cluster instead of just the leaders. This gives a better sense of the collective proximity and may therefore result in more closely coupled clusters.

4.5 Node Departure

Nodes in the system can leave at any time. If the departing node is the primary leader of a cluster, the primary leader will appoint the second best node (node with second largest management capacity) as the new primary leader and inform the nodes in the cluster to update their new primary leader information before leaving the system. The new primary leader will, of course, join the multicast group.

If the departing node is not the primary leader of a cluster, the node will simply inform the the primary leader of its departure and the primary leader will update its list of nodes in the cluster (removing the departing node from its list).

4.6 Cluster Merging

There are two ways in which cluster merging may take place:

- Each cluster has an associated "minimum" capacity, which is some X times smaller than the maximum capacity of the cluster, reflecting the fact that the cluster's capacity is well below its maximum capacity. When a cluster reaches its minimum capacity, it should merge with another cluster. Doing this means that the number of clusters in the entire system will be reduced resulting in a reduction of the number of messages sent in the system and also providing, on average, a larger set of resources in each cluster which is, intuitively, more useful.

  When a cluster's minimum capacity is reached, on the departure (or failure) of a node, the leader sends a 'Request to Merge' message specifying its characteristics (the number of nodes, node id of the primary leader etc. in the cluster) to the multicast group. Then, the leader of every other cluster will reply by sending a response containing its characteristics. The leader of the cluster that has reached its minimum capacity selects the best available cluster to join (the one that has a small number of nodes relative to all the available clusters and that is reasonably nearby).

- The leader of each cluster also periodically sends a message consisting of the number of nodes in its cluster and its address to the multicast group. Based on this message, a leader from another cluster may choose to offer the sending leader the option to merge with it to form a larger (and potentially more useful) cluster.

In both situations, the existence of a suitable cluster is required to effect a merge. Let X be a cluster and D be the difference between X's maximum capacity and the number of nodes in X. For a cluster Y, X is a suitable cluster to join with only if D is greater than the number of nodes in Y.

4.7 Node Failure

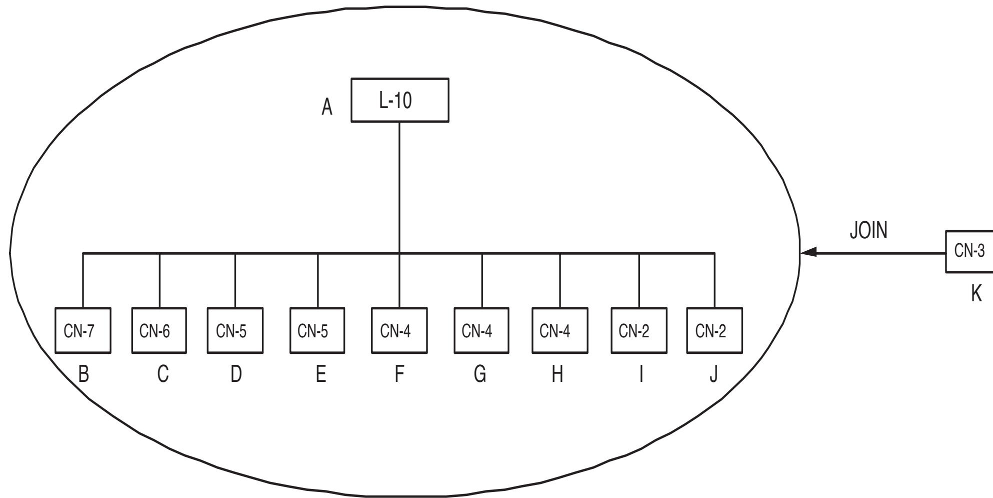
Nodes in the system can fail at any time. If the failed node is the primary leader of a cluster, all the nodes in the cluster will be forced to re-join the system as new nodes since they have no knowledge of the other members of the cluster they are in and thus cannot elect a new leader independently. The nodes from the cluster whose leader has failed will therefore be distributed over the surviving clusters as a result. If the failed node is not the primary leader of the cluster, the primary leader will simply update its list of nodes in its cluster (removing the failed node from its list).

4.8 Use of Resulting Clusters

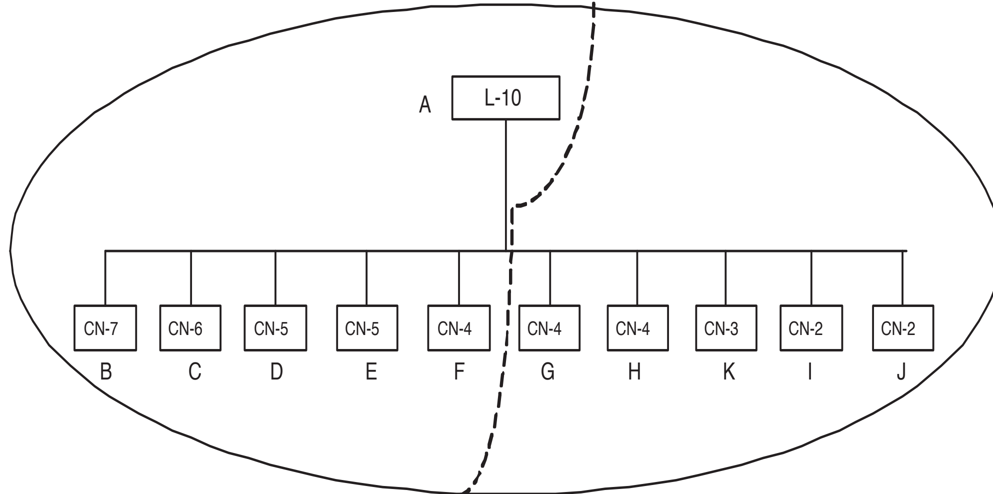
The end result of ongoing cluster construction, at any point in time, will be a collection of dynamically constructed and geographically dispersed clusters, each composed of machines able to effectively inter-communicate and inter-operate with one another. These clusters will then be available for use by any user as needed.

A user, who wants access to computing resources, can run a client program that sends a "Resource Request" message to the multicast group. Each cluster's leader can then

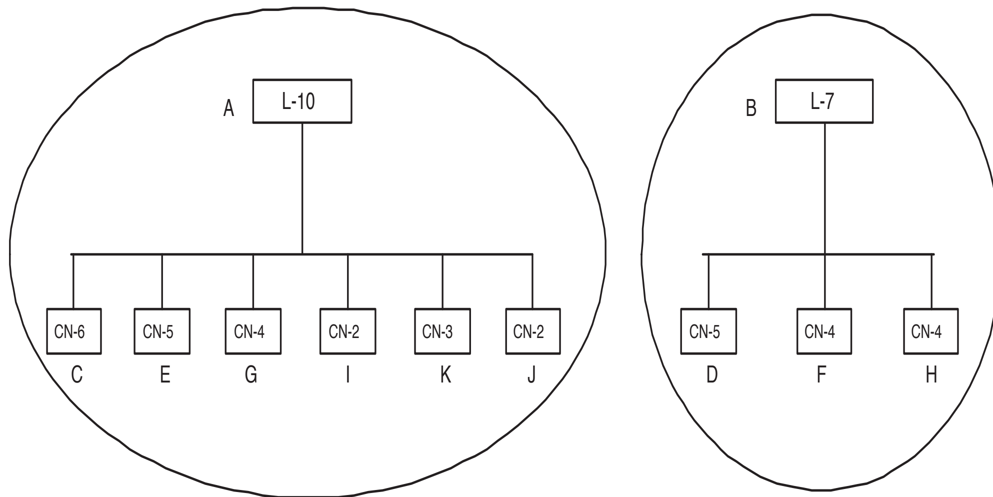
respond by sending a reply enumerating the resources available in its cluster as well as, perhaps, other information such as current load and any pending resource reservations. The client program can then select a suitable available cluster and send a job request to the corresponding cluster(s) that offer the resources it has selected. A resource broker instead of the client program itself could also do this job, thereby making the use of the system client-program transparent. The design of such a resource broker, however, is beyond the scope of this thesis.



(a) Arrival of a new node



(b) Cluster's maximum capacity has been exceeded



(c) Cluster Splitting

Figure 4.2: Cluster Splitting

Chapter 5

Implementation And Simulation Results

In this chapter, I present the implementation of my architecture. Additionally, the effectiveness of the architecture described is assessed using simulation.

I implemented the code and associated data structures using Java. None of the code is particularly performance sensitive so there was no need to use a more efficient programming language such as C. Further, Java provides many easy-to-use facilities for developing and manipulating the data structures needed to implement my system. Finally, Java is compatible with the SSJ simulation system [27] which I chose to use for the simulation work.

I chose SSJ for the simulation of my architecture because it offers general-purpose facilities for simulation programming and is also free for use. I used SSJ to develop a simple discrete-event simulation used to assess the effectiveness of my system under various conditions. SSJ is a Java library of classes that provide suitable tools for stochastic sim-

ulation. Being Java based, it also provides the generality and power of a general-purpose programming language. It was relatively straight forward to integrate my implemented algorithms (written in Java) into the simulation system.

I also used BRITE [32], a widely used Internet topology generation tool for generating realistic network structures which are used in the simulations. BRITE generates accurate synthetic topologies while taking into consideration various aspects of the actual Internet topology such as link bandwidths, hierarchical structure, node degree distribution, etc..

5.1 Implementation of the architecture

In this section, I briefly describe the key data structures used and sketch the fundamental processing done by the important algorithms that together constitute my implementation.

5.1.1 Key Data Structures

There are two types of nodes in the system: active nodes and passive nodes. Active nodes are either new nodes joining the system or the primary leaders of clusters. These are the nodes that are *active* in the cluster formation process. Passive nodes are all those nodes that already belong to a cluster, not including the primary leader of the cluster. Naturally, nodes may move between being active and passive. For example this may happen when a node has finished joining a cluster or when there is a change of leaders. Each active node in the network also has an associated java thread (`simprocess` in terms of SSJ [27]) and is implemented as a Java object containing the following fields:

- node_id: unique identifier for the node

- processor_speed_in_MHZ: processor speed of the node in MHz
- memory_in_MB: memory size of the node in MB
- storage_space_in_GB: available hard disk space on the node in GB
- capacity: management capacity of the node
- nodes: array for storing all the nodes in the cluster lead by this active node (passive nodes and primary leader). This array is empty if the corresponding node is a new node rather than a leader.
- multicast_messages: queue for storing all the messages sent to the multicast group

Each passive node is implemented as a Java object containing the following fields:

- node_id: unique identifier for the node
- processor_speed_in_MHZ: processor speed of the node in MHz
- memory_in_MB: memory size of the node in MB
- storage_space_in_GB: hard disk space of the node in GB
- capacity: management capacity of the node
- primary_leader_id: unique identifier for the primary leader of the cluster

Each active node will be a member of the multicast group and all the messages sent to the multicast group will thus be delivered to all the active nodes. There are 3 types of messages: bid_request_messages, bid_response_messages, and join_messages. Recall that

a bid_request_message is sent by a new node wishing to join the system to the multicast group and is delivered to all the primary leaders of the clusters. A bid_response_message is sent by the leaders to requesting new nodes in response to a received bid_request_message. Finally, a bid_request_message is sent from a new node to the specific leader whose cluster the new node wants to join. All of these messages must also be implemented (as Java Objects).

A bid_request_message object contains the following fields:

- message_id: unique identifier identifying the type of the message (value=2)
- sender_id: unique identifier of the node that sent the message
- processor_speed_in_MHZ: processor speed of the node that sent the message
- memory_in_MB: memory size of the node that sent the message
- storage_space_in_GB: hard disk space of the node that sent the message
- capacity: capacity of the node that sent the message

A bid_response_message object contains the following fields:

- message_id: unique identifier identifying the type of the message (value=1)
- sender_id: unique identifier if the node that sent the message
- bid_value: bid value

Finally, a join_message object contains the following fields:

- message_id: unique identifier identifying the type of the message (value=3)

- sender_id: unique identifier of the node that sent the message
- processor_speed_in_MHZ [3]: processor speed of the node that sent the message
- memory_in_MB[3] : memory size of the node that sent the message
- storage_space_in_GB[3]: hard disk space of the node that sent the message
- capacity[3]: capacity of the node that sent the message

Using these various objects, the algorithmic processes required to do cluster formation can be implemented. These processes are now described at a high level using simple flow charts prior to discussing the simulation and presenting the results.

5.1.2 Joining of a node

New nodes join the system at unpredictable times. When a new node wants to join the grid, it sends a bid_request_message to the multicast group. After waiting some period of time to allow responses to arrive, the new node processes all the bid_response_messages it received. If the highest bid value is greater than zero and minimum bid value, a join_message is sent to the highest bidder. This is shown in Figure 5.1.

