

**CLUSTER COMPUTING:  
A NOVEL PEER-TO-PEER CLUSTER FOR GENERIC  
APPLICATION SHARING**

GUO CHEN

NATIONAL UNIVERSITY OF SINGAPORE

2013

**CLUSTER COMPUTING:  
A NOVEL PEER-TO-PEER CLUSTER FOR GENERIC  
APPLICATION SHARING**

GUO CHEN

(B.ENG. (HONS.), NUS)

A THESIS SUBMITTED FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY  
DEPARTMENT OF ELECTRICAL AND COMPUTER  
ENGINEERING  
NATIONAL UNIVERSITY OF SINGAPORE

2013

# DECLARATION

I hereby declare that the thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.



---

Guo Chen

01 Aug 2013

# ACKNOWLEDGEMENTS

I owe my deepest gratitude to my supervisor, Associate Professor Tay Teng Tiow for his unceasing support and inspiration in guiding me through all these years to make this thesis possible. I am truly grateful for his constant encouragement and teachings during this journey. In addition to the valuable technical knowledge, I have also learned from him the importance of being persistent, thoughtful and conscientious. I sincerely wish him happiness every day.

Special thanks go to Associate Professor Bharadwaj Veeravalli and Dr. Ha Yajun from ECE department of National University of Singapore. I am thankful for their helpful comments and invaluable feedbacks during my research work.

I would like to express thanks to my current employer, Computational Engineering department in Advanced Technology Centre of Rolls-Royce Singapore, my manager and colleagues for their support during the time that I spent working on this thesis.

I thank my lab partner Dr. Zhu Cen Zhe who contributed his time and ideas whenever I talked to him about the difficulties encountered. I also would like to acknowledge a group of FYP students who have contributed their time in related work about this research: Mr. Tan Kah Onn, Mr. Mohammed Kassim and Mr. Chan Chew Wye.

I would like to thank department of Electrical and Computer Engineering, National University of Singapore for offering me the scholarship of my study and providing me with this great opportunity to work on this exciting project.

On a personal note, I would like to thank my families for their unlimited love and support. I wish to offer my heartfelt gratitude to my husband Zhao Fucai who has constantly supported and encouraged me at difficult times to work on completing my thesis. I would like to dedicate this thesis to my loving son Zhao Xinhong, who has accompanied me throughout the writing process and helped me to stay light-hearted.

Lastly, I am very grateful to those who have given me their support in any respect during the completion of the thesis.

Guo Chen

01 Aug 2013

# TABLE OF CONTENTS

---

---

<b>DECLARATION</b> .....	<b>1</b>
<b>ACKNOWLEDGEMENTS</b> .....	<b>2</b>
<b>TABLE OF CONTENTS</b> .....	<b>3</b>
<b>SUMMARY</b> .....	<b>7</b>
<b>LIST OF TABLES</b> .....	<b>9</b>
<b>LIST OF FIGURES</b> .....	<b>10</b>
<b>CHAPTER 1 INTRODUCTION</b> .....	<b>13</b>
<b>1.1 Cluster Computing</b> .....	<b>13</b>
1.1.1 Definition.....	13
1.1.2 Applications of Cluster Computing.....	14
1.1.3 Advantages and Disadvantages of Cluster Computing.....	16
<b>1.2 Application Sharing</b> .....	<b>20</b>
1.2.1 Definition.....	20
1.2.2 Application Specific v.s. Generic Application Sharing .....	21
1.2.3 Scenarios: Remote Log-in v.s. Real-time Collaboration.....	22
1.2.4 Benefits and Challenges .....	23
<b>1.3 P2P Network System</b> .....	<b>24</b>
1.3.1 Structured P2P System .....	24
1.3.2 Unstructured P2P System.....	25
<b>1.4 Research Problem and Scope of Work</b> .....	<b>26</b>
1.4.1 Problem Statement .....	26
1.4.2 Sub-problems .....	27
<b>1.5 Contributions</b> .....	<b>29</b>
<b>1.6 Thesis Outline</b> .....	<b>31</b>
<b>CHAPTER 2 RELATED WORK</b> .....	<b>33</b>
<b>2.1 Cluster Computing Solutions</b> .....	<b>33</b>
2.1.1 Heterogeneous support .....	33
2.1.2 Parallel programming support .....	33

2.1.3	Check-pointing .....	34
2.1.4	Process migration.....	34
2.1.5	Load balancing.....	35
2.1.6	Graphical user interface .....	35
<b>2.2</b>	<b>Application Sharing Solutions .....</b>	<b>36</b>
<b>2.3</b>	<b>Communication Protocols for Application Sharing .....</b>	<b>42</b>
2.3.1	Remote Frame Buffer (RFB) for Virtual Network Computing (VNC) .....	42
2.3.2	Microsoft Remote Desktop Protocol (RDP).....	43
2.3.3	ITU-T T.128. Multipoint Application Sharing .....	44
<b>CHAPTER 3 A NOVEL BROKER-MEDIATED SOLUTION TO GENERIC APPLICATION SHARING IN A CLUSTER OF CLOSED OPERATING SYSTEMS</b>		<b>46</b>
<b>3.1</b>	<b>Introduction.....</b>	<b>46</b>
<b>3.2</b>	<b>System Overview .....</b>	<b>48</b>
3.2.1	System Architectures .....	49
3.2.2	Use Case Diagram.....	53
<b>3.3</b>	<b>Design and Methodology .....</b>	<b>54</b>
3.3.1	Establishing Multiple Remote Application Sessions.....	55
<b>3.4</b>	<b>Implementation of a Demonstrating System.....</b>	<b>68</b>
3.4.1	Detailed Programming Model .....	68
3.4.2	App Share Client .....	70
3.4.3	App Share Server .....	72
<b>3.5</b>	<b>Results and Discussion .....</b>	<b>76</b>
3.5.1	User Interface.....	76
3.5.2	Multi-session Load Analysis.....	78
3.5.3	License Issue on Application Sharing .....	83
3.5.4	Some Limitations of Our Implementations .....	84
<b>3.6</b>	<b>Summary.....</b>	<b>85</b>
<b>CHAPTER 4 BUILDING A RELIABLE FILE SYSTEM FOR FAULT-TOLERANT SERVICES .....</b>		<b>86</b>
<b>4.1</b>	<b>Introduction.....</b>	<b>86</b>

<b>4.2</b>	<b>Portable File System (PFS) on Filesystem in User Space (FUSE)</b>	<b>87</b>
<b>4.3</b>	<b>Implementation of PFS</b>	<b>88</b>
4.3.1	Set-up of FUSE and Host Computers	88
4.3.2	Loggings of File Operations	90
4.3.3	Client-Server Communication	90
4.3.4	Explanation of Callback Functions	92
<b>4.4</b>	<b>Testing and Evaluation</b>	<b>95</b>
4.4.1	Latency Test	96
4.4.2	Integrity Test for File System	96
<b>4.5</b>	<b>Summary</b>	<b>98</b>
<b>CHAPTER 5 IMPRECISE COMPUTATION SCHEDULING ALGORITHMS FOR REAL-TIME CLUSTER COMPUTING</b>		<b>99</b>
<b>5.1</b>	<b>Introduction</b>	<b>99</b>
<b>5.2</b>	<b>System Model</b>	<b>102</b>
<b>5.3</b>	<b>Scheduling Method and Modelling</b>	<b>105</b>
5.3.1	Scheduling Algorithms	105
5.3.2	Optimal Load Distribution	107
<b>5.4</b>	<b>ICSCluster Simulator</b>	<b>108</b>
<b>5.5</b>	<b>Results and Analysis</b>	<b>112</b>
<b>5.6</b>	<b>Summary</b>	<b>117</b>
<b>CHAPTER 6 CONCLUSIONS AND FUTURE WORK</b>		<b>118</b>
<b>6.1</b>	<b>Conclusions</b>	<b>118</b>
<b>6.2</b>	<b>Future Work</b>	<b>119</b>
6.2.1	Security Management	119
6.2.2	Reliability Management	120
6.2.3	Resource Management	120
<b>BIBLIOGRAPHY</b>		<b>122</b>
<b>GLOSSARY</b>		<b>129</b>
<b>APPENDICES</b>		<b>131</b>
<b>A.</b>	<b>RDP Connection Sequence and PDU</b>	<b>131</b>
a.	RDP Connection Sequence	131
b.	Protocol Data Unit (PDU)	133

c.	Protocol Packet Analysis for Initializing the Connection .....	134
<b>B.</b>	<b>Cluster Management .....</b>	<b>135</b>
<b>C.</b>	<b>Incoming and Outgoing Packet Management.....</b>	<b>137</b>
<b>D.</b>	<b>Demonstrations .....</b>	<b>141</b>
a.	Rdesktop as the Client program.....	141
b.	Compile Rdesktop for Windows.....	142
c.	SeamlessRDP and accessing remote applications.....	144
<b>E.</b>	<b>Customization of a Remote Application Session Using RDP File .....</b>	<b>150</b>
<b>F.</b>	<b>Integrity Test for PFS File System .....</b>	<b>152</b>
<b>G.</b>	<b>Latency Test for PFS File System .....</b>	<b>154</b>
<b>H.</b>	<b>ICSCluster (Imprecise Computation Scheduling Cluster) Simulation .....</b>	<b>156</b>
<b>I.</b>	<b>Research Process .....</b>	<b>177</b>
	<b>PUBLICATIONS .....</b>	<b>178</b>



## SUMMARY

With advances in hardware and networking technologies and mass manufacturing, the cost of high end hardware has fallen dramatically in recent years. However, software cost still remains high and is the dominant fraction of the overall computing budget. Application sharing is a promising solution to reduce the overall IT cost. Currently software licenses are still based on the number of copies installed. An organization can thus reduce the IT cost if the users are able to remotely access the software that is installed on certain computer servers instead of running the software on every local computer. In this research, a generic application sharing architecture was proposed for users' application sharing in a cluster of closed operating systems such as Microsoft Windows. The broker-mediated solution allows multiple users to access a single user software license on a time multiplex basis through a single logged in user. An application sharing tool called ShAppliT has been introduced and implemented in Microsoft Windows operating system. Their performance has been evaluated on CPU usage and memory consumption when a computer is hosting multiple concurrent shared application sessions.

In addition, a failure-save solution was implemented for fault-tolerant application services in clusters which enabled user to login to the file server from anywhere, synchronize document to last saved state on server and provide certain degree of portability. The proposed idea of building a reliable file system was implemented successfully. Testing and evaluation of the system were also performed and results showed that the implemented had reached reasonable level of reliability.

Finally, imprecise computation scheduling was modelled and simulated to enhance QoS for real-time systems and improve the energy efficiency for large

scale computing in clusters. Measurements of simulation on a large number of task sets showed that imprecise computation improved the system reliability when scheduling intensive workloads with less schedule timing faults, CPU cycles and energy-efficiency improvement.