5.1.3 Primary Leader Message Processing

Each primary leader of a cluster receives two types of messages: bid_request_messages and join_messages. When the primary leader receives a bid_request_message from a new node through the multicast group, the primary leader calculates the bid value based

[3]Resent for convenience.

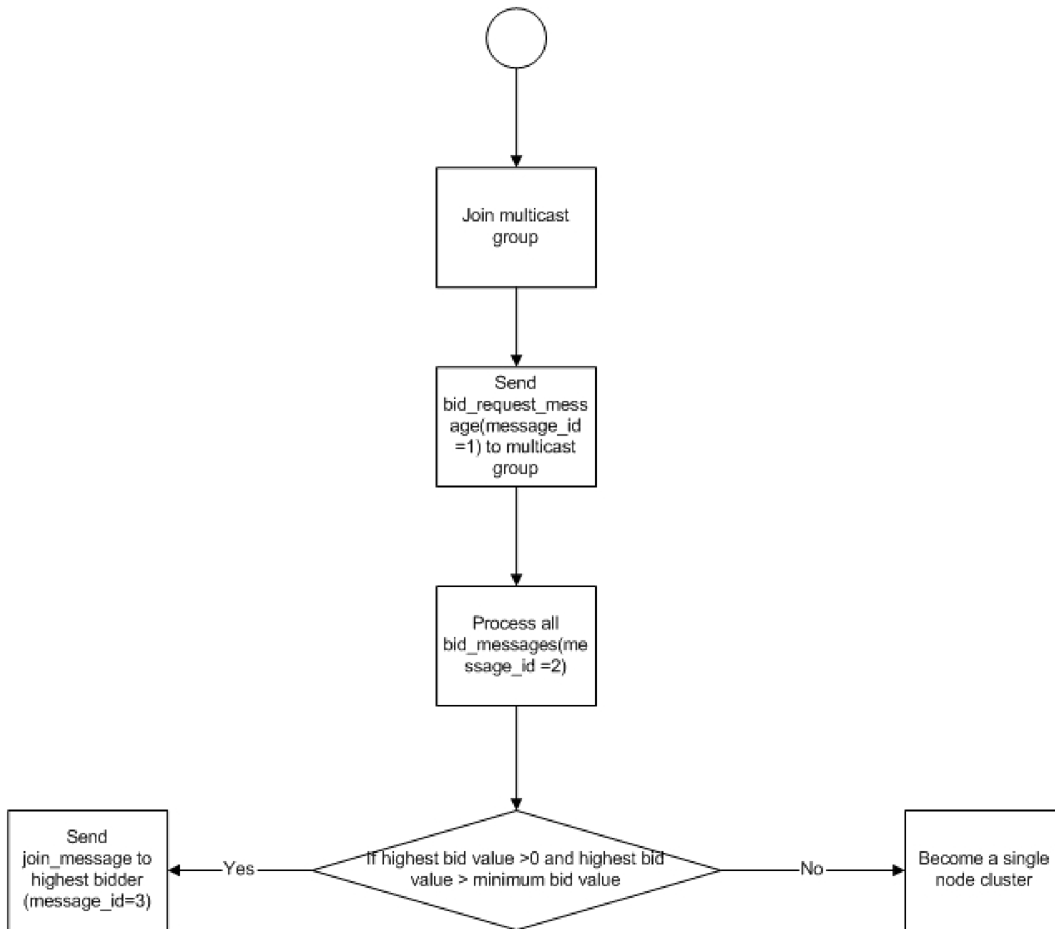


Figure 5.1: Flowchart for joining of a node

on the information about the new node and sends a bid_response_message to the new node. When the primary leader receives a join_message from a new node, the new node becomes a member of the cluster. If the new node's capacity is greater than the primary leader's capacity, the new node becomes the primary leader of the cluster. If the number of nodes is equal to the primary leader's capacity, cluster splitting takes place resulting in the formation of two clusters with their respective primary leaders. This is shown in Figure 5.2.

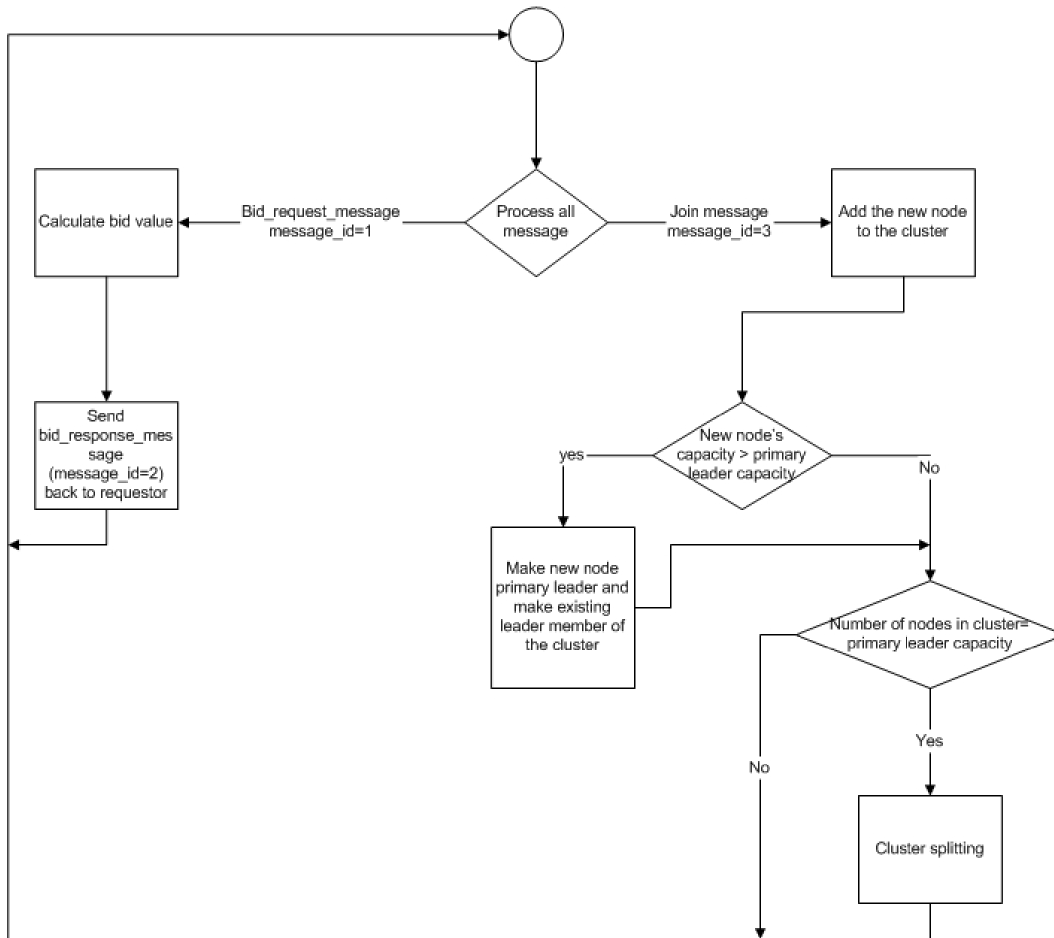


Figure 5.2: Flowchart for Primary Leader Message Processing

5.2 Simulation Setup

In this section I describe how I used the BRITE topology generator to create a model Internet structure for my simulations, how I assigned device characteristics (e.g. CPU speed, memory capacity, etc.) to each node in the model, and what simulation parameters and assumptions I used.

5.2.1 Network Topology Generation Using BRITE

It is almost impossible to capture the true characteristics of the Internet because of its dynamic nature and enormous size. It is possible, however, to model the general (high-level) characteristics of the Internet to obtain useful simulation results. I have chosen BRITE's top-down hierarchical topology model to simulate the Internet's structure for my simulations as shown in Figure 5.3. This reflects the large scale structure in the real Internet. At the Autonomous System (AS) level, each node represents an autonomous system consisting of a set of routers under a single administrative domain such as a university or a local Internet Service Provider (ISP). At the router level, each node represents a router that falls under a single administrative domain (autonomous system) and connects host machines in that domain. Interconnections between ASs are generated between selected routers within each AS.

BRITE offers two types of node placement while generating the network: uniform random or heavy tailed. In random node placement, each node is placed somewhere on the plane with equal probability. In heavy tailed node placement, the plane is divided into number of squares, and each square is assigned a number of nodes according to a heavily tailed distribution. Then the prescribed number of nodes are placed uniformly randomly in each square. I have chosen the heavy-tailed distribution for node placement and assigning bandwidth between AS (autonomous system) level nodes and router level nodes as there is a generally accepted assumption that the current Internet traffic and topologies exhibit are heavily-tailed [31].

Since all the intra-domain nodes (router level nodes) are geographically close and can

therefore potentially offer high bandwidth, for the purpose of my simulations, I have assumed the bandwidth between routers within an AS is ten times that of the bandwidth between routers in different ASs. I used the following bandwidth distributions to generate the network topology in BRITE for my simulations:

- For router-level nodes, a heavy-tailed distribution with a minimum value 100 Mbps and a maximum value of 1000 Mbps
- For AS-level nodes, a heavy-tailed distribution with a minimum value 10 Mbps and a maximum value of 100 Mbps

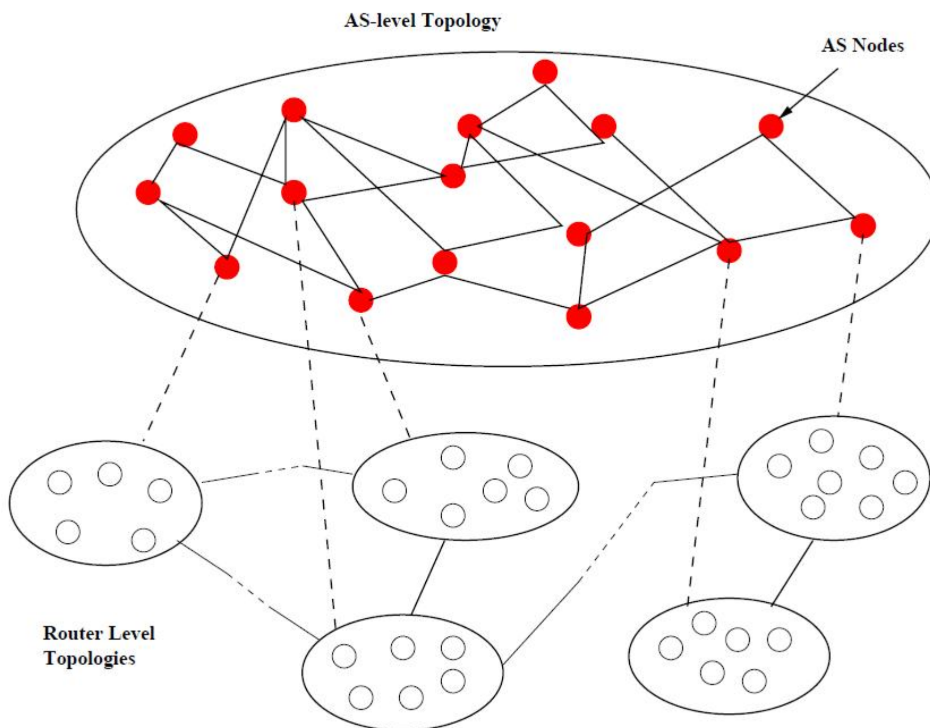


Figure 5.3: Top-down hierarchical topology (adapted from [31])

In the real world, great variance might be seen in the bandwidth links but the frequency of occurrence of very high and low bandwidths is likely to be low. This is why I chose to

stay within 100 Mbps and 1000 Mbps for router-level nodes and 10 Mbps and 100 Mbps for AS-level nodes.

After generating the network structure using BRITE, I calculated the shortest path between all pairs of nodes using Floyd's algorithm. Three two-dimensional arrays are used to store the bandwidth, latency and number of hops between all pairs of nodes for easy use during simulation.

5.2.2 Assigning Hard Disk Space, Processor Speeds, and Memory Sizes to Each Node

For the purpose of my simulation experiments, I assume that each node in the system will have one of the following values for processor speed, memory size and available hard disk capacity:

- Assumed processor speeds in MHz=[500,1000,1500,1700,1800,2200,2500,3000,3500]
- Assumed memory sizes in MB=[32,64,128,256,512,1024,2048,4096,8192]
- Assumed hard disk spaces in GB=[10,20,40,60,80,100,120,160,200]

Three independent, uniformly distributed, random values are generated for each node, with values between 1 and 9. These random values are used as the indexes into the lists of the assumed processor speeds, memory sizes and hard disk spaces for assigning processor speed, memory size and hard disk capacity, respectively, to each node. That is:

- Processor speed of node=Assumed processor Speeds[RandomValue1]

- Memory size of node=Assumed memory sizes[RandomValue2]

- Hard disk space of node=Assumed hard disk spaces[RandomValue3]

I also assume that an average node (in terms of computing power) will be able to effectively manage some N=200 nodes[1]. Based on this assumption, we derived the assumed minimum processor speed, minimum memory size, and minimum hard disk space required to manage a node in a cluster as follows:

$$\text{Assumed minimum processor speed} = \frac{sumofallassumedprocessorspeeds}{200 \times numberofvaluesinthelistofassumedprocessorspeeds}$$

$$= \frac{17700}{200 \times 9} = 9.83$$

$$\text{Assumed minimum memory size} = \frac{sumofallassumedmemorysizes}{200 \times numberofvaluesinthelistofassumedmemorysizes}$$

$$= \frac{16352}{200 \times 9} = 9.1$$

$$\text{Assumed minimum hard disk space} = \frac{sumofallassumedharddiskspaces}{200 \times numberofvaluesinthelistofassumedharddiskspaces}$$

$$= \frac{790}{200 \times 9} = 0.44$$

These values were used during the simulation.