## LIST OF TABLES

Table 1 Comparison of related work.....	37
Table 2 Comparison of application sharing solutions.....	40
Table 3 Client ID and Window ID.....	75
Table 4 Details on Windows Task Manager Performance analysis [62] .....	78
Table 5 Multi-session load analysis on host computer with ShAppliT V1.0.....	79
Table 6 Multi-session load analysis on host computer with ShAppliT V2.0.....	80
Table 7 Call-back functions implemented in PSF .....	92
Table 8 Notations and definitions of the System .....	104
Table 9 Structure array and fields' definition .....	109
Table 10 Scheduling algorithms for mandatory and optional tasks .....	111

## LIST OF FIGURES

Figure 1 Architecture of Hadoop Ecosystem.....	16
Figure 2 Why Cluster Computing.....	17
Figure 3 Hadoop Core.....	18
Figure 4 Taxonomy study on application sharing.....	20
Figure 5 Application Sharing Models.....	22
Figure 6 Definition of research problem.....	26
Figure 7 Main Contributions.....	29
Figure 8 Windows XP server architecture [14] .....	40
Figure 9 Multipoint application sharing protocol T. 128 and its family [54] .....	45
Figure 10 System overview .....	48
Figure 11 Application sharing cluster overview .....	49
Figure 12 Access shared application resources in a cluster .....	50
Figure 13 Illustration of system architecture .....	51
Figure 14 Layered architecture of a cluster system .....	52
Figure 15 Application sharing use cases diagram.....	53
Figure 16 Broker mediated application sharing system architecture .....	55
Figure 17 System architecture model of ShAppliT .....	56
Figure 18 State diagram of App Share Client during connection sequence .....	57
Figure 19 State diagram of App Share Server during connection sequence.....	59
Figure 20 RDP connection sequence diagram [55] .....	62
Figure 21 RDP architecture .....	63
Figure 22 Virtual channel in RDP .....	64
Figure 23 Data stream controller .....	66
Figure 24 Illustration of focused window and allocated client.....	67
Figure 25 Programming model of ShAppliT system.....	69
Figure 26 Programming flow chart of App Share Server .....	73

Figure 27 Control messages in seamless virtual channel.....	75
Figure 28 Screen shot of the demonstrated App Share.....	77
Figure 29 Screen shot of the demonstrated App Share: setting share/un-share applications .....	77
Figure 30 Memory performance of ShAppliT V1.0 when hosting multiple remote sessions .....	81
Figure 31 Memory performance of ShAppliT V2.0 when hosting multiple remote sessions .....	82
Figure 32 Comparison between ShAppliT V1.0 and ShAppliT V2.0 on commit charge when hosting multiple remote sessions.....	83
Figure 33 Overview of reliable file system architecture.....	88
Figure 34 Flow for ID checking on server site .....	92
Figure 35 Flow-chart for write operation at client.....	94
Figure 36 Flow-chart for write operation at server .....	95
Figure 37 Graph for read latency test results .....	97
Figure 38 Graph for write latency test results.....	97
Figure 39 Cluster computing system overview.....	103
Figure 40 Cluster computing system model .....	104
Figure 41 Timing diagram of the system.....	106
Figure 42 Timing diagram: optimal load divisible for a cluster of processing nodes [84] .....	107
Figure 43 Timing diagram: optimal load divisible for equivalent cluster network [84].	108
Figure 44 Block diagram of the simulator .....	109
Figure 45 Flow chart of simulation imprecise computation scheduling .....	111
Figure 46 Schedulable rates vs. work load for precise scheduling .....	113
Figure 47 Schedulable rates vs. workload for imprecise computation .....	114
Figure 48 Comparison between precise and imprecise computation on schedulable rates for EDF scheduling algorithms .....	114
Figure 49 Comparison between precise and imprecise computation on schedulable rates for RMS scheduling algorithms .....	115
Figure 50 Comparison between precise and imprecise computation on schedulable rates for LEF scheduling algorithms .....	115

Figure 51 Comparison between precise and imprecise computation on schedulable rates for MEF scheduling algorithms .....	116
Figure 52 Taxonomy for security management .....	120
Figure 53 Taxonomy for reliability management .....	120
Figure 54 Taxonomy for resource management .....	121
Figure 55 Connection sequence of RDP [55] .....	131
Figure 56 MCS connect initial PDU [55] .....	132
Figure 57 MCS connect response PDU [55].....	133
Figure 58 Multicast group.....	136
Figure 59 Flow chart of joining a multicast group.....	137
Figure 60 Flow chart of processing datagram.....	138
Figure 61 C++ codes of message structures used to store the receiving packet from the cluster.....	140
Figure 62 Run Linux sessions inside Windows .....	141
Figure 63 Compile rdesktop for Windows.....	143
Figure 64 Screen shot of notepad on local machine .....	144
Figure 65 Screen shot of notepad on remote desktop connection.....	145
Figure 66 Screenshot of seamless application .....	146
Figure 67 Command of seamless remote applications.....	146
Figure 68 Screenshot of opening more remote applications .....	147
Figure 69 Editing an RDP file .....	148
Figure 70 Remote accessing explorer.exe.....	149
Figure 71 Local (client) command window .....	150
Figure 72 A RDP file being edited by notepad.....	151
Figure 73 Windows remote desktop connection.....	152
Figure 74 Integrity test script.....	153
Figure 75 Main function to detect any discrepancies between the files in the client and server.....	154
Figure 76 Latency test script.....	155
Figure 77 Flowchart of research process .....	177

# CHAPTER 1 INTRODUCTION

---

---

## 1.1 Cluster Computing

### 1.1.1 *Definition*

A cluster is a type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers cooperatively working together as a single, integrated computing resource [1]. Cluster creates a single system image of resources from personal computers on a local area network, and offers high system availability and reliability through the redundancy of resources (e.g. hardware, operating systems and applications). There are many names for Cluster computing system including Clusters of Workstations (COW), Networks of Workstations (NOW), Workstation Clusters (WCs), Clusters of PCs (CoPs). The simplest hardware set up will be a few computers connected via the local area network which constitute a cluster workstation. Besides that, a middleware on the workstation cluster control the system behaviour of a distributed or parallel system and the software/application they support to run.

Cluster computing is based on low-end workstations and network technologies, which may not seem very useful at first. However, such systems have been the test-beds for a new computing era of high-performance and high-availability cluster computing. Technological advances in recent years made clustering systems burgeon. Because of the increasing performance of general purpose computer and emerging high speed communication, clustering becomes a promising research area in computer science and technology. It has become a popular topic of research among the academic and industrial communities including system designers, network developers, algorithm developers, as well faculty and graduate researchers [2]. Moreover, this class of system is becoming

more and more commonplace. Based on the survey, most academic institutions and industries have already start to use or are thinking of using clusters to run their most computation demanding applications instead of using high performance machines. Clusters become more and more attractive to companies who can even afford traditional supercomputers [3].

The terms “cluster computing” “cloud computing” and “grid computing” have been used almost interchangeably to describe networked computers that run distributed applications and share resources. All technologies improve application performance by executing parallel computations on different machines simultaneously, and enable the usage of distributed shared resources. They have been used to describe such a diverse set of distributed computing solutions that their meanings have become ambiguous. However, they represent different approaches in solving computation problems. Cluster computing aggregates the resources locally and shares the load, which form the base of all distributed computing paradigm. Cluster can contribute resources to Grid and Cloud. Grid computing is the extended version of cluster, in which resources are provisioned through internet. Cloud computing is “A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on” [4]. Therefore, on top of all, cloud provides almost the same functionalities as the above two systems. But it provides them in the form of services and bills which are the same as consuming utility.

### *1.1.2 Applications of Cluster Computing*

Clusters have been employed as a platform for a number of applications:

For scientific applications, clusters have been used in grand challenge or supercomputing applications, such as earthquakes or hurricanes prediction, weather forecasting, life sciences, computational fluid dynamics, nuclear



simulations, image processing, machine learning, data mining, astrophysics, complex crystallographic, micro-tomographic structural problems, protein dynamics, bio-catalysis, relativistic quantum chemistry of actinides, virtual materials design and processing, crash simulations, and global climate modelling. The use of clusters as computing platform is not just limited to scientific and engineering applications. [2] [5]

For the commercial applications, cluster can be best used in Internet and E-commerce as super-server, by putting together web server, ftp server, e-mail server, database server, etc. Other commercial applications include image rendering, network simulation, etc. Therefore, clusters can provide an excellent platform for solving a range of parallel and distributed applications in both scientific and commercial areas. [2] [5]

Clusters can also be used in big data applications to provide the storage and data management services for the data sets being analysed and computing resources required by the data processing tasks. A Hadoop cluster is a special type of computational cluster designed specifically for storing and analysing huge amounts of unstructured data in distributed machines. The Hadoop Data Processing Ecosystem is shown in Figure 1 Architecture of Hadoop Ecosystem below.

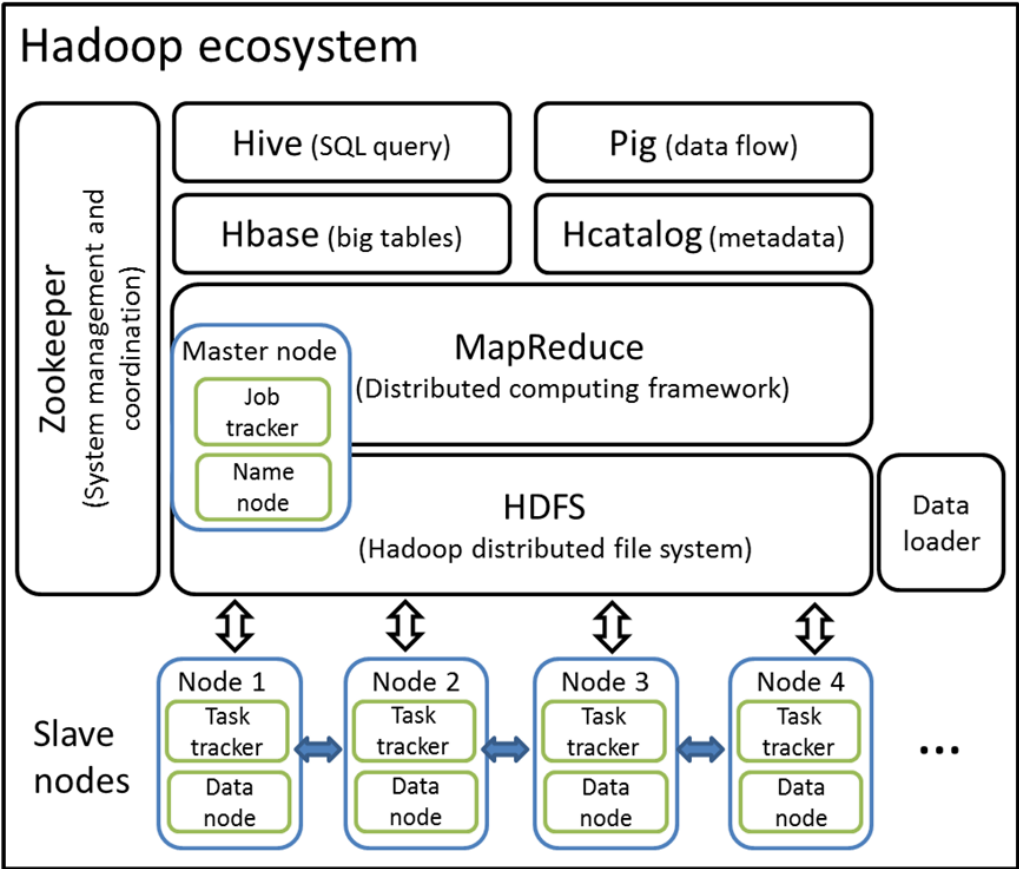
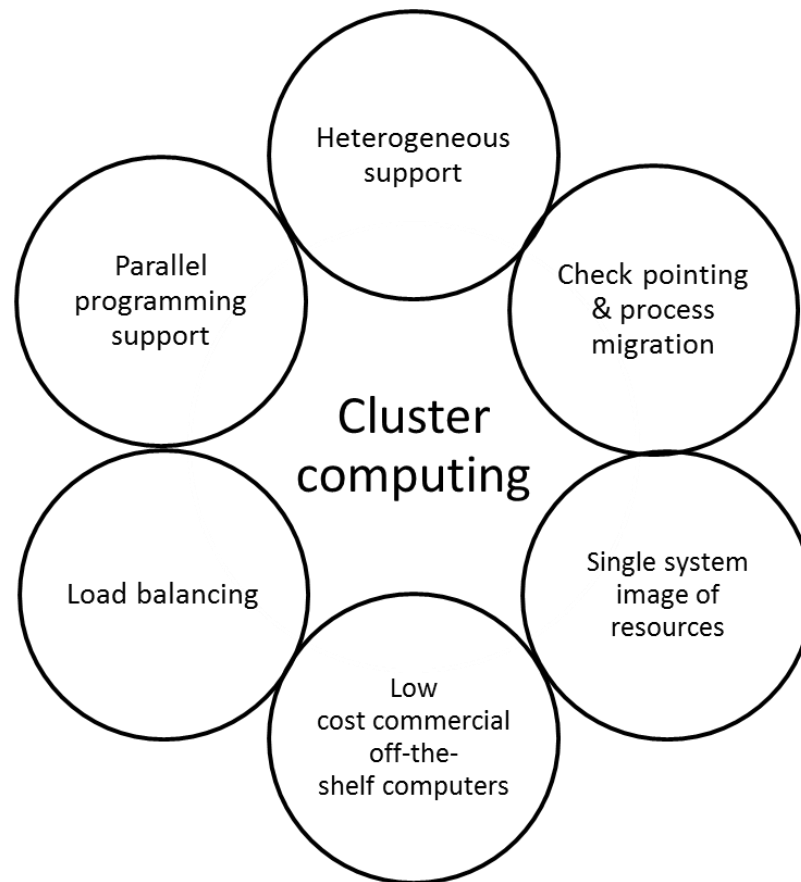


Figure 1 Architecture of Hadoop Ecosystem

1.1.3 Advantages and Disadvantages of Cluster Computing



*Figure 2 Why Cluster Computing?*

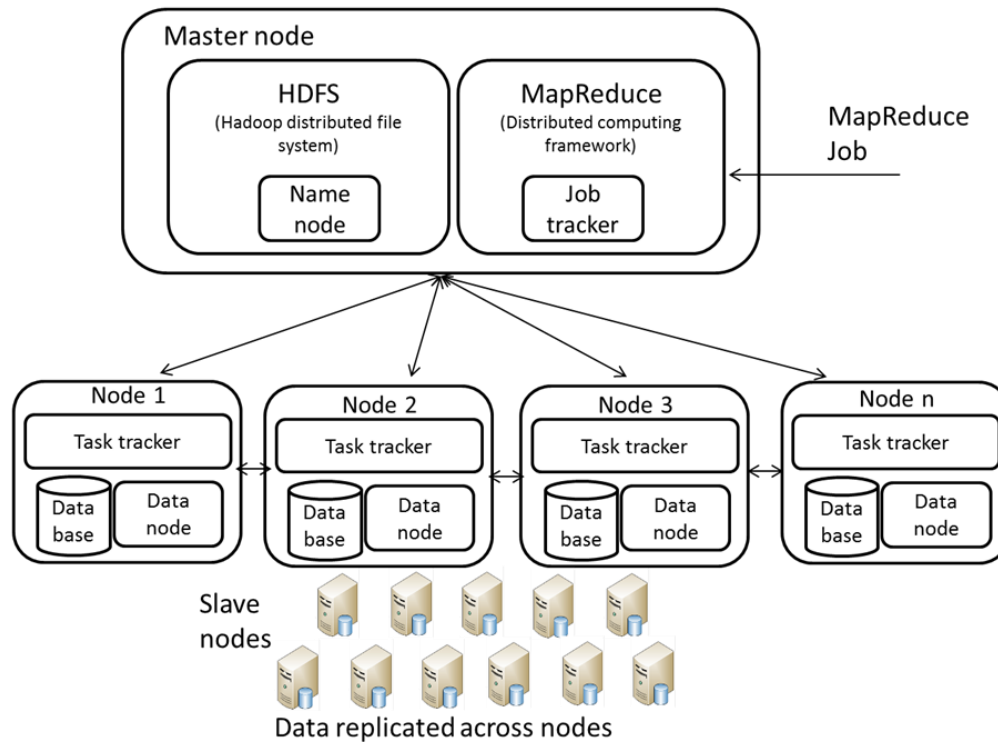
The reason of using clusters as a platform for high-performance (HP) and high-availability (HA) computing is mainly because of their cost-effectiveness and high scalability. Here is a summary of main advantages of cluster computing:

**Lower cost:** cluster owners/users can reduce the cost and complexity of purchasing, configuring and operating HPC clusters. The lower cost is achievable by using the shared computer resources in a cluster using different pricing strategies, e.g. on demand (pay-as-you-go), reserved or spot instances strategy.

**Scalability:** when the problem is complicated or the workload is large, a single system cannot process it due to time constraint. Clusters can provide an easier way to increase the computational resources. Based on the size and time requirements of workloads, users can add or remove compute resources to cater

their requirements. E.g. Apache Hadoop is an open source software project that enables the distributed processing of large data sets across clusters of commodity servers. Apache Hadoop for big data processing is designed to scale up from a single server to thousands of machines, with a very high degree of fault tolerance by using the Hadoop Distributed File System.

### Hadoop core



*Figure 3 Hadoop Core*

**Vendor independence:** It is good for cluster to be vendor independent, although it is in general advisable to use similar component across various servers in a cluster. A Linux cluster based on most commodity hardware allows for greater vendor independence than those using proprietary operating systems e.g. Windows. Recently, software releases have greatly improved on proprietary operating systems [6].

Reliability, Availability and Serviceability: because the redundancy of resources in the cluster, high reliability and availability can be provided. When one system is down, the user can switch his work to another machine with available resources. If it is a single machine being deployed when there is a major hardware or software component failure, the whole computational system will be brought down. In case of a cluster, a single component failure only affects a small proportion of the overall computational resources. Also, a system in the cluster can be powered off without bringing the rest of the cluster down. Also, additional computational resources can be added to a cluster while it is running the user workload. Hence a cluster maintains continuity of user operations in both of these cases. In similar situations a SMP (Symmetric multiprocessing) system will require a complete shutdown and restart. [7]Therefore, in terms of serviceability cluster provides better service than a single system in general.

Faster technology innovation: Clusters benefit from thousands of researchers around the world, who typically work on cluster of smaller systems rather than expensive high end systems [8].

There are a number of disadvantages that clusters have as compared to SMP's. Some of these challenges are described in the following paragraphs:

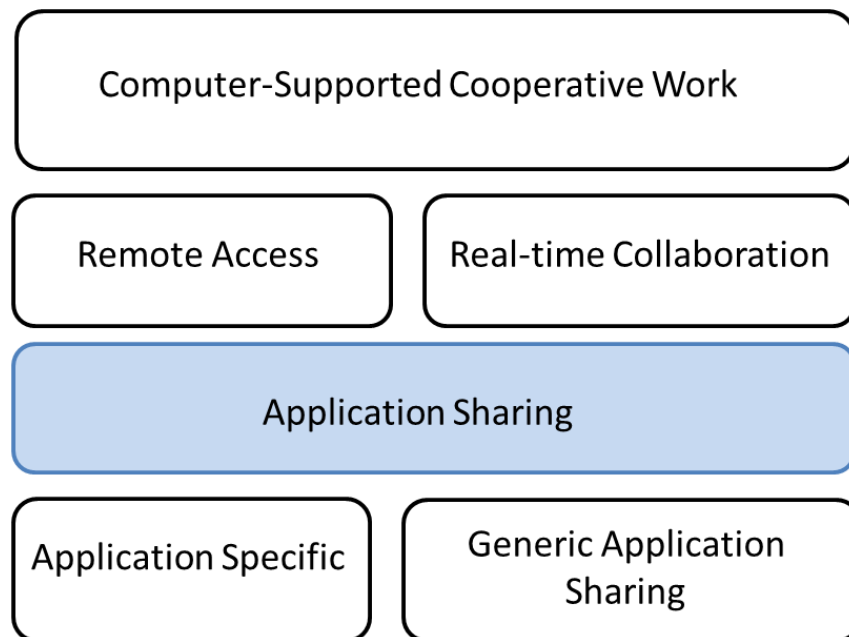
One of the challenges in the use of a computer cluster is the cost of administration. If the cluster has  $N$  nodes when  $N$  is large, the administration cost can be linearly increasing and becomes a serious concern [9]. The possible solution is a unified monitoring/reporting framework with data visualization support to simplify cluster administration [10].

Node failure management in clusters leads directly to the need to handle partial failures as compared to SMPs (i.e., the ability to survive and adapt to failures of subsets of the system). Traditional workstations and SMPs never face this issue, since the machine is either up or down. [10] When a node in a cluster fails, strategies such as "fencing" may be employed to keep the rest of the system

operational. [11] Fencing is the process of isolating a node or protecting shared resources when a node fails to function normally. There are two fencing methods: one disables a node itself and the other disallows access to resources provided by the node without powering off the node [9].

Task scheduling becomes a challenge when a large multi-tenant cluster needs to access very large amounts of data simultaneously. Also if the cluster is a heterogeneous cluster and a complex application environment the performance of each job depends on the characteristics of the underlying cluster. In this case, that is great challenge to map tasks onto CPU cores and GPU devices [11].