5.2.3 Simulation Parameters

The base parameters used for my simulations are shown in Table 5.1. Unless mentioned specifically in the following subsections these parameters were in effect for all simulations.

I also assumed that the threshold on the number of nodes in a cluster for determining cluster need is 0.8 times the capacity of the primary leader and the threshold on the capacity of the second best node in a cluster and new nodes for cluster need is 0.3 times

[1] This value for N was selected without extensive study but is based on the expected capacity of similar machines acting as various types of servers in other applications.

Parameter	Value
Bandwidth Weighting factor (β_1)	1
Latency Weighting factor (β_2)	1
Number of hops weighting factor (β_3)	1
Proximity weighting factor (α_1)	1
Processor speed similarity Weighting factor (α_2)	1
Memory size similarity weighting factor (α_3)	1
Hard disk space similarity weighting factor (α_4)	1
Cluster need weighting factor (α_5)	1
Minimum bid value	0
Type of Splitting	Capacity-based
Number of Nodes	50 AS $\times$ 50 Router (2500 nodes)
Number of edges of each node	2

Table 5.1: Default Parameters

the capacity of the primary leader. I will be varying some of the parameters listed in Table 5.1 to assess the impact on the behaviour of my system. Based on the default parameters I ran my simulations and produced the basic clusters described in Table 5.2:

Finally, for all simulations, I assume that new nodes attempt to join the system every 10 units of time and that the timeout period for receiving responses to bid_request_messages is 5 time units.

5.3 Simulation Results

In this Section I describe the results of my simulation experiments. Three broad sets of experiments were done and are reported on here. The first adjusted the various weighting factors to gain confidence in the simulation model and then to confirm the behaviour of

Cluster No	Cluster Size	Cluster Capacity
1	217	305
2	267	356
3	241	305
4	302	356
5	276	356
6	194	356
7	249	356
8	247	305
9	168	305
10	167	254
11	172	305

Table 5.2: Default Discovered Clusters

the general clustering approach. The second set of experiments considered the impact of changes in the available resources (both network and compute nodes) and the third compared the different splitting algorithms described earlier.

5.3.1 Controlling the Weighting Factors

I began by experimenting with changes to the weighting factors. As will be seen, the effects of changing the weights are as would be expected. This tells us that the general technique can be customized through variation of the weights to achieve desired characteristics in the resulting clusters. For example, if we are considering application of the technique to generate clusters for parameter-sweep experiments where communication between nodes is not required, the weights for the network characteristics in the clustering formula can be decreased, thereby placing greater emphasis on the uniformity of the

β_1	Weighted Average Bandwidth of all clusters
1	19.367
2	19.215
3	20.181
4	20.114
5	21.757
7	23.226
12	24.851
15	25.335
20	29.255
50	31.082
100	30.867
1000000	31.659

Table 5.3: Varying the bandwidth weighting factor

nodes in the clusters formed.

5.3.1.1 Controlling the Weighting Factors of Network Characteristics

I first varied the weighting factors for the network characteristics (bandwidth, latency and number of hops) to determine their effects on the clusters formed.

I started by varying the bandwidth weighting factor to see its effect on the weighted average bandwidth of all clusters. The results are shown in Table 5.3 and in Figure 5.4.

As β_1 is increased from 1 to 4, the weighted average bandwidth of all clusters changes slightly. As β_1 is increased from 5 to 50, the weighted average bandwidth of all clusters increases significantly. As β_1 is increased from 50 to 1000000, the weighted average bandwidth of all clusters again changes slightly. This plateauing effect is expected since

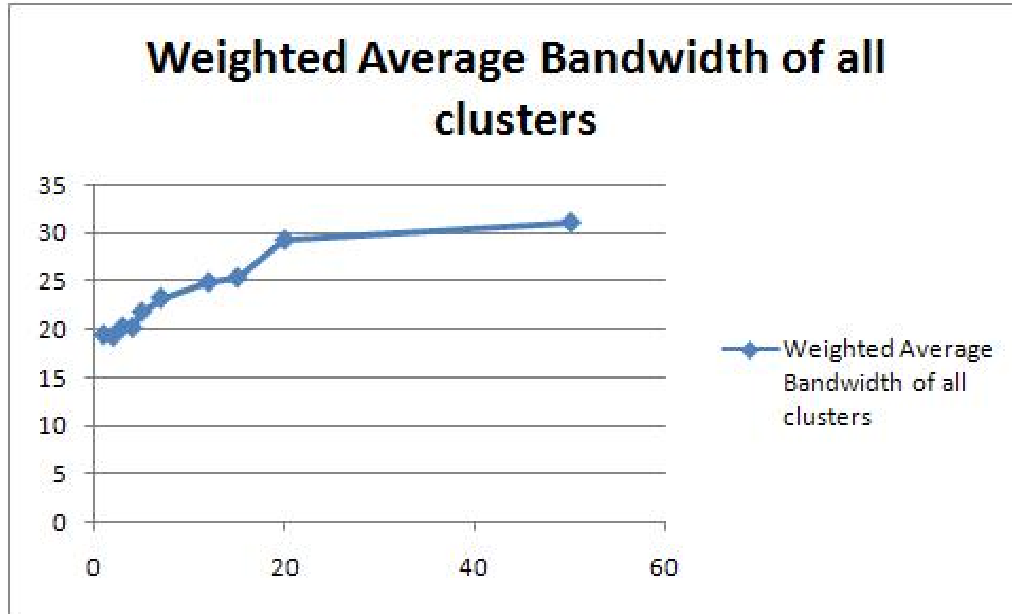


Figure 5.4: β_1 vs Weighted Average Bandwidth of all clusters

the characteristics of the network restrict what is ultimately possible in terms of clusters formed[2]. The weighted average bandwidth of all clusters can be increased by increasing β_1. If it is important for clusters to have high inter-node bandwidth, a weight of $\beta_1 = 50$ would be effective given the other simulation parameters.

Next, I varied the latency weighting factor to see its effect on the weighted average latency of all clusters. The results are shown in Table 5.4 and in Figure 5.5.

As β_2 is increased from 1 to 100, the weighted average latency of all clusters changes slightly. As β_2 is increased from 100 to 400, the weighted average latency of all clusters decreases slightly. As β_2 is increased from 400 to 10000, the weighted average latency of all clusters changes only slightly. The weighted average latency of all clusters can be decreased by increasing β_2. If it is important for clusters to have low inter-node latency,

[2]This behaviour will be seen in other results presented but, for succinctness, will not be discussed again.

β_2	Weighted Average Latency of all clusters
1	16.808
100	16.269
200	15.586
300	14.534
400	14.055
500	14.058
10000	14.037

Table 5.4: Varying the latency weighting factor

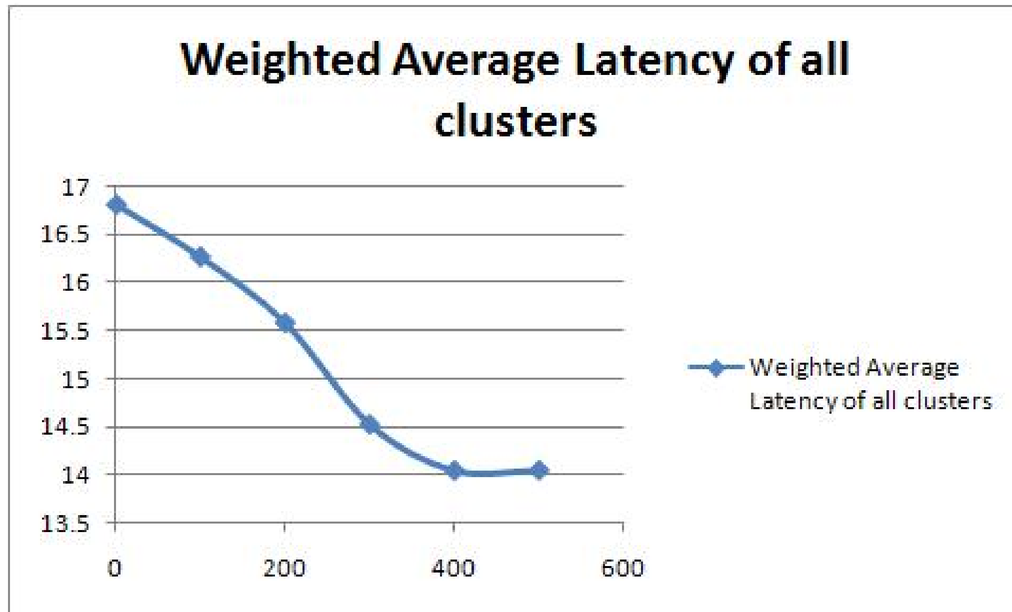


Figure 5.5: β_2 vs Weighted Average Latency of all clusters

a weight of $\beta_2 = 400$ would be effective given the other simulation parameters.

I also varied the number of hops weighting factor to see its effect on the weighted average number of hops of all clusters. The results are shown in Table 5.5 and in Figure 5.6.

β_3	Weighted Average Number of Hops of all clusters
1	12.192
2	12.056
3	11.688
5	10.074
7	10.113
9	10.280
10	10.070
100	10.026
10000	10.002

Table 5.5: Varying the number of hops weighting factor

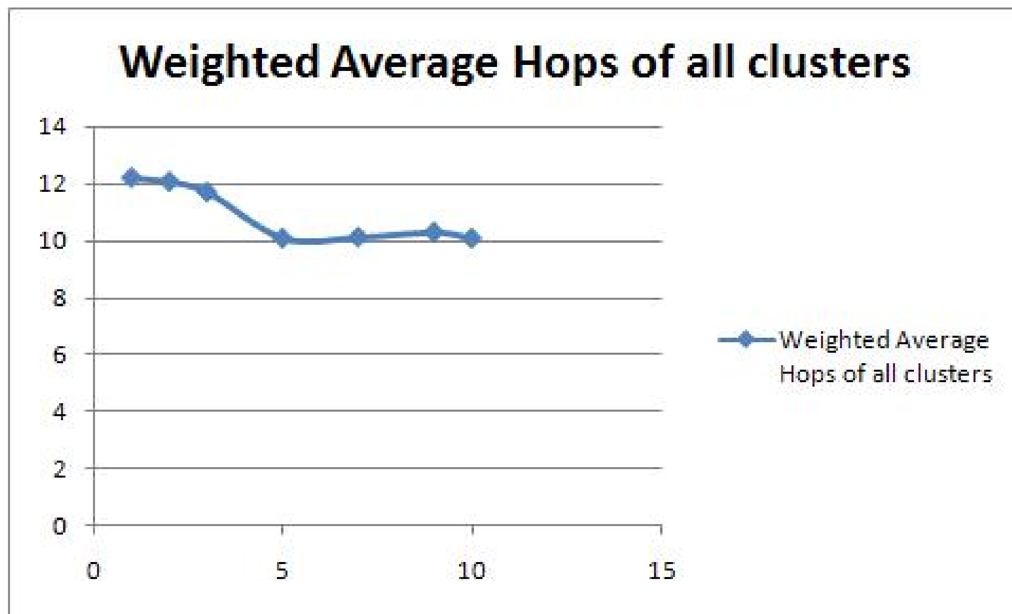


Figure 5.6: β_3 vs Weighted Average Hops of all clusters

As β_3 is increased from 1 to 2, the weighted average number of hops of all clusters changes slightly. As β_3 is increased from 2 to 5, the weighted average number of hops of all clusters decreases slightly. As β_3 is increased from 5 to 10000, the weighted average

α_1	Weighted Average Proximity of all clusters
1	0.0231
3	0.0285
5	0.0392
7	0.0383
9	0.0396
10	0.0397
100	0.0398
10000	0.0398

Table 5.6: Varying the proximity weighting factor

number of hops of all clusters changes slightly. The weighted average number of hops of all clusters can be decreased by increasing β_3. If it is important for clusters to have a low number of inter-node hops, a weight of $\beta_3 = 5$ would be effective given the other simulation parameters.

Finally, I varied the proximity weighting factor to see its effect on the weighted average proximity of all clusters. The results are shown in Table 5.6 and in Figure 5.7.

As α_1 is increased from 1 to 5, the weighted average proximity of all clusters increases significantly. As α_1 is increased from 5 to 10000, the weighted average proximity of all clusters changes slightly. The weighted average proximity of all clusters can be increased by increasing α_1. If it is important for clusters to have high inter-node proximity, a weight of $\alpha_1 = 5$ would be effective given the other simulation parameters.

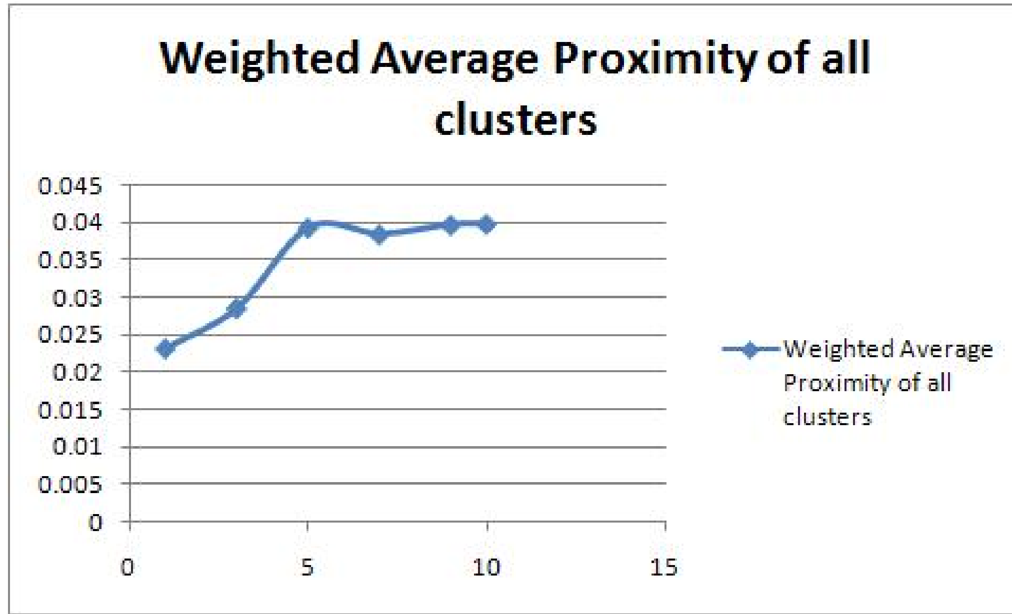


Figure 5.7: α_1 vs Weighted Average Proximity of all clusters

5.3.1.2 Controlling the Weighting Factors of Node Characteristics

After experimenting with varying the weights of the network characteristics, I did experiments that varied the weighting factors for the node characteristics (processor speed similarity, memory capacity similarity, etc.) to determine their effects on the clusters formed.

I started by varying the processor speed similarity weighting factor to see its effect on the weighted average processor speed similarity of all clusters. The results are shown in Table 5.7 and in Figure 5.8.

As α_2 is increased from 1 to 9, the weighted average processor speed similarity of all clusters increases significantly. As α_2 is increased from 9 to 10000, the weighted average processor speed similarity of all clusters changes only slightly. As expected, the weighted average processor speed similarity of all clusters can be increased by increasing α_2. If it

α_2	Weighted Average Processor Speed Similarity of all clusters
1	0.751
3	0.855
5	0.861
7	0.857
9	0.871
10	0.871
100	0.865
10000	0.865

Table 5.7: Varying the processor speed similarity weighting factor

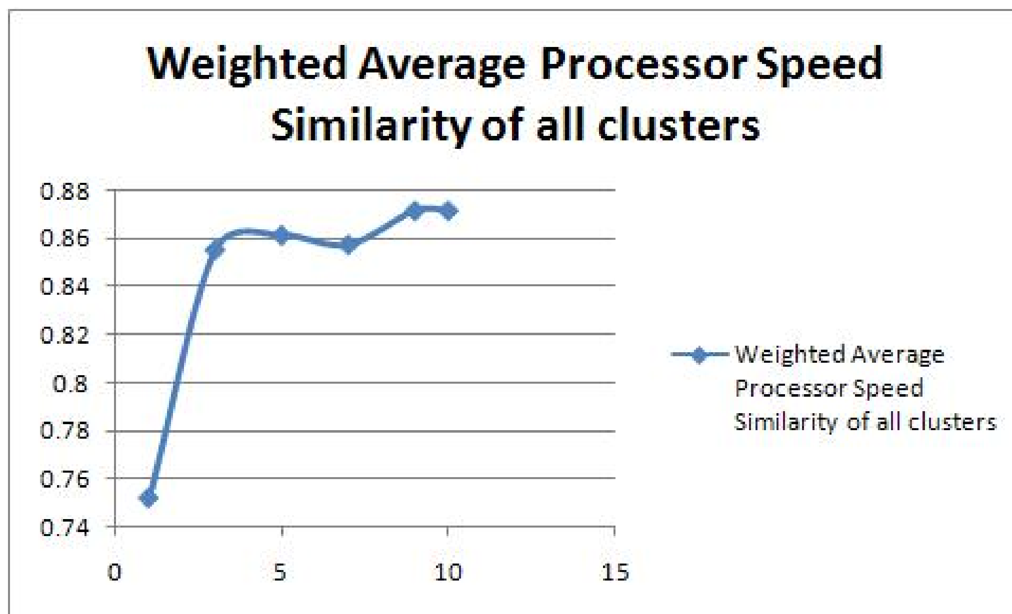


Figure 5.8: α_2 vs Weighted Average Processor Speed Similarity of all clusters

is important for clusters to have high inter-node processor speed similarity, a weight of $\alpha_2 = 9$ would be effective given the other simulation parameters.

I then varied the memory size similarity weighting factor to determine its effect on the weighted average memory size similarity of all clusters. The results are shown in

α_3	Weighted Average Memory Size Similarity of all clusters
1	0.442
3	0.667
5	0.656
7	0.677
9	0.678
10	0.678
100	0.679
10000	0.679

Table 5.8: Varying the memory size similarity weighting factor

Table 5.8 and in Figure 5.9.

As α_3 is increased from 1 to 3, the weighted average memory size similarity of all clusters increases significantly. As α_3 is increased from 3 to 10000, the weighted average memory size similarity of all clusters changes slightly. As expected, the weighted average memory size similarity of all clusters can be increased by increasing α_3. If it is important for clusters to have high inter-node memory size similarity, a weight of $\alpha_3 = 3$ would be effective given the other simulation parameters.

Finally, I varied the hard disk space similarity weighting factor to see its effect on the weighted average hard disk space similarity of all clusters. The results are shown in Table 5.9 and in Figure 5.10.

As α_4 is increased from 1 to 5, the weighted average hard disk space similarity of all clusters increases significantly. As α_4 is increased from 5 to 10000, the weighted average hard disk space similarity of all clusters changes only slightly. The weighted average hard disk space similarity of all clusters can be increased by increasing α_4. If it is important

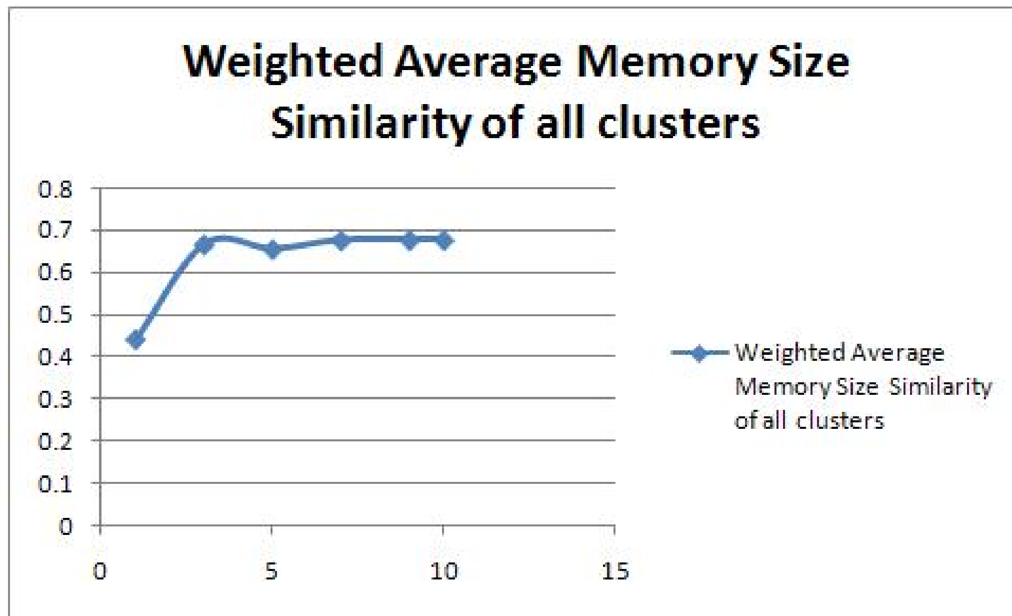


Figure 5.9: α_3 vs Weighted Average Memory Size Similarity of all clusters

for clusters to have high inter-node disk space similarity, a weight of $\alpha_4 = 5$ would be effective given the other simulation parameters.

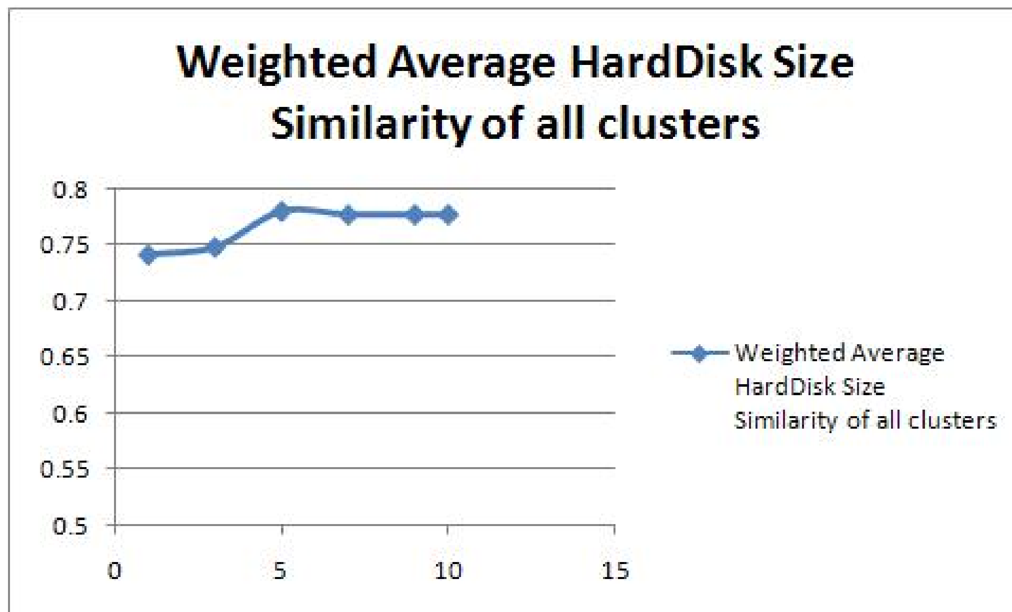


Figure 5.10: α_4 vs Weighted Average Hard Disk Space Similarity of all clusters

α_4	Weighted Average Hard Disk Space Similarity of all clusters
1	0.741
3	0.748
5	0.779
7	0.776
9	0.776
10	0.777
100	0.758
10000	0.772

Table 5.9: Varying the hard disk similarity weighting factor

5.3.2 Impact of Available Resources

In my second group of experiments I experimented by varying the available network and computing resources to determine the impact on clustering effectiveness.