## 1.2 Application Sharing



*Figure 4 Taxonomy study on application sharing*

### 1.2.1 Definition

Application and desktop sharing (ADS) is the technologies and products that allow remote access and collaboration on a person's application or computer

desktop through a graphical emulator. Application sharing is different than desktop sharing in which there is only one shared application rather than sharing the entire desktop. For application sharing, there is only one copy of the shared application image running on the server. The key challenge is that some other application's interface window can sit on top of the shared application's window and also the shared application can open new child windows like Tools or Font. A true application sharing system should blank other applications if they are on top of the shared one and should transfer all the child windows of the shared application to the correct owner who are using this application.

### 1.2.2 *Application Specific v.s. Generic Application Sharing*

There are two kinds of applications sharing models: one is application specific and the other one is generic application sharing [12]. The application-specific model requires this sharing feature added to the applications specifically by the developers. For example, NetBeans an integrated development environment (IDE), Microsoft Office and many other applications have this sharing feature added. In order to have a sharing session all participants must have a copy of the shared application installed and running in their computer. In the generic application sharing model, the application is not specific meaning it can be any application such as PowerPoint, calculator, word processor, browser, or picture editor. Also, the participants do not have to install and run the application on their systems. Due to its generic nature the only disadvantage of generic application sharing may be the inefficiency as compared to the application-specific model in certain scenarios. ShAppliT (an application sharing tool in a cluster) has been developed based on the generic model; therefore, users can share any application without requiring the participants to have the application.

Application specific	Generic application sharing
<ul style="list-style-type: none"> <li>• Build-in feature to support specific application sharing, e.g. NetBeans IDE, Microsoft office</li> <li>• Participants must have applications installed and running in their local computer in order to share</li> </ul>	<ul style="list-style-type: none"> <li>• No build-in sharing feature in the application required</li> <li>• Participants do not need to install the application</li> <li>• Application sharing systems: BASS, MAST</li> </ul>

*Figure 5 Application Sharing Models*

### 1.2.3 Scenarios: Remote Log-in v.s. Real-time Collaboration

Among all the scenarios of application and desktop sharing, two scenarios are the most common ones that are “remote log-in” and “real-time collaboration”:

Remote log-in allows users to access to their own desktop even when they are not sitting in front of their computers. Some of the systems that support remote log-in are the Unix-based X Window System, Microsoft’s NetMeeting [13] and some products provided by VNC. Windows has this built-in solution by using the Remote Desktop Protocol (RDP) after Windows 2000 and prior to this version the systems have Microsoft’s NetMeeting. The open source products of VNC provide cross-platform solution for remote log-in.

Real-time collaboration is a bigger area of application and desktop sharing which allows sharing an application with remote users by multicasting the screen view to all the participants. Real-time collaboration is becoming more and more attractive in the area of rich multimedia communications. During the application or desktop sharing, all the users can see the same screen view and use the same application in a collaborative way where some of them can be in control mode and some of them can be in the view mode. Moreover, web conferencing is another application of desktop sharing by leveraging with multimedia communication technology



such as audio and video. Web conferencing creates a virtual space in which people can meet, socialize and work together.

#### 1.2.4 *Benefits and Challenges*

The greatest benefit of application sharing is that a remote user can run software that is not installed on his computer, even software that is not compatible with his operating system or that requires much more processing power than his computer can usually handle. This is because the remote user is not actually running the software on his computer, he is just viewing and controlling the desktop (and therefore the software) of the host computer. Through the use of application sharing software, it becomes possible for individual and organization to save huge sums of money they would have spent on rarely used, but essential software. Current computer technology trend is that hardware and connection cost decrease whereas the cost of the software is remaining high and becomes a larger fraction of the overall computing budget [14]. The diverging cost for software and hardware and the low usage of network and computer resources are the motivations of software/application sharing in a cluster.

From the research on related application sharing technology and products, a list of challenges are concluded. They are reliability, operating system independence, true application sharing, scalability and performance [12]. In an application sharing cluster, all the peers are independent and they may turn off their computer from time to time. Therefore, application and desktop sharing systems must be designed with reliability in mind. And the system should support heterogeneous operating systems because the participants in a sharing system could use different operating systems, e.g. Windows, Linux or Mac. Therefore, the application and desktop sharing system should be operating system independent. Scalability is another challenge when multiple users participate in application sharing or e-learning session. Research shows that systems with multicasting scales much better than unicast systems. Moreover, application sharing system should support

true application sharing where only the screen belongs to the user will be transmitted and viewed by the user. Some products provide more efficient transmission by only transmit the changed part to the user. They have better performance and utilization of resources. [12]

#### Why application sharing?

- By giving access to a larger body of users through one platform
  - ❖ Lower cost of ownership of software and hardware
  - ❖ Better return on investment for individual, family and organization
- Enable the user to run an application that is not installed in local machine
- Able to run applications in remote computer if it is not compatible with the local machine or requires more processing power
- Achieve easy and transparent scalability and maintenance
- Enable the user access multiple applications (in different host machines) or customized tasks/ workflows through a common platform

### 1.3 P2P Network System

Peer-to-peer (P2P) eliminates the one monopoly server and multiple clients' model and offers scalability and robustness due to its distributed nature. P2P computing aggregates computer resources from PCs connected by internet, including idle computing cycles, storage space, files and software applications. It is a new approach to establish a high performance computing system [15]. P2P systems can be classified into two different classes: structured P2P systems and unstructured P2P systems.

#### 1.3.1 *Structured P2P System*

In structured P2P systems, there are fixed connections among peers who maintain information about the resources (e.g., shared resources) that their neighbour peers have. Therefore, the data queries can be directed to the neighbour peers who own the desired data efficiently. Structured P2P systems enable efficient discovery of data. The most common indexing that is used to structure P2P systems is the Distributed Hash Tables (DHTs) indexing which stores a lookup service with (key, value) pairs. On one hand, any participating peers can efficiently retrieve the value associated with a given unique key. On the other hand, structured P2P network system leads to higher overhead.

### 1.3.2 *Unstructured P2P System*

In centralized peer-to-peer systems, a central directory server is used for indexing and bootstrapping the entire network system. A peer in the network sends the directory server of its IP address and the names of the contents that it makes available for sharing. Thus, the directory server knows which objects each peer in the network have, and then, creates a centralized and dynamic database which maps content name into a list of IPs. The main drawback of the design is that the directory server is a single point of failure. Moreover, when user request and data flow increase the directory server becomes bottleneck of the network.

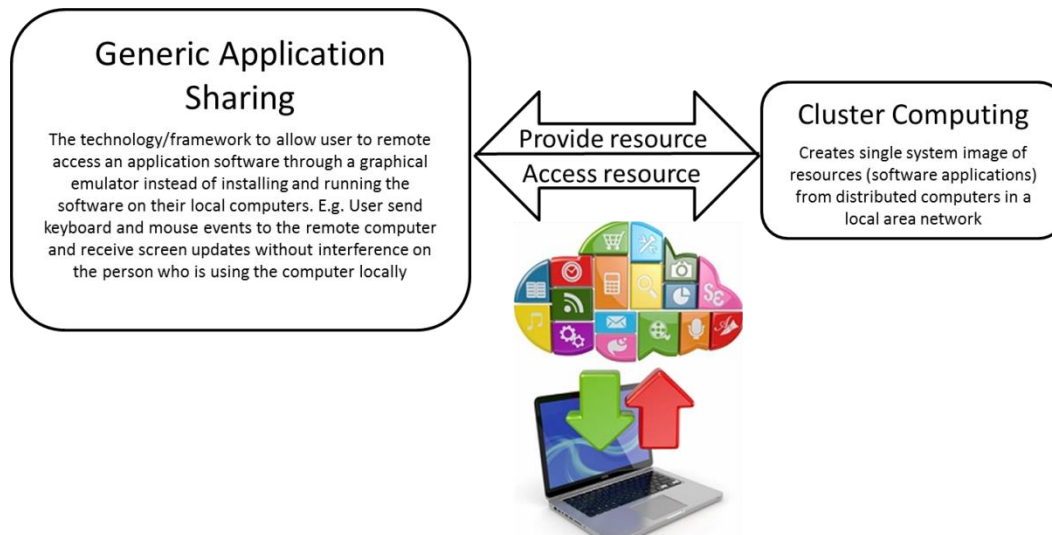
In pure peer-to-peer systems, TCP connections are maintained between any pair of peers. The peers in this network are aware only of their neighbour peers. Queries are sending by broadcasting or flooding. If a peer sends a query about a specific content interested in to its neighbours in the overlay network. Every neighbour will then forward the query to all of their neighbour peers. The drawback of the system can be the traffic in the network will reach its limit due to the broadcasting and flooding of information. And a peer may not be able to find the peer with the information if the information is rare.

Hybrid peer-to-peer system allows the existence of super node. This creates a hierarchical overlay network that addresses the scalability issues on pure P2P networks. The super-peer facilitates maintain a database that maps content to peer. However, hybrid P2P network system is more complicated as compared to centralized P2P system and pure P2P system. [16]

## 1.4 Research Problem and Scope of Work

### 1.4.1 Problem Statement

My aim in this research is to design and develop a novel P2P application sharing cluster architecture for generic application sharing in a cluster. There are two main concepts in this problem statement, namely generic application sharing and cluster computing as shown in the picture below.



*Figure 6 Definition of research problem*

To achieve generic application sharing, we provide a technique/framework for user to access and share generic applications/software with scalability, QoS and reliability in a P2P cluster. It allows applications to be remotely accessed by multiple users without interfering with other users or the user sitting at the

computer where the applications are installed, with special consideration to single user system (e.g. Windows). To achieve application sharing in heterogeneous cluster, we provide a methodology to support multiple users' access to computer system (not server) without modification of the proprietary OS.

## 1.4.2 *Sub-problems*

### 1.4.2.1 *Generic application sharing: extend single user application to multiple-user usage*

In general, software tends to be priced on the basis of the number of copies installed. So, if this software is essential, but only rarely used, the organization can decide to purchase a single copy of it, install it on a given computer and anyone who wants to make use of that particular application accesses it. In addition, the low usage of networks, personal computers and other computer resources are noticed as technology improves. Surveys show that the utilization of CPU cycles of desktop workstations is generally less than 10% [7]. So, general purpose computers are able to provide services and resources for others without adverse effect for themselves. Moreover, if application software serves their standalone machine, then users have limited reusability and limited ability to exploit the software capability within local area network. Therefore, this research is to establish a solution to extend single user software license to multiple user usage with seamless scalability and exploitation of the software with large group of users for better return of investment for companies or lower cost of ownership for individuals.