5.3.2.1 Connectedness of Nodes

I varied the number of edges between each node (both AS-level nodes and router-level nodes) to see its effect on the weighted average proximity, weighted average bandwidth, weighted average latency, and weighted average number of hops in all created clusters. The effect on the weighted average proximity of all clusters as the number of edges associated with each node is increased is shown in Table 5.10 and in Figure 5.11

As the number of edges associated with each node is increased (i.e. the number of hops between nodes of the generated network is decreased), the weighted average proximity of

Number of Edges of Each node	Weighted Average Proximity of all clusters
1	0.0211
2	0.0230
3	0.0259
4	0.0272
5	0.0284
6	0.0303
7	0.0316

Table 5.10: Number of Edges of Each node vs Weighted average Proximity of all clusters

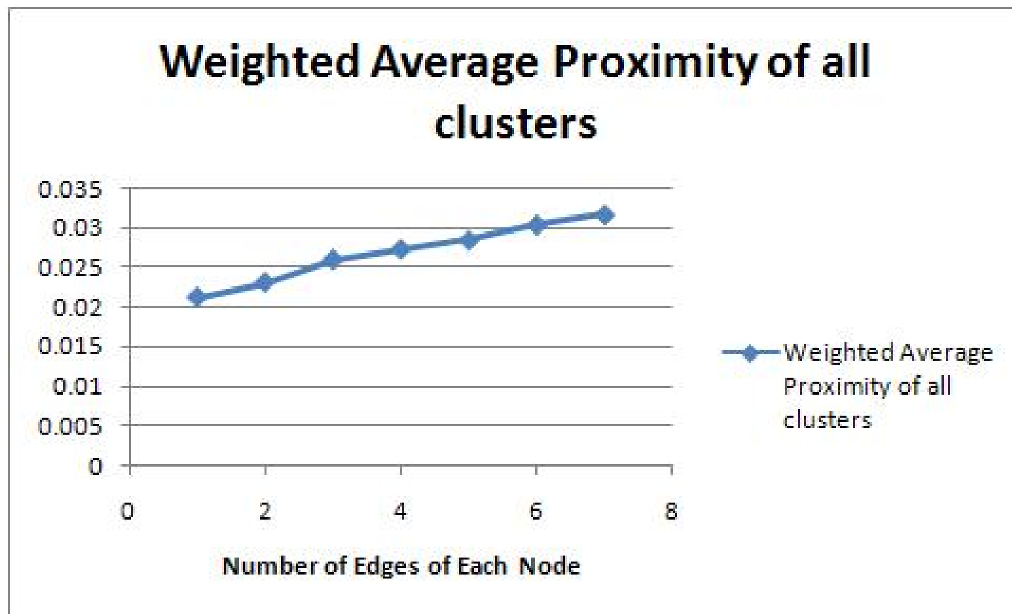


Figure 5.11: Number of Edges of Each node vs Weighted average Proximity of all clusters

all clusters increases. Thus, the lesser the number of hops between nodes of the network, the more likely my system is to form clusters which are closer in terms of proximity.

The effect on the weighted average bandwidth of all clusters as the number of edges associated with each node is increased is shown in Table 5.11 and in Figure 5.12

Number of Edges of Each node	Weighted Average Bandwidth of all clusters
1	18.200
2	19.367
3	19.532
4	19.932
5	20.681
6	21.293
7	20.170

Table 5.11: Number of Edges of Each node vs Weighted average bandwidth of all clusters

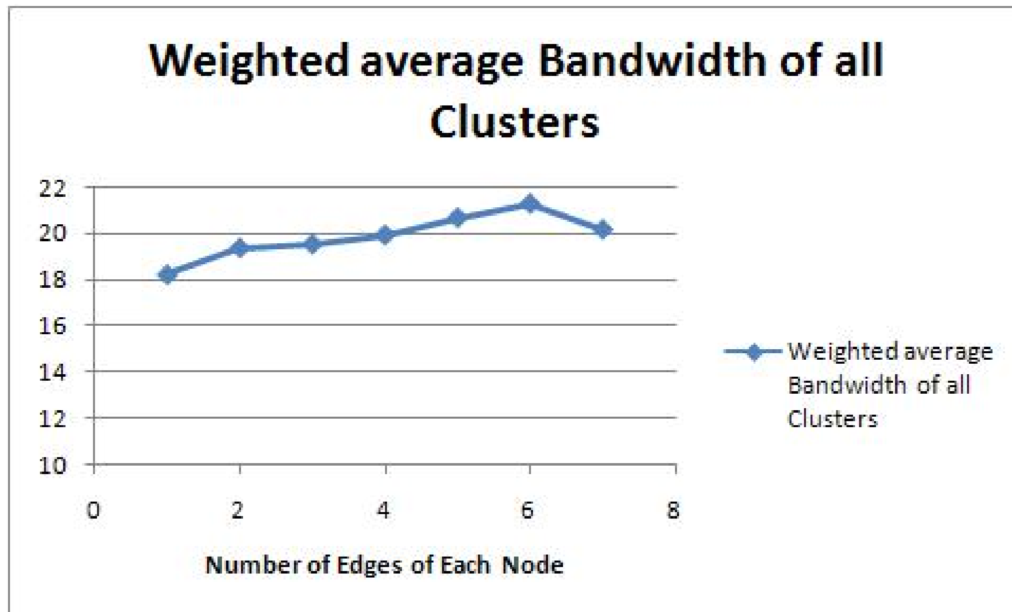


Figure 5.12: Number of Edges of Each node vs Weighted average Bandwidth of all clusters

As the number of edges associated with each node is increased (i.e. the number of hops between nodes of the generated network is decreased), the weighted average bandwidth of all clusters increases. As the number of hops between nodes of the network decreases, my system forms clusters which have higher average inter-node bandwidth.

Number of Edges of Each node	Weighted Average Latency of all clusters
1	40.108
2	16.808
3	11.702
4	10.117
5	8.612
6	7.953
7	7.362

Table 5.12: Number of Edges of Each node vs Weighted average latency of all clusters

The effect on the weighted average latency of all clusters as the number of edges associated with each node is increased is shown in Table 5.12 and in Figure 5.13

As the number of edges associated with each node is increased, the weighted average latency of all clusters decreases.

The effect on the weighted average number of hops of all clusters as the number of edges associated with each node is increased is shown in Table 5.13 and in Figure 5.14

As the number of edges of each node is increased, the weighted average number of hops in all clusters decreases.

5.3.2.2 Number of Nodes

I also varied the number of nodes joining the system to see its effect on the weighted average proximity, weighted average bandwidth, weighted average latency,weighted average number of hops, weighted average processor speed similarity, weighted average memory size similarity, and weighted average hard disk space similarity of all clusters. Increasing the rate at which nodes join the system also increases the overall number of nodes in the

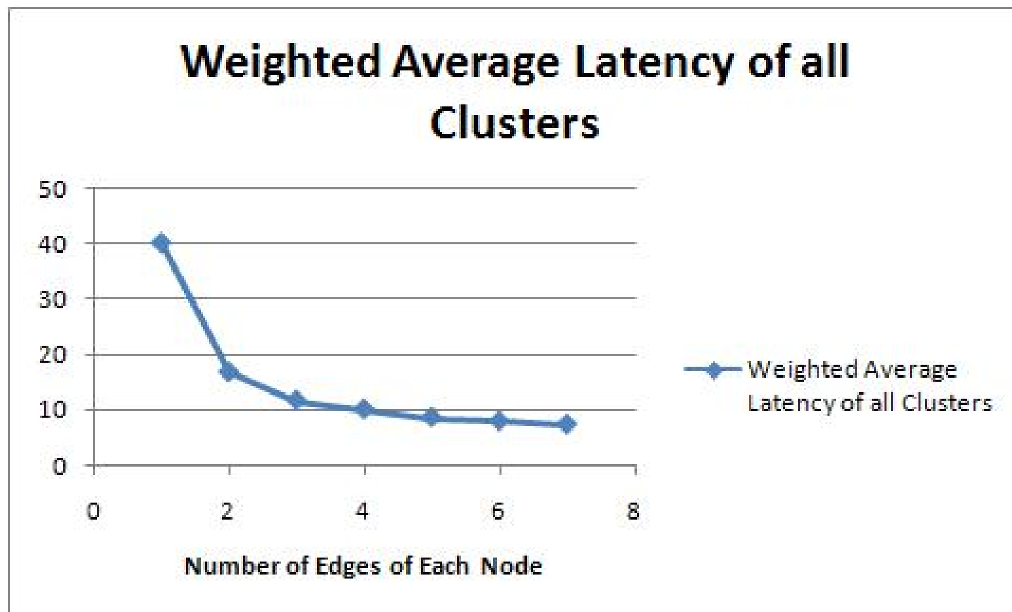


Figure 5.13: Number of Edges of Each node vs Weighted average Latency of all clusters

Number of Edges of Each node	Weighted Average Hops of all clusters
1	22.778
2	12.192
3	9.939
4	8.985
5	8.174
6	7.733
7	7.358

Table 5.13: Number of Edges of Each node vs Weighted average number of hops of all clusters

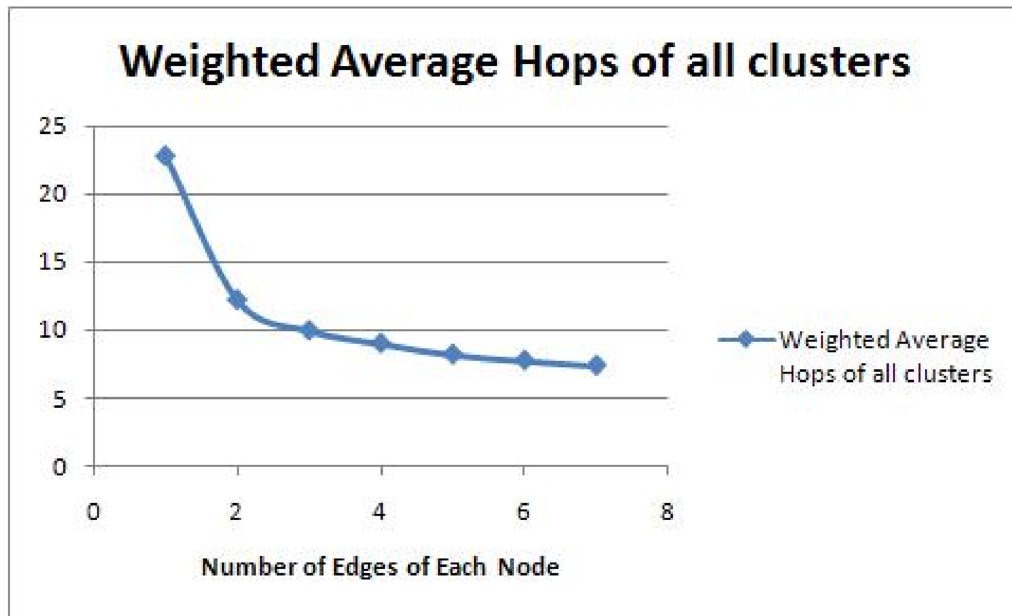


Figure 5.14: Number of Edges of Each node vs Weighted average Hops of all clusters

system and hence the node density in the network.

The effect on the weighted average bandwidth of all clusters as the number of nodes joining the system is increased is shown in Table 5.14 and in Figure 5.15

As the number of nodes is increased within the system, the weighted average bandwidth of all clusters decreases slightly in most cases. This is because as more nodes join the system, the number of hops between nodes of the network increases resulting in each cluster potentially choosing nodes that are farther away and that have less bandwidth to the other cluster nodes.

The effect on the weighted average latency of all clusters as the number of nodes joining the system is increased is shown in Table 5.15 and in Figure 5.16

As the number of nodes is increased in the system, the weighted average latency of all clusters increases slightly. This is because as more nodes join the system, the number of

Total number of nodes	Weighted Average Bandwidth of all clusters
400	22.492
625	19.754
900	21.050
1225	19.687
1600	18.007
2025	18.422
2500	19.367

Table 5.14: Total number of nodes vs Weighted average bandwidth of all clusters

Total number of nodes	Weighted Average Latency of all clusters
400	11.873
625	13.108
900	13.823
1225	14.869
1600	15.651
2025	15.835
2500	16.808

Table 5.15: Total number of nodes vs Weighted average latency of all clusters

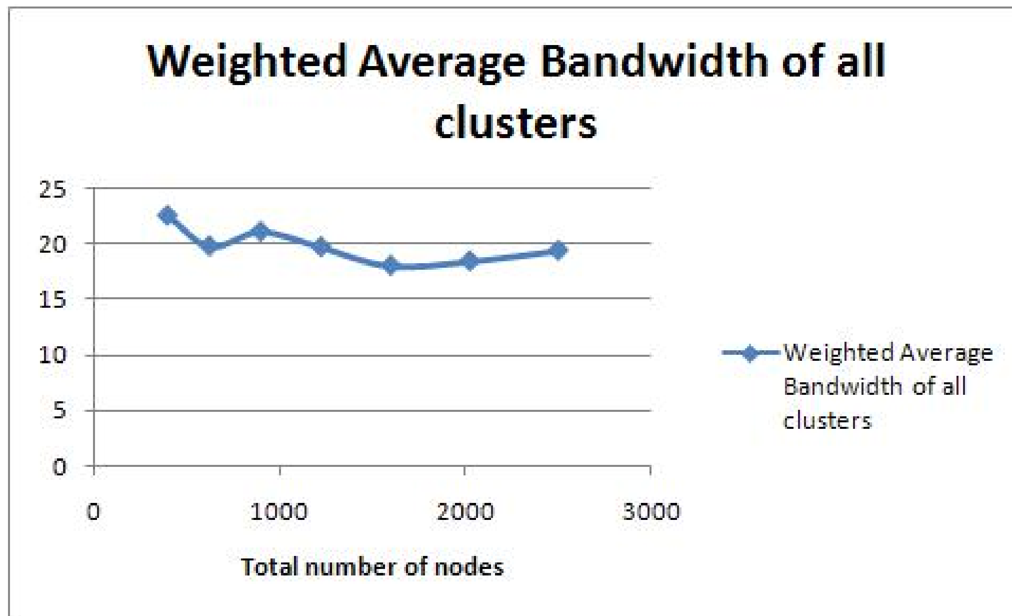


Figure 5.15: Total number of nodes vs Weighted Average Bandwidth of all clusters

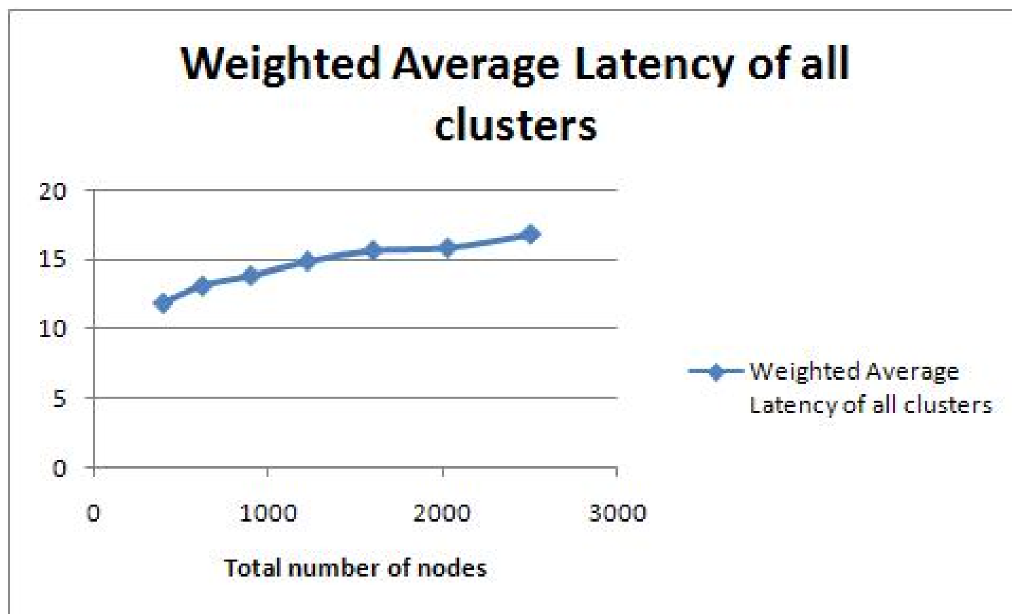


Figure 5.16: Total number of nodes vs Weighted Average Latency of all clusters

hops between nodes of the network increases resulting in each cluster possibly choosing nodes that are far away and that will have higher latency in communications.

Total number of nodes	Weighted Average Number of Hops of all clusters
400	8.815
625	9.525
900	9.829
1225	11.101
1600	11.912
2025	11.891
2500	12.192

Table 5.16: Total number of nodes vs Weighted average number of hops of all clusters

The effect on the weighted average number of hops of all clusters as the number of nodes joining the system is increased is shown in Table 5.16 and in Figure 5.17

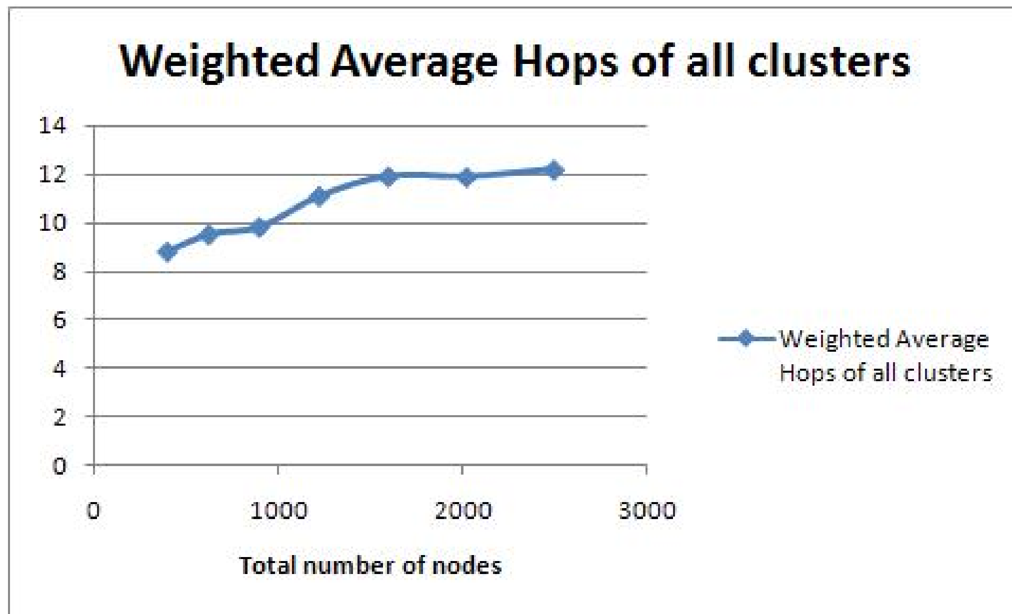


Figure 5.17: Total number of nodes vs Weighted Average Hops of all clusters

As the number of nodes is increased, the weighted average number of hops for all clusters increases. This is again because as the number of nodes increases, the number of

Total number of nodes	Weighted Average Proximity of all clusters
400	0.0396
625	0.0322
900	0.0317
1225	0.0249
1600	0.0219
2025	0.0215
2500	0.0231

Table 5.17: Total number of nodes vs Weighted average proximity of all clusters

hops between nodes of the network increases resulting in each cluster being more likely to choose nodes that are many hops away.

The effect on the weighted average proximity of all clusters as the number of nodes joining the system is increased is shown in Table 5.17 and in Figure 5.18

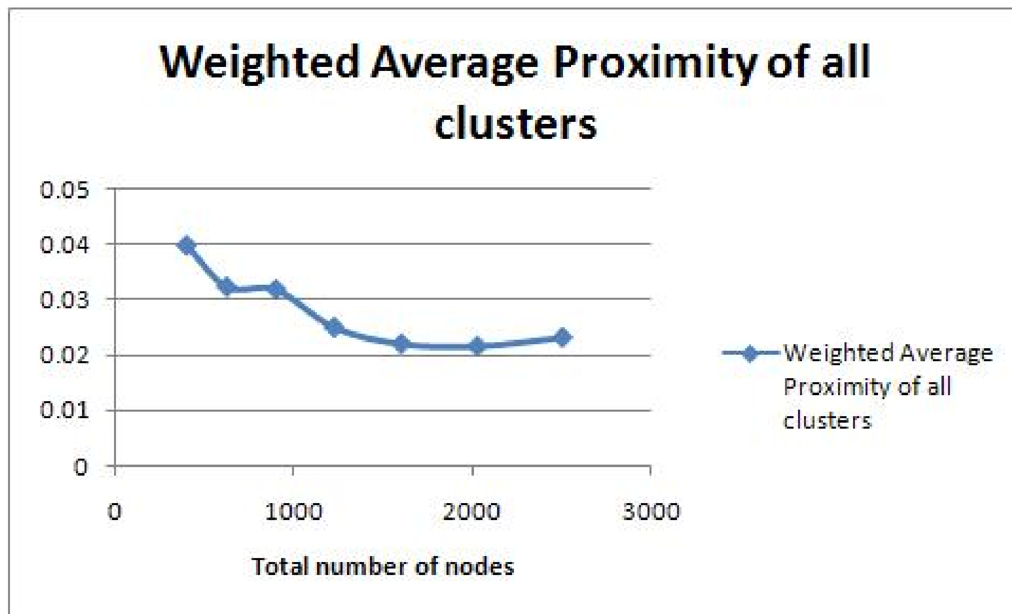


Figure 5.18: Total number of nodes vs Weighted Average Proximity of all clusters

Total number of nodes	Weighted Average Processor Speed Similarity of all clusters
400	0.752
625	0.742
900	0.742
1225	0.752
1600	0.752
2025	0.766
2500	0.751

Table 5.18: Total number of nodes vs Weighted average processor speed similarity of all clusters

As the number of nodes is increased in the system from 400 to 2500 (more than 6 times), the weighted average proximity of all clusters decreases (but only by less than 50 percent). Again, with more nodes, the nodes in each cluster may be further away from one another.

The effect on the weighted average processor speed similarity of all clusters as the number of nodes joining the system is increased is shown in Table 5.18 and in Figure 5.19

As the number of nodes is increased from 400 to 2500 (more than 6 times), the weighted average processor speed similarity of all clusters increases slightly. This is because as more nodes join the system, each node will have more clusters to choose to join, resulting in joining a cluster that is more similar in terms of processor speed.

The effect on the weighted average memory size similarity of all clusters as the number of nodes joining the system is increased is shown in Table 5.19 and in Figure 5.20

As the number of nodes is increased in the system from 400 to 2500, the weighted

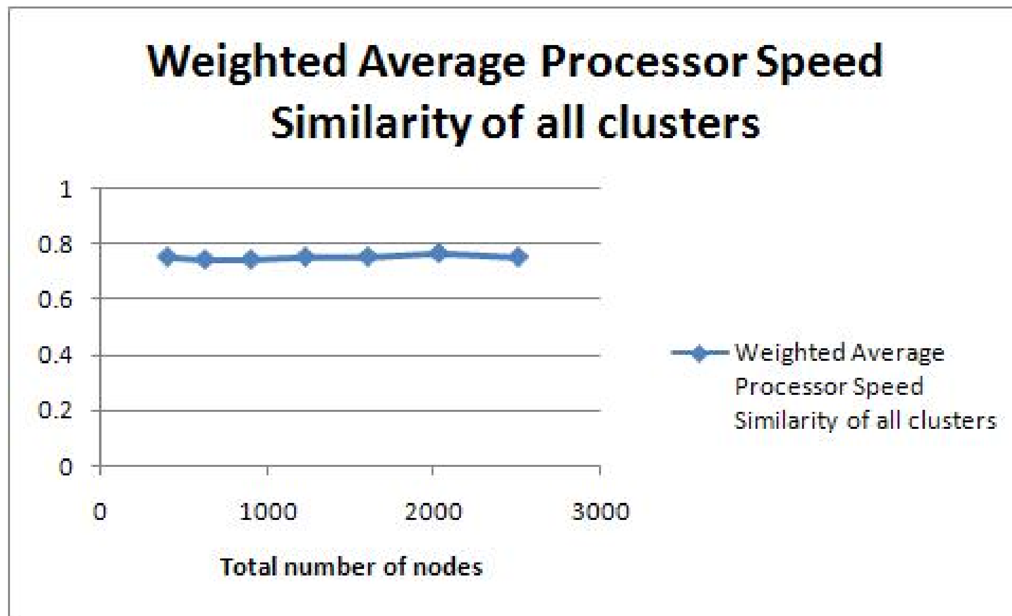


Figure 5.19: Total number of nodes vs Weighted Average Processor Speed Similarity of all clusters

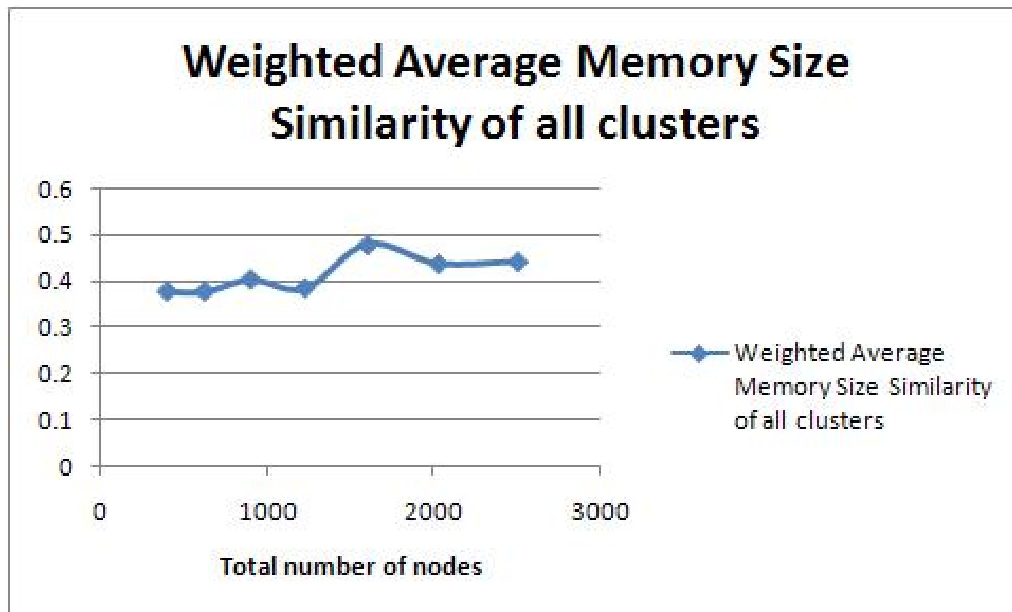


Figure 5.20: Total number of nodes vs Weighted Average Memory Size Similarity of all clusters

Total number of nodes	Weighted Average Memory Size Similarity of all clusters
400	0.377
625	0.377
900	0.403
1225	0.384
1600	0.479
2025	0.437
2500	0.442

Table 5.19: Total number of nodes vs Weighted average memory size similarity of all clusters

average memory size similarity of all clusters increases slightly because as more nodes join the system, each node will have more clusters to choose from resulting in clusters that, on average, are more similar in terms of memory size.