### 1.4.2.2 *Work on proprietary operating system*

A cluster environment may consist of heterogeneous operating systems including closed/proprietary operating systems and open source operating systems. A closed operating system is one where source code is not made available. Users may license the object code, but is not at liberty to modify or change. Examples of proprietary operating systems are Windows and Mac OS X. Open source operating systems allow the user to tweak and change. Examples of open source

operating systems are Linux for personal computers and Android for mobile devices. In the cluster environment, proprietary operating systems are in the consideration in design. By using this technique, only add-ons are provided to the systems but no modification of the source code is needed at the operating system level. For example, the client version of Windows is designed to be used by one person at a time and the terminal service also limits the number of users logged in to one at a time [17]. Two people cannot log on and access the computer system at the same time even if it includes just a physical, local-console login and a remote login. How to perform application sharing by allowing multiple users' access to proprietary operating systems is an important issue to be addressed in our research.

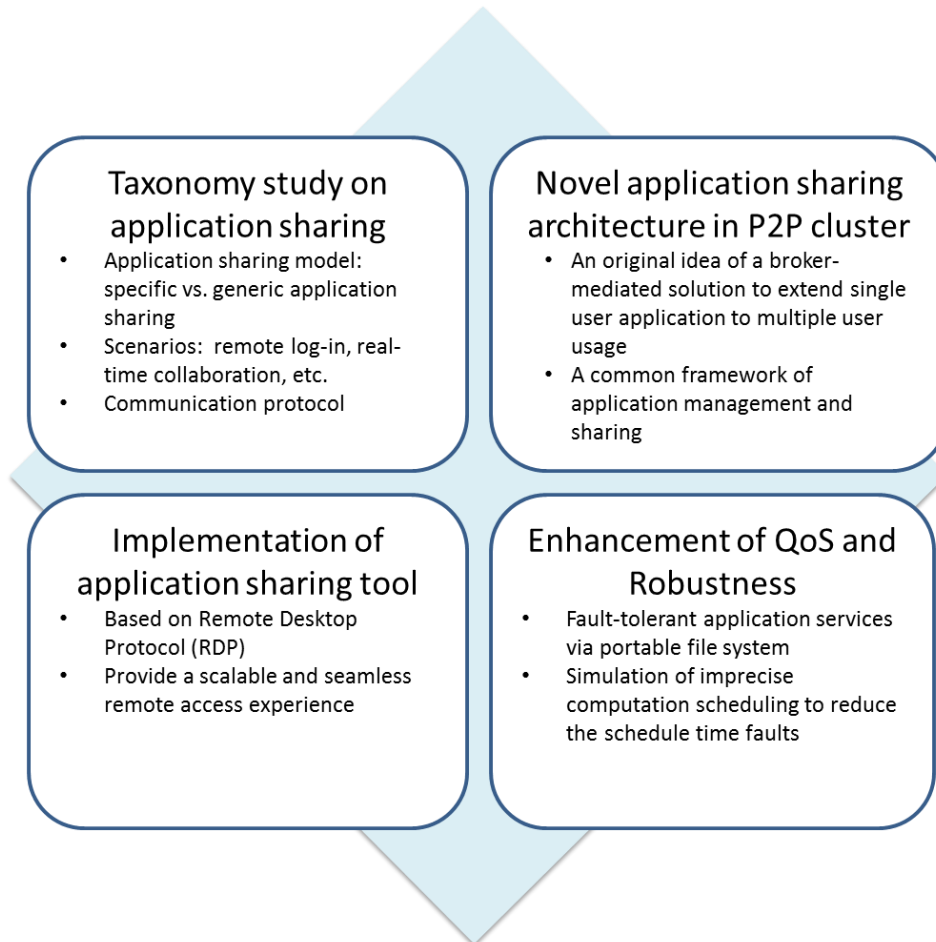
#### 1.4.2.3 *Fault tolerance of application services*

Real time applications are required to perform their functions under strict timing constraints. A task missing its deadline may cause other tasks to miss their deadlines resulting in a system failure. For real time applications such as image processing, the user may accept timely fuzzy and approximate results. Therefore, the imprecise computation workload model has to adjust the trade-off between computation time and result quality. Imprecise computation scheduling provides the solution to enhance QoS for real-time systems and improve the energy efficiency as well.

Besides, as a cluster is scaled up to large number of nodes and disks it becomes more risky that some components are working incorrectly at certain times. This leads the need to handle component failures gracefully and keep operating in the presence of failures. Due to the high possibilities of system and media failures, as well as the presence of user and application faults, hence this calls for a need to protect important file system data so that data loss can be minimized. A successful application sharing system should provide reliable services. A reliable file system need to be designed and implemented which enables user to login to the file server from anywhere, synchronizes document to last saved state on server and

provides certain degree of portability. Through this research, appropriate techniques need to be established for building a reliable file system to accomplish fault-tolerant application services.

## 1.5 Contributions



*Figure 7 Main Contributions*

This research has made the contribution to the field of application sharing in cluster computing by proposing a novel application sharing architecture for a cluster of closed operating system, building a reliable file system for fault-tolerant application services in clusters, simulation of imprecise scheduling to enhance

QoS for real-time computing system in enabling cost-effective and scalable high performance computing.

Firstly, in this research a novel application sharing architecture was proposed for generic application sharing in a standard local area network. This research is based on an original idea of a broker-mediated solution to extend single user application to multiple user usage. This framework has many benefits: resolving the problem of multiple users' access to proprietary operating systems, providing a common framework of application management, seamless updating of applications, allowing more users to exploit the applications in the cluster which leads to better return of investment. The objectives of our work were achieved through the implementation of a peer-to-peer application sharing tool called ShAppliT. ShAppliT is a middleware residing on top of the operating system. It implements a multiple-user and resource management protocol and provides a single client access to the underlying computer system. And it behaves like an agent to receive and manage tasks from multiple clients and provide a single client view for the server. Also, it allows applications to be remotely accessed by multiple clients without interfering with the person sitting at the computer where the application is installed. In addition, this architecture is based on Remote Desktop Protocol (RDP) to provide a scalable and seamless remote access experience. The user could feel as if he is working on the local computer despite working from a remote session.

Secondly, a failure-save solution has been designed and implemented for fault-tolerant application services in clusters which enabled user to login to the file server from anywhere, synchronize document to last saved state on server and provide certain degree of portability. The proposed idea of building a reliable file system was implemented successfully in this work. Upon the completion of the development of the file system, testing and evaluation of the system were also performed and results showed that the implemented has reached a reasonable level of reliability. In addition, through this implementation, appropriate



techniques have been established for the actual implementation of a reliable file system to accomplish fault-tolerant application sharing services in clusters.

Finally, imprecise computation scheduling was modelled and simulated to enhance QoS for real-time systems and improve the energy efficiency for large scale computing in clusters. Also four imprecise scheduling algorithms have been implemented and simulated namely earliest deadline first (EDF), rate monotonic scheduling (RMS), least execution time first (LEF) and most execution time first (MEF) under varying system workload from 0 to 100% loading. Measurements of simulation on a large number of task sets showed that imprecise computation improved the system reliability when scheduling intensive workloads with less schedule timing faults, CPU cycles and energy-efficiency improvement.

## 1.6 Thesis Outline

This thesis is structured as follows.

Chapter 2 surveys the literature on state of the art cluster computing technologies, application sharing solutions and communication protocols enabling application sharing.

Chapter 3 proposes a novel application sharing architecture for generic application sharing in a cluster of closed operating system.

Chapter 4 explains the design and implementation of a reliable file system for fault-tolerant application services. The latency test and integrity test of the file system were carried out.

Chapter 5 describes model and simulation of imprecise computation scheduling for large scale computation in cluster computing to enhance QoS for real-time systems and improve the energy efficiency.

Chapter 6 concludes the achievements of this research work and provides recommendations for future work.

## CHAPTER 2 RELATED WORK

---

### 2.1 Cluster Computing Solutions

Among the cluster computing solutions, some of their key features are listed out based on their technical reports or documentation. The combination of the features leads to the functionality and capability of the cluster system to meet a specific application's need. Next, each of the features will be discussed individually

#### 2.1.1 *Heterogeneous support*

Heterogeneous cluster is a cluster consists of different computing system architectures with different operating systems. For example, local area or campus-type networks consist of PCs using different operating systems, e.g. Windows, Linux, BSD or Mac. Beowulf Clusters [18] is a homogeneous cluster because it is a Linux-based cluster. Nowadays more cluster applications are built to support for a cluster consisting of heterogeneous operating systems. A success case is to combine coLinux with an openMosix enabled kernel to build a hybrid cluster [19]. coLinux is a new open source vitalization solution that lets you run a Linux kernel on top of a Windows kernel. openMosix is a cluster middleware which provides load levelling and transparent process migration. [19]

#### 2.1.2 *Parallel programming support*

Parallel Virtual Machine (PVM) and Message Passing Interface (MPI) are used by developers to exploit parallelism across computer systems with same or different architectures. Users are finding cluster systems with parallel support in these environments useful than those who do not have. Therefore, many vendors

and researchers are working on providing these capabilities and developing high performance parallel codes. The Beowulf project [18] initially begun at NASA's Goddard space flight centre, opened the door for low-cost, high performance cluster computing. In addition, standards and tools have been developed for distributed memory parallel computer systems and make it easier for programmers to build scalable and portable parallel computer applications. [20] A cluster of Beowulf uses parallel processing libraries including MPI and PVM in general. They allow the developers to divide a workload among a cluster of network connected computers and collect the processing results.

### 2.1.3 *Check-pointing*

Check-pointing is the technique to save the necessary application state for restarting it in case of failure. Checkpoint/restart is a mechanism for fault tolerance. Check-pointing has three possible implementation approaches: an application itself with built-in checkpoint/restart implementation, the user to link the application with a specific set of libraries that provide the check-pointing capability and run on a system which provides checkpoint/restart capability within the operating system. Condor's [21] implements process migration using checkpoint/restart for the Condor load balancing system. DMTCP (Distributed Multi-Threaded Check-Pointing) [22] is a transparent user-level check-pointing package for distributed applications. Check-pointing and restart is demonstrated for a wide range of over 20 well known applications including TightVNC [23], OpenMPI [24], MPICH2 [25] and python [26], etc.

### 2.1.4 *Process migration*

Process migration is closely related to checkpoint/restart. Process migration is to move process from one machine to another machine when there is a termination of the task execution on the original machine. In computer cluster, it is very common that application processes need to migrate to another machine due to

load balancing or failure during processes. Process migration and checkpoint/restart must both arrange to save all the process states including heap, registers, and stack of a process. The process states and the data must be stored and transmitted to the new machine environment for restarting. If the cluster environment is heterogeneous meaning the system environment is different from each other, then process migration is very complicated in this case. A middleware called M-JavaMPI [27] was developed to run on top of standard JVM to support transparent Java process migration and communication redirection to achieve load balancing.