The effect on the weighted average hard disk space similarity of all clusters as the number of nodes joining the system is increased is shown in Table 5.20 and in Figure 5.21

As the number of nodes is increased, the weighted average hard disk space similarity of all clusters also increases. This is again because as more nodes join the system, each node will have more clusters to choose to join, resulting in clusters that are more similar in terms of hard disk space available).

5.3.3 Assessment of Different Splitting Algorithms

In my final set of experiments, I compare the different splitting algorithms I have proposed to determine the differences between them so potential users of my system can select the

Total number of nodes	Weighted Average Hard Disk Space Similarity of all clusters
400	0.592
625	0.614
900	0.614
1225	0.660
1600	0.612
2025	0.662
2500	0.741

Table 5.20: Total number of nodes vs Weighted average hard disk space similarity of all clusters

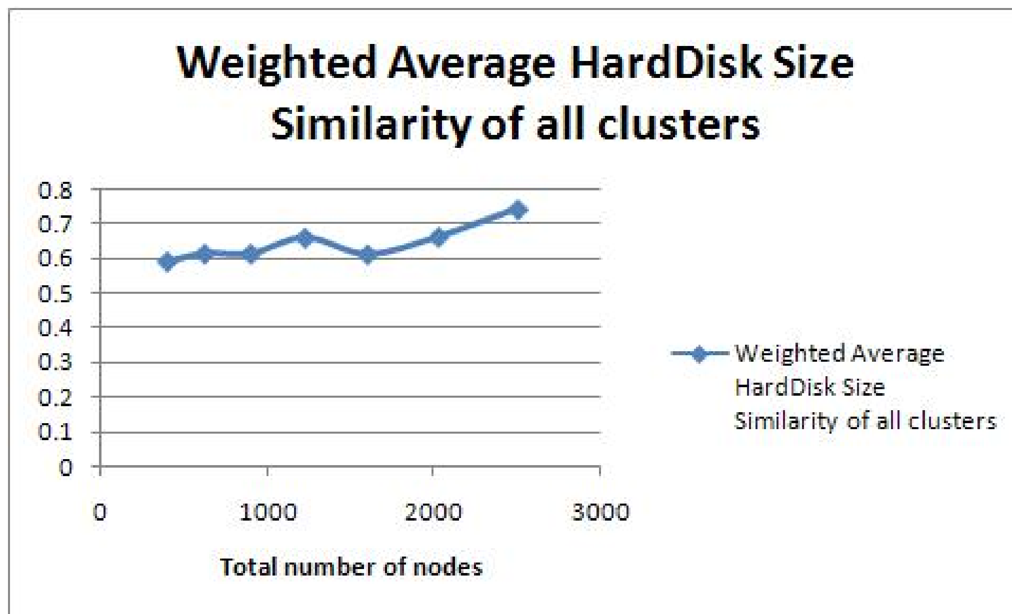


Figure 5.21: Total number of nodes vs Weighted Average Hard Disk Space Similarity of all clusters

algorithm that best meets their needs. Each algorithm is assessed in terms of its impact on the various network-related parameters of the clusters being constructed.

The effect of the different splitting algorithms on the weighted average proximity is

Total number of nodes	Capacity-based splitting	Proximity-based splitting	Improved Proximity-based splitting
400	0.0396	0.0515	0.0548
625	0.0322	0.0444	0.0473
900	0.0317	0.0364	0.0424
1225	0.0249	0.0314	0.0338
1600	0.0219	0.0282	0.0307
2025	0.0215	0.0331	0.0329
2500	0.0231	0.0309	0.0340

Table 5.21: Effect of different splitting algorithms on weighted average proximity

shown in Table 5.21 and in Figure 5.22

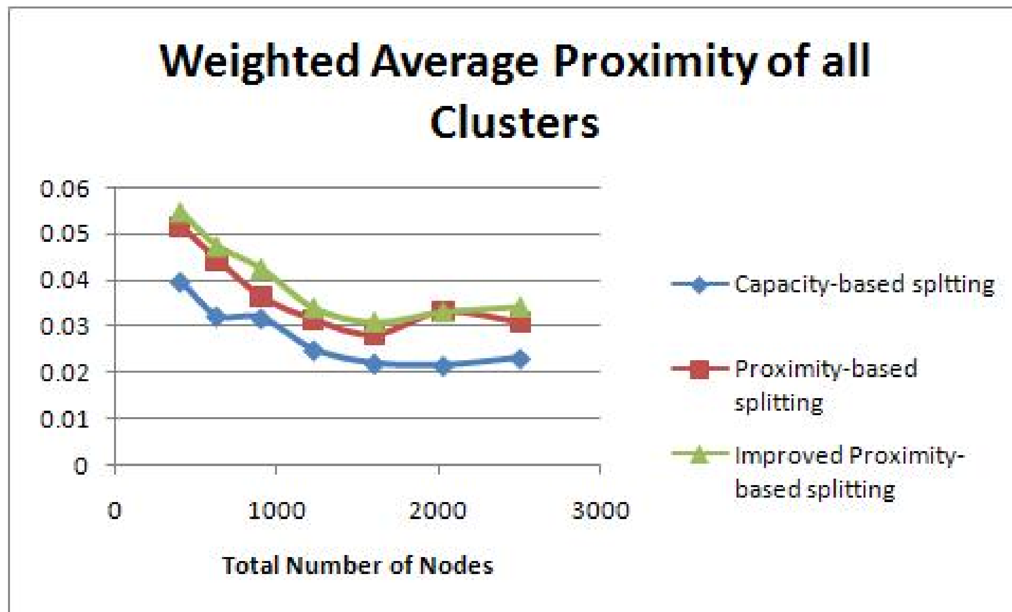


Figure 5.22: Effect of different splitting algorithms on weighted average proximity

Proximity-based splitting increases the weighted average proximity of all clusters significantly over Capacity-based splitting and the improved Proximity-based splitting

Total number of nodes	Capacity-based splitting	Proximity-based splitting	Improved Proximity-based splitting
400	22.492	27.093	28.907
625	19.754	24.898	26.174
900	21.050	23.340	26.261
1225	19.687	22.962	24.572
1600	18.007	21.589	22.661
2025	18.422	25.454	24.932
2500	19.367	24.664	26.360

Table 5.22: Effect of different splitting algorithms on weighted average bandwidth

increases the weighted average proximity of all clusters slightly over Proximity-based splitting. This confirms that the algorithms achieve their designed purpose.

The effect of the different splitting algorithms on weighted average bandwidth is shown in Table 5.22 and in Figure 5.23

Proximity-based splitting increases the weighted average bandwidth of all clusters significantly over Capacity-based splitting. Improved Proximity-based splitting increases the weighted average bandwidth of all clusters slightly over Proximity-based splitting.

The effect of the different splitting algorithms on the weighted average latency is shown in Table 5.23 and in Figure 5.24

Proximity-based splitting decreases the weighted average latency of all clusters significantly over Capacity-based splitting. Improved Proximity-based splitting further decreases the weighted average latency of all clusters over Proximity-based splitting.

The effect of the different splitting algorithms on the weighted average number of

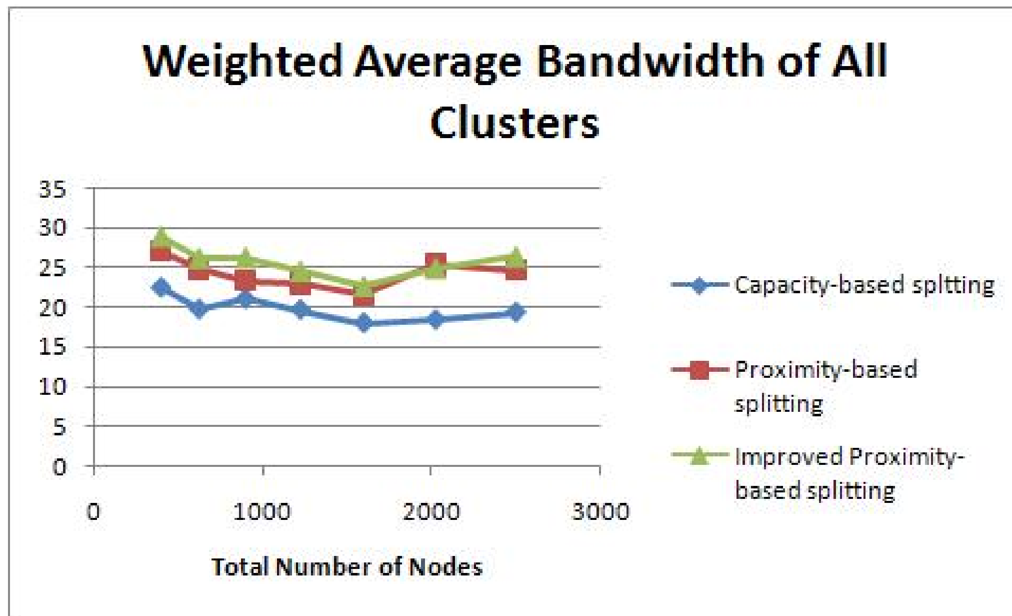


Figure 5.23: Effect of different splitting algorithms on weighted average bandwidth

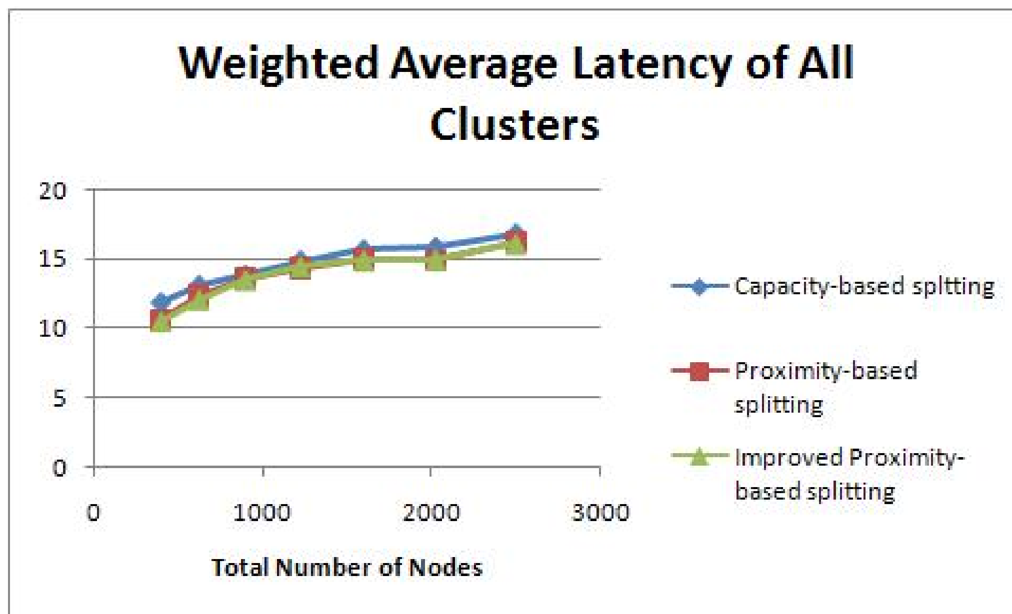


Figure 5.24: Effect of different splitting algorithms on weighted average latency

hops is shown in Table 5.24 and in Figure 5.25

Proximity-based splitting decreases the weighted average number of hops of all clusters

Total number of nodes	Capacity-based splitting	Proximity-based splitting	Improved Proximity-based splitting
400	11.873	10.641	10.504
625	13.108	12.387	12.077
900	13.823	13.560	13.505
1225	14.869	14.370	14.454
1600	15.651	14.956	14.930
2025	15.835	14.909	14.972
2500	16.808	16.219	16.125

Table 5.23: Effect of different splitting algorithms on weighted average latency

Total number of nodes	Capacity-based splitting	Proximity-based splitting	Improved Proximity-based splitting
400	8.815	7.973	7.615
625	9.525	8.991	8.816
900	9.829	9.656	9.605
1225	11.101	10.732	10.747
1600	11.912	11.346	11.332
2025	11.891	11.176	11.267
2500	12.192	11.723	11.667