### 2.1.5 *Load balancing*

Load balancing is the process of balancing the work load among the machines in the cluster to prevent some machine overloaded when some machines are idle. The load information of each machine is retrieved by a central server in charge of load distribution. Based on the load information of the cluster, the server is able to allocate and spread the load accordingly in the most computational efficient way. The changes of available processing and network resources in the cluster raise the strong need to make applications robust against the dynamics of cluster environments. There are two main techniques that are most suitable to cope with the dynamic nature of the cluster or grid: dynamic load balancing (DLB) and job replication (JR). In a reach article, they analysed and compared the effectiveness of these two approaches by means of trace-driven simulations. [28]

### 2.1.6 *Graphical user interface*

Many cluster systems supports a command line interface for user to access their environment. Command line interface is the basic feature to monitor, request and maintaining jobs on the cluster. While a graphical user interface (GUI) can significantly improve the productivity of cluster user especially who do not have professional skills in this area. By using GUI, more people are able exploit the

system. As a result, better return to investment can gain by making more users to access the system. For example, HP Insight Cluster Management Utility [29] graphical interface enables an easy view of the entire cluster, provides remote management and analysis, and allow quick software provided to all the nodes of the system [30].

## 2.2 Application Sharing Solutions

Application and desktop sharing enables remote administration, group collaboration, remote trouble shooting, e-learning, software tutoring and so on [14]. In the market, many remote control and desktop sharing solutions are available. The application sharing products use similar technology to implement. However the system design concepts are different. The differences are discussed on concept and philosophy of related solutions as compared with our proposed solution ShAppliT (see Table 1).

Table 1 Comparison of related work

Software Name	True Application Sharing	Support Closed OS	Peer-to-peer Architecture	Support Generic Application	No Modification of OS
TeleTeachingTool [31]	-	+	-	-	+
MAST [32]	-	+	-	-	+
Apple Remote Desktop [33]	-	+	-	+	+
GoToMyPC [34]	-	+	-	-	-
ThinLinc [35]	+	-	-	+	-
RealVNC [36]	-	+	-	-	+
BASS [14]	+	+	-	+	-
XenApp [37]	+	+	-	+	+
ShAppliT*	+	+	+	+	+

Microsoft has Windows Meeting Space for Windows Vista and Netmeeting for Windows XP. Netmeeting was released in 1999 for Windows 98; Windows Vista introduces an application sharing feature as part of Windows Meeting Space, but all the attendees must use Windows Vista. VNC [38] is a cross-platform open

source desktop sharing system but it supports only screen sharing. VNC supports multiple users but it lacks a floor control protocol. VNC uses a client-pull based transmission mechanism which performs poorly compared with server-push based transmissions under high round-trip time (RTT). SharedAppVnc [39] supports true application sharing, but the delay is on the order of seconds. It uses a loss codec and does not support multicast.

TeleTeachingTool [31] and MAST [32] use multicast in order to build a scalable sharing system. TeleTeachingTool is developed just for online teaching so it does not allow participants to use the shared desktop. Also, it does not support real application sharing. MAST (Multicast Application Sharing Tool) allows geographically distributed participants to share arbitrary legacy applications. MAST supports scalable group to group collaboration by using Multicast. It is being used within the eMinerals project to augment the Access Grid functionality. MAST allows remote users to participate via their keyboard and mouse but its screen capture model is based on polling the screen which is very primitive and not comparable to current state of art the capturing methods like mirror drivers. Although both TeleTeachingTool and MAST use multicasting for scalability, they do not address the unreliable nature of UDP transmissions. UDP does not guarantee delivery of packets. Even if the packets are delivered, they may be out of order. In order to compensate for packet loss, the TeleTeachingTool and MAST periodically transmit the whole screen which increases the bandwidth and CPU usage. In addition, they do not support real application sharing. When one user manipulates the application via keyboard and mouse events, other users receive the screen updates simultaneously.

X Window System [40] (also known as X11) is a computer software system and network protocol originally developed by MIT in 1984. X provides a basis for graphical user interfaces (GUIs) and rich input device capability for networked computers. It creates a hardware abstraction layer where software is written to use a generalized set of commands, allowing for device



independence and reuse of programs on any computer that implements x. Several x protocol multiplexors have been developed such as DMX, XMX, SharedX and CCFX [41]. Xrdp [42] is an open source remote desktop protocol (RDP) server with an x window desktop display to the user. It provides Linux terminal server, connections from rdesktop and Microsoft's terminal server or remote desktop clients. Xrdp uses Xvnc or X11rdp to manage the X session. Xrdp project is released under the GNU Public License (GPL).

BASS [14] is an application and desktop sharing platform which allows two or more people to collaborate on a single document, drawing or project in real-time. BASS supports all application due to its generic model. However, BASS is developed on Windows XP server and the server is modified by adding a mirror driver (see Figure 8). In addition, BASS is based on client-server system architecture which is different with our peer-to-peer cluster computing application sharing system where a peer can be client and server at the same time. And also there are no modifications of the OS at all. The solution proposed is to add a broker middleware on top of the Windows OS of personal computers, instead of Windows server.

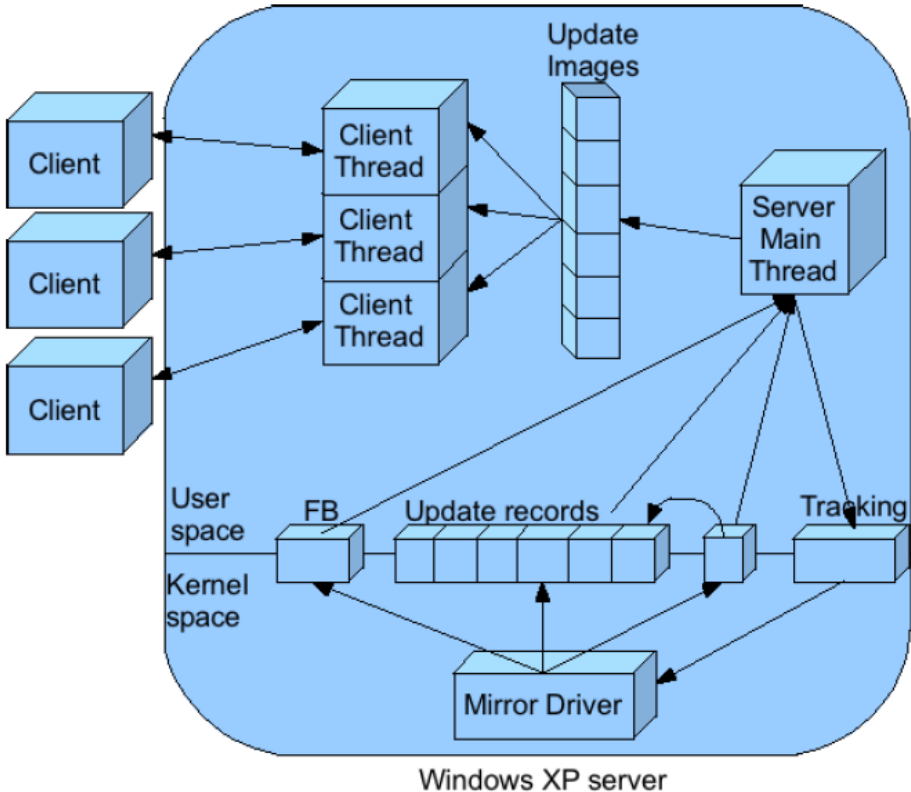


Figure 8 Windows XP server architecture [14]

In the Table 2, a comparison is made among the state of the art application sharing solutions focusing on the communication protocol used, the creator and licence group they belongs to, such as proprietary license or GPL.

Table 2 Comparison of application sharing solutions

Software name	Protocol used	Creator	Release date	License
Apple Remote Desktop [33]	RFB(VNC)	Apple	2002	Proprietary
Cendio ThinLinc [35]	RFB(VNC)	Cendio AB	2003	Proprietary

Chrome Remote Desktop [43]	Chromoting	Google	2011	BSD
Citrix XenApp [37]	RDP, ICA	Citrix Systems		Proprietary
RealVNC Enterprise [36]	RFB(VNC)	RealVNC	2002	Proprietary
Remote Desktop Services/Terminal Services [44]	RDP	Microsoft	1998	Proprietary
Ericom Blaze [45]	RDP	Ericom Software	2009	Proprietary
GoToMyPC [34]	Proprietary	Citrix Online	2000	Proprietary
N-central [46]	RDP, VNC, Proprietary	N-able Technologies	2011	Proprietary
RapidSupport [47]	RFB(VNC)	Tech Dimension	2012	Proprietary
Team Viewer [48]	Proprietary	Team Viewer GmbH	2005	Proprietary
UltraVNC [36]	RFB(VNC)		2005	GPL
Xrdp [49]	RDP			GPL

## 2.3 Communication Protocols for Application Sharing

The application sharing protocol enables multipoint computer application sharing by allowing a view onto a computer application executing at one site to be advertised within a session to other site(s). There are many communication protocols defined by different vendors or organizations, such as RFB [50] (Remote Frame Buffer) for VNC, RDP for Windows Terminal Service, and ITU-T T.128 [51] for NetMeeting and SunForum [52]. In general, most communication protocols used in application sharing are similar in terms of the functionality they offer. However, these protocols can be differentiated by the way the implementation of system layer where all the redirection of graphical output and user input take place. This is also the key component that determines the speed and quality of a remote desktop protocol. Some protocols compress the graphical images for transmission while other uses kernel level driver for transmission. There were two ways to implement application sharing systems. The difference is the transmission of screen contents or drawing commands [53]. In the following section, RFB protocol, RDP protocol and ITU-T T.128 will be discussed and compared to identify the key differences that separate them.

### 2.3.1 *Remote Frame Buffer (RFB) for Virtual Network Computing (VNC)*

RFB is a simple protocol for remote access to graphical user interface that function at the frame buffer level [50]. Therefore, it is highly versatile and applicable to applications and systems across different platforms and operating systems. As for the display side of the protocol, a low level primitive graphics concept has been applied. The data containing the graphical display information at the pixel level such as coordinate and image block of a particular group of pixels are compressed and transmitted regularly from the server to the client. In another words, the update of a display screen consists of a series of frame buffer updates that refresh the display screen block by block. The way this concept works is

similar to how video frames refresh. Virtual Network Computing (VNC) was originally developed by at the Olivetti Research Laboratory in Cambridge, United Kingdom [38]. It is a graphical sharing system that uses RFB protocols. Many VNC source code available nowadays are open sources under the GNU General Public License. The most popular implementations of VNC available in the market are RealVNC and UltraVNC [36].

### 2.3.2 *Microsoft Remote Desktop Protocol (RDP)*

RDP provides remote display and input capabilities over network connections for Windows-based applications running on a server. RDP is designed to support different types of network topologies and multiple LAN protocols. RDP is an extension of the ITU-T.128 application sharing protocol developed by Microsoft [54]. Basic connectivity and graphics remoting is designed to facilitate user interaction with a remote computer system by transferring graphics display information from the remote computer to the user and transporting input commands from the user to the remote computer, where the input commands are replayed on the remote computer. RDP also provides an extensible transport mechanism which allows specialized communication to take place between components on the user computer and components running on the remote computer including RSA Security, bandwidth reduction features, roaming disconnect, clipboard mapping, print redirection, virtual channels, remote control and network load balancing. This proprietary protocol provides a mean to access the graphical interface of a remote host computer. Similar to other remote desktop applications, the processing of a running application is being done in the host computer, only the graphical presentation of the desktop is being transmitted to the client. However, as compared to VNC, RDP provides a faster remote access speed [44]. This is due to the fact that RDP hooks deeper into Windows API to optimize the information required by the client to construct the display screen. For example, while VNC is transmitting blocks of bitmap for the client to construct a

display screen of a text document, RDP transmits the texts in the document itself for the client to render a display screen.

### 2.3.3 *ITU-T T.128. Multipoint Application Sharing*

T.128 is accepted by the ITU, Telecommunication Standardization Sector (ITU-T) [51]. T.128 specifies the program sharing protocol, defining how participants in a T.120 conference can share local programs. Figure 9 presents an overview of the scope of T.128 and its relationship to the other elements of the T.120 framework within a single node.

Specifically, T.128 enables multiple conference participants to view and collaborate on shared programs, and it is the foundation for RDP. The T.128 protocol supports multipoint computer application sharing by allowing a view onto a computer application executing at one site to be advertised within a session to other sites. Each site can, under specified conditions, take control of the shared computer application by sending remote keyboard and pointing device information. This style of application sharing does not require and does not make provision for synchronizing multiple instances of the same computer application running at multiple sites. Instead, it enables remote viewing and control of a single application instance to provide the illusion that the application is running locally. A multichannel-capable protocol allows for separate virtual channels for carrying presentation data, serial device communication, licensing information, highly encrypted data (keyboard and mouse activity), and so on [55].

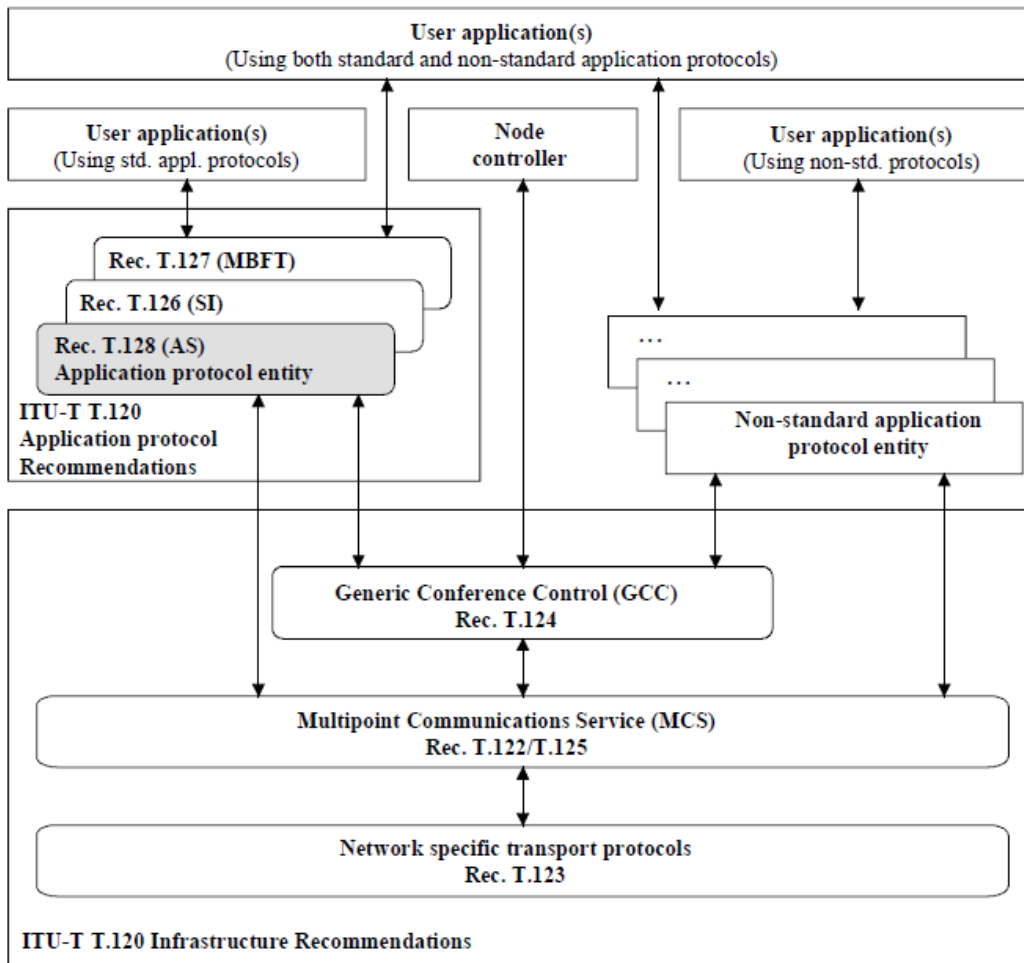


Figure 9 Multipoint application sharing protocol T. 128 and its family [54]

# **CHAPTER 3 A NOVEL BROKER-MEDIATED SOLUTION TO GENERIC APPLICATION SHARING IN A CLUSTER OF CLOSED OPERATING SYSTEMS**

---

---

## **3.1 Introduction**

With advances in hardware and networking technologies and mass manufacturing, the cost of high end hardware has fallen dramatically in recent years. However, software cost still remains high and is the dominant fraction of the overall computing budget. Application sharing is a promising solution to reduce the overall IT cost. Currently software licenses are still based on the number of copies installed. An organization can thus reduce the IT cost if the users are able to remotely access the software that is installed on certain computer servers instead of running the software on every local computer

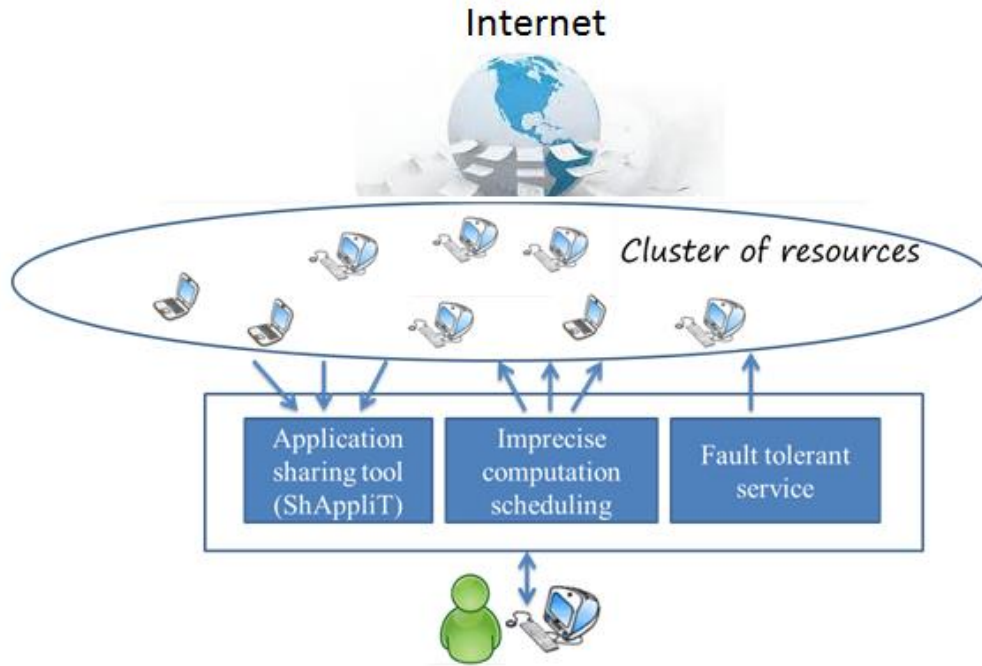
Application sharing is a promising solution to effectively reduce the overall cost of computing. The greatest benefit of application sharing is that software can be remotely used by the users from their local computers which may have incompatible operating system and lower processing power required by the software. This is because the users are not actually running the software on their local computer, but remotely accessing and controlling the desktop (and therefore the software) of the host computer. With the use of the application sharing software, it is possible for individuals and organization to save huge amount of money that they would have spent on purchasing more copies of software to cater for all of the local computers.



With increasing performance of general purpose computer and high speed communication, cluster computing is becoming a promising research area. A cluster environment may consist of heterogeneous operating systems including closed/proprietary operating systems and open source operating systems. A closed operating system is one where source code is not made available. Users may license the object code, but is not at liberty to modify or change. Examples of proprietary operating systems are Windows and Mac OS X. Open source operating systems allow the user to tweak and change. Examples of open source operating systems are Linux for personal computers and Android for mobile devices. In the cluster environment, proprietary operating systems are in consideration in the design. Add-ons are designed to these systems but no modification of the source code at the operating system level. For example, the client version of Windows is designed to be used by one person at a time and the terminal service also limits the number of users logged in to one at a time [56]. Two people cannot log on and access the computer system at the same time even if it includes just a physical, local-console login and a remote login. How to perform application sharing on such a proprietary operating system is an important issue to be addressed in our research.

A novel application sharing architecture is proposed in this thesis for generic application sharing in a standard local area network. A broker-mediated solution is designed to extend single user software license to multiple user usage and resolve the problem of multiple users' access to proprietary operating systems. The objectives of our work are achieved through the implementation of a peer-to-peer application sharing tool called ShAppliT. ShAppliT is a middleware residing on top of the operating system. It implements a multiple-user and resource management protocol and provides a single client access to the underlying computer system. ShAppliT have been implemented based on Microsoft Windows operating system.

### 3.2 System Overview



*Figure 10 System overview*

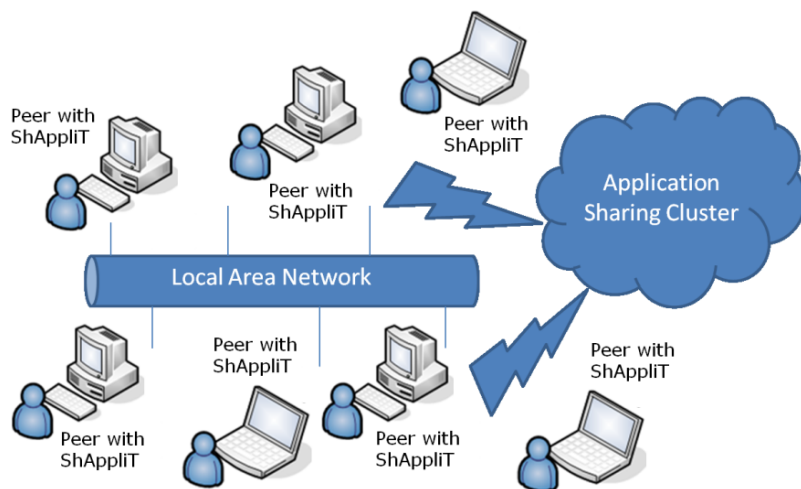
A cluster creates a single system image of resources from personal computers on a local area network, and offers high system availability and reliability through the redundancy of resources (e.g. software/applications, CPU cycles and hard disk). In our current application sharing cluster computing system, the technique is provided (ShAppliT in Figure 10 System overview) to coordinate multiple users' assessments for closed system using a broker-mediated mechanism. This application sharing system aims for sharing of application/software resources with general applicability and scalability. A novel application sharing architecture is introduced for generic application sharing in a cluster of closed operating system. More details will be presented in the rest of sections in Chapter 3.

The peers in the cluster are unreliable. A successful application sharing system should provide reliable services (see Figure 10 System overview). One chief

technology to accomplish fault-tolerant application services is data replication at client or server or third peer. A failure-save solution for fault-tolerant application services in clusters enables user to login to the file server from anywhere, synchronize document to last saved state on server and provide certain degree of portability. A reliable file system for fault-tolerant application services will be presented in Chapter 4.

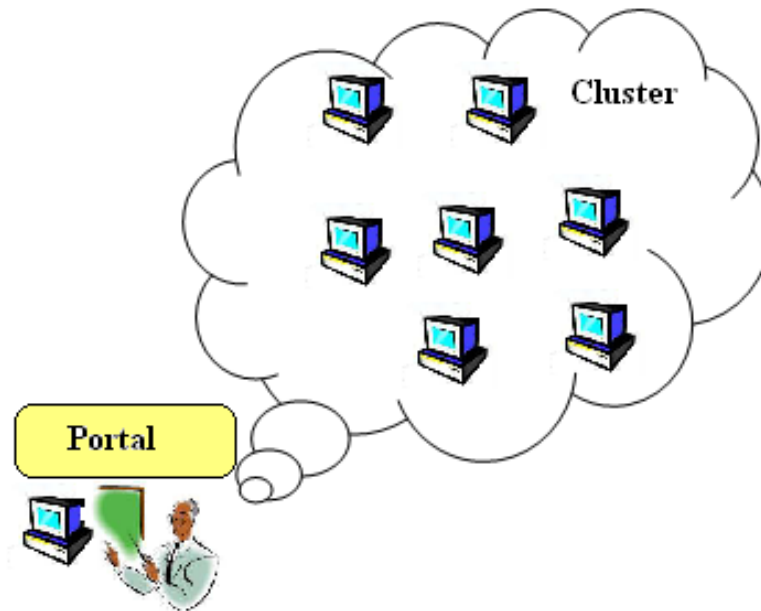
In addition, cluster computing has attracted attention for large scale computing using idle CPU cycles of personal computers connected in local area network. In this thesis, a broker with imprecise computation scheduling is proposed for large scale computing in clusters (see Figure 10 System overview). Model and simulation of imprecise computation techniques are carried out for scheduling flexibility by trading off result quality to meet computation deadlines. This technique is to enhance QoS for real-time systems and improve the energy efficiency for large scale computing in clusters. It will be described in details in Chapter 5.

### 3.2.1 System Architectures



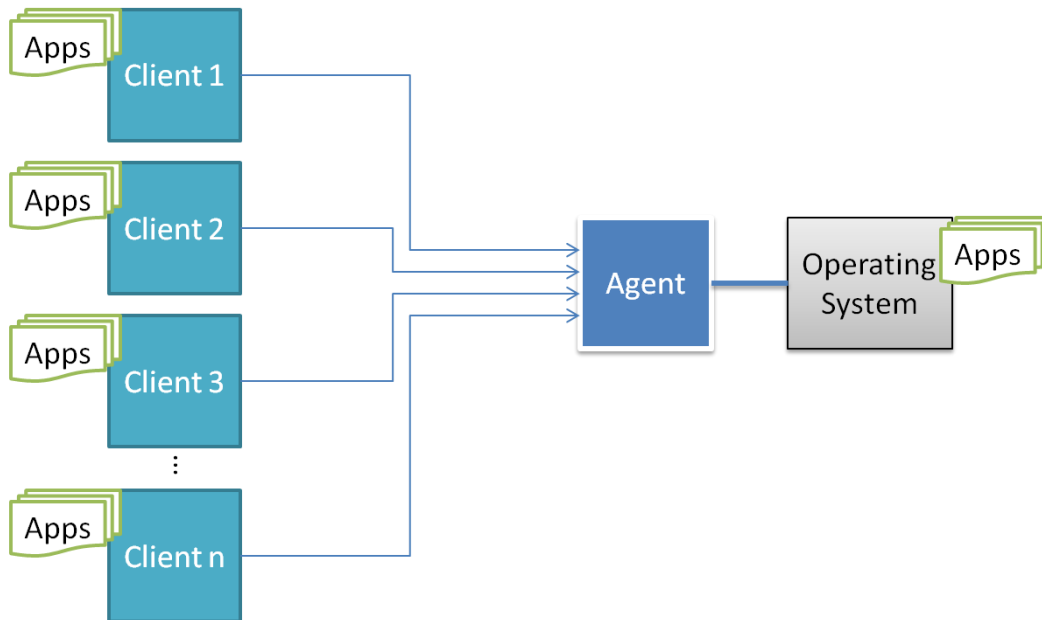
*Figure 11 Application sharing cluster overview*

As shown in Figure 11, each node with ShAppliT in the cluster is called a peer. All the peers are equal among each other, meaning it can act as an application provider (server) or/and as an application consumer (client). All the computers are connected via a high speed local area network. Computers with ShAppliT installed form a cluster network within the LAN to facilitate handshaking, message exchanging and remote desktop connections that are exclusive for ShAppliT users.



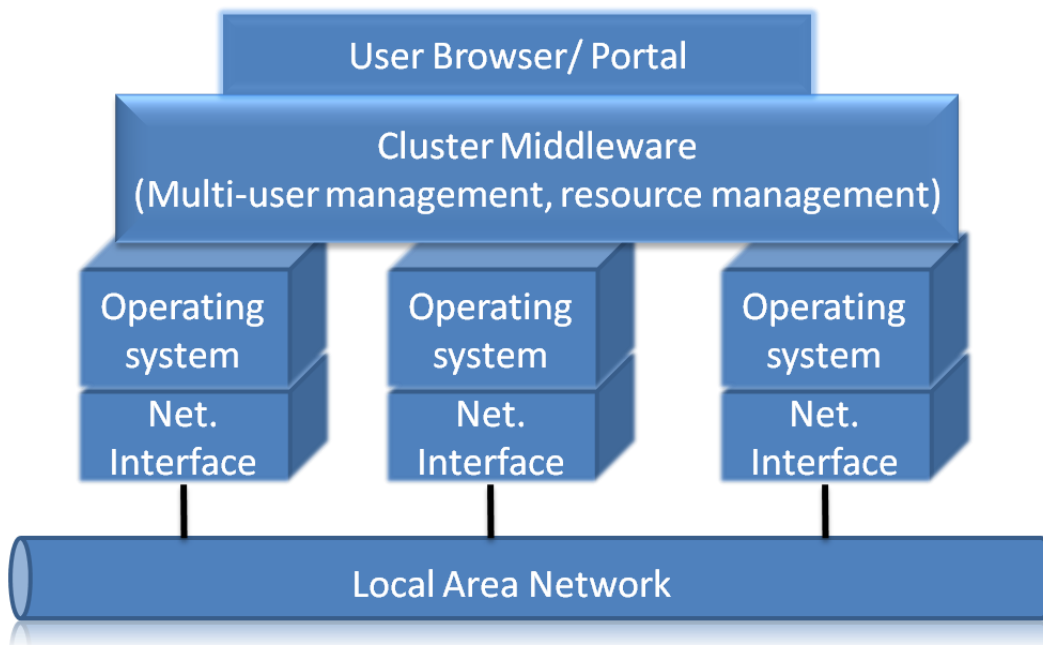
*Figure 12 Access shared application resources in a cluster*

In Figure 12, each user sees the Cluster as a single system image of the resources sharable in the cluster, in this case the software/application resources. The user may choose any application to launch via a thin client portal e.g. the browser or software plugins. The application will be executed in the remote computer and the program display will be shown at the client's desktop. A peer in the cluster can act as client to search and use applications shared by other peers in the network through remote access. And a peer who acts as a host/server can opt and provide application for sharing.



*Figure 13 Illustration of system architecture*

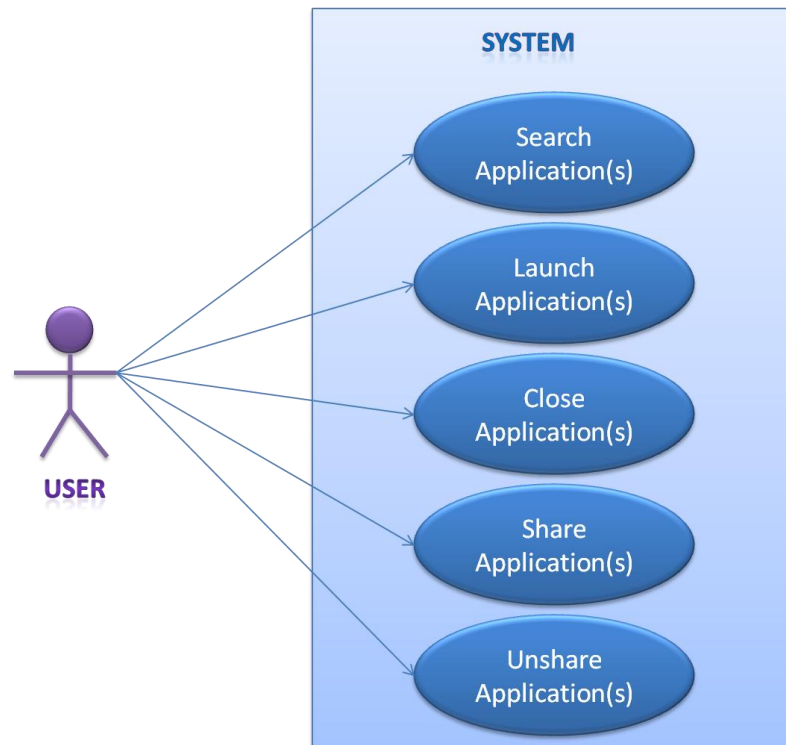
There is a layer on top of all operating systems for multiple user and resource management. It works as an agent/broker to receive request from multiple users and manage the session for each user and only have one access to the operating system, refer to Figure 13. The operating