Table 5.24: Effect of different splitting algorithms on weighted average number of hops

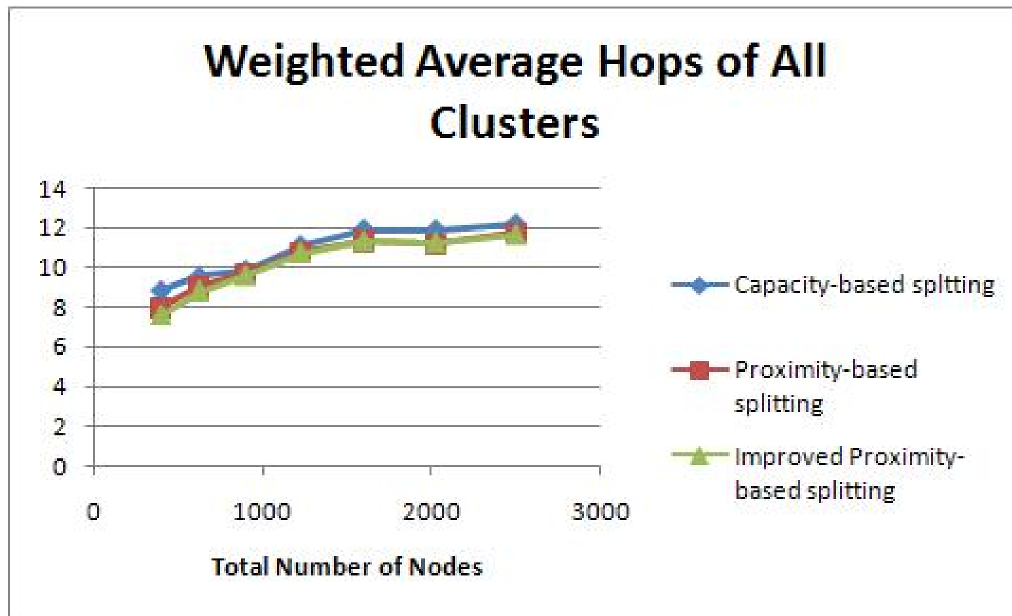


Figure 5.25: Effect of different splitting algorithms on weighted average number of hops

significantly over Capacity-based splitting and the improved Proximity-based splitting algorithms decreases the weighted average number of hops of all clusters slightly over Proximity-based splitting.

Chapter 6

Conclusions and Future work

6.1 Conclusions

The grouping of geographically distributed idle workstations into useful clusters presents a significant opportunity for users needing to run large-scale, loosely coupled parallel jobs. Building on the ideas behind volunteer computing, in this thesis, I have described a peer-to-peer (P2P) resource discovery system that enables idle workstations to be grouped into useful compute clusters. My system clusters together workstations that are similar in terms of computation power (processor speed, memory size, and available hard disk space) and that are relatively close in terms of bandwidth, latency and number of hops. My system uses a bidding-based approach to group similar, nearby workstations and an election mechanism to choose leaders from among the clustered machines. It also performs cluster splitting to split excessively large clusters into smaller ones when it becomes problematic for the leader to manage them (i.e. when they reach their management capacity).

To assess the usefulness of my system, I have also done some initial simulation experiments. The network structure used in the simulations was created using BRITE, a topology generator designed to produce realistic network structures. In this way, my simulation experiments should reflect potential use in a real, Internet environment. The bidding function (and hence the clustering criterion) used in the simulations was the one presented in this thesis, though other such functions could be easily incorporated into the framework to achieve different clustering goals. The results of my simulations confirm that the clustering function proposed in the thesis: (i) achieves the goal of creating similar and nearby clusters, and (ii) is tunable using the various weights to achieve different behaviours (within the constraints imposed by the machines available in the simulated network environment). They further show the effects of varying the number of nodes available to form clusters and the average "closeness" of nodes in the simulated network (both resulting in better clusters). Finally, the simulation results also demonstrate the superiority of my proximity-based and improved proximity-based splitting algorithms.

Thus, the contributions made in this thesis are:

- Development of a P2P framework for grouping idle workstations into clusters,
- Description of a general purpose clustering strategy that uses the framework to group similar, nearby workstations into *useful* clusters suitable for running loosely coupled parallel jobs, and
- An initial simulation-based assessment of the framework and clustering strategy that shows that they are effective in doing cluster formation.

6.2 Future work

There are several specific opportunities to extend and, in some cases, improve on the work presented in this thesis.

In the short term, both cluster merging and handling of node failures, as discussed earlier, need to be implemented. Additionally, the simulation experiments should be expanded to at least assess the effectiveness of the presented clustering strategy for a wider range of network structures.

In the mid term, it would be interesting to explore some other possible splitting algorithms. The presented algorithms were primarily focused on maintaining a high-degree of proximity in the resulting clusters. Other factors might also be considered. For example, if the algorithms to be run on the resulting clusters required only very infrequent communication, then the importance of proximity would be diminished. In general, such changes in anticipated use of the clusters might also lead to significantly different clustering strategies. For example, for parameter sweep applications, clustering based primarily on number and availability of nodes might make more sense. Similarly in a commercial environment, cost of use would be a likely clustering factor.

Finally while this thesis does not address the use of the resulting clusters, in the long term, some software to support such use will be needed. Thus, at least how the primary leaders of each cluster would reply to a "Resource Request" message from a user (or broker) and how each cluster is managed while jobs are being run on the cluster need to be considered. Ultimately, issues such as resource reservation and trust issues related to the use of shared infrastructure will also need to be addressed.

Bibliography

[1] Gabrielle A, Gerd Lanfermann, Andr Merzkyt, Thomas Radke, Edward Seidel, and John Shalo. The cactus code: A problem solving environment for the grid. In *In Proc. High Performance Distributed Computing (HPDC-2000*, pages 253–260.

[2] Refining Bias in Clustering Algorithms via Problem-specific constraints. http://www.litech.org/˜wkiri/Papers/wagstaff-proposal.ps. Last accessed on: May 11, 2005.

[3] H. Abu-Amara and V. Gummadi. New model and algorithms for leader election in synchronous fiber-optic networks. *IEEE Trans. Parallel Distrib. Syst.*, 5(8):891–896, 1994.

[4] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley. The Design and Implementation of an Intentional Naming System. In *Symposium on Operating Systems Principles*, pages 186–201, 1999.

[5] Gabrielle Allen, Thomas Dramlitsch, Ian Foster, Tom Goodale, Nick Karonis, Matei Ripeanu, Ed Seidel, and Brian Toonen. Cactus-G: Enabling High-Performance Simulation in Heterogeneous Distributed Computing Environments. In *Fourth Globus Retreat*, Pittsburg, May 2000.

[6] Magdalena Balazinska, Hari Balakrishnan, and David Karger. Ins/twine: A scalable peer-to-peer architecture for intentional resource discovery. In *Pervasive '02: Proceedings of the First International Conference on Pervasive Computing*, pages 195–210. Springer-Verlag, 2002.

[7] Mohammad Banikazemi. IP Multicasting: Concepts, Algorithms, and Protocols. http://www.cse.ohio-state.edu/jain/cis788-97/ftp/ipmulticast/index.htm. Last accessed on: May 11, 2005.

[8] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet group management protocol, version 3, 2002.

[9] Ajay Chander, Steven Dawson, Paptrick Lincoln, and David Stringer-Calvert. NEVRLATE: Scalable Resource Discovery. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, pages 382–388, May 2002.

[10] Chunming Chen, M. Maheswaran, and M. Toulouse. Supporting Co-allocation in an Auctioning-based Resource Allocator for Grid Systems. In *11th IEEE Heterogeneous Computing Workshop (HCW 2002)*, Fort Lauderdale, Florida, April 2002.

[11] Y.-C. Chow, K.C.K. Luo, and R. Newman-Wolfe. An optimal distributed algorithm for failure-driven leader election in bounded-degree networks. In *Distributed Computing Systems, 1992., Proceedings of the Third Workshop on Future Trends of*, pages 136–141, Apr 1992.

[12] Israel Cidon and Osnat Mokryn. Propagation and Leader Election in a Multihop Broadcast Environment. In *International Symposium on Distributed Computing*, pages 104–118, 1998.

[13] Ian Foster and Carl Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[14] Ian Foster and Carl Kesselman. The Globus project: a status report. *Future Generation Computer Systems*, 15(5–6):607–621, 1999.

[15] Eitan Frachtenberg, Fabrizio Petrini, Juan Fernandez, and Salvador Coll. Scalable Resource Management in High Performance Computers. In *IEEE Cluster 2002*, pages 305–314, Chicago, IL, September 2002.

[16] Greg N. Frederickson and Nancy A. Lynch. Electing a leader in a synchronous ring. *J. ACM*, 34(1):98–115, 1987.

[17] Hector Garcia-Molina. Elections in a distributed computing system. *IEEE Trans. Computers*, 31(1):48–59, 1982.

[18] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: an efficient clustering algorithm for large databases. *Inf. Syst.*, 26(1):35–58, 2001.

[19] Hartigan, John A. In *Clustering Algorithms, John Wiley & Sons, New York, NY*, 1975.

[20] Shing-Tsaan Huang. Leader election in uniform rings. *ACM Trans. Program. Lang. Syst.*, 15(3):563–573, 1993.

[21] A. Iamnitchi, I. Foster, Nurmi, and D.C. A peer-to-peer approach to resource discovery in grid environments. In *11th IEEE International Symposium on High Performance Distributed Computing*, page 419, July 2002.

[22] Alon Itai, Shay Kutten, Yaron Wolfstahl, and Shmuel Zaks. Optimal distributed t-resilient election in complete networks. *IEEE Trans. Softw. Eng.*, 16(4):415–420, 1990.

[23] J. Frey, T. Tannenbaum, I. Foster, M. Livny, S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *Cluster Computing, 5(3):237-246*, 2002.

[24] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[25] Nicholas T. Karonis, Brian Toonen, and Ian Foster. Mpich-g2: a grid-enabled implementation of the message passing interface. *J. Parallel Distrib. Comput.*, 63(5):551–563, 2003.

[26] Tai Woo Kim, Eui Hong Kim, Joong Kwon Kim, and Tai Yun Kim. A leader election algorithm in a distributed computing system. In *FTDCS*, pages 481–487, 1995.

[27] Pierre L'Ecuyer, Lakhdar Meliani, and Jean Vaucher. SSJ: A Framework for Stochastic Simulation in Java. In *WSC '02: Proceedings of the 34th conference on Winter simulation*, pages 234–242. Winter Simulation Conference, 2002.

[28] Juhnyoung Lee, Sang goo Lee2, and Suekyung Lee2. A Framework of Winner Determination Algorithms for Internet Auctions. In *Lecture Notes in Computer Science*, volume 2713, pages 415–424, August 2003.

[29] M. Litzkow, M. Livny, and M.W. Mutka. Condor — A Hunter of Idle Workstations. In *Eighth International Conference on Distributed Computing Systems*, pages 104–111, 1988.

[30] Navneet Malpani, Jennifer L. Welch, and Nitin Vaidya. Leader election algorithms for mobile ad hoc networks. In *DIALM '00: Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 96–103, New York, NY, USA, 2000. ACM.

[31] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. Brite: An approach to universal topology generation. *Modeling, Analysis, and Simulation of Computer Systems, International Symposium on*, 0:0346, 2001.

[32] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: Universal topology generation from a user's perspective. Technical Report 2001-003, 1 2001.

[33] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. Technical Report UT-CS-94-230, 1994.

[34] Pragyansmita Paul. Seti @ home project and its website. *Crossroads*, 8(3):3–5, 2002.

[35] Donald P. Pazel, Tamar Eilam, Liana L. Fong, Michael Kalantar, Karen Appleby, and German Goldszmidt. Neptune: A Dynamic Resource Allocation and Planning System for a Cluster Computing Utility. In *Second ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 42–55, Berlin, Germany, May 2002. IEEE.

[36] Fabrizio Petrini, Wu chun Feng, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg. Quadrics Network (QsNet): High-Performance Clustering Technology. In *Hot Interconnects 9*, pages 46–57, Stanford University, Palo Alto, CA, August 2001.

[37] Ralph Wittmann and Martina Zitterbart. *Multicast communication: protocols and applications*. Morgan Kaufmann Publishers Inc., 1999.

[38] Multicasting in the Internet. http://www.rubynet.com/multicasting.htm. Last accessed on: November 1, 2004.

[39] An Overview of different Clustering Algorithms. http://www.comp.lancs.ac.uk/ kristof/research/notes/clustr/. Last accessed on: May 11, 2005.

[40] Gnutella. http://www.gnutella.com/. Last accessed on: May 11, 2005.

[41] Clustering. http://riot.ieor.berkeley.edu/riot/Applications/Clustering/. Last accessed on: May 11, 2005.

[42] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. pages 103–114, 1996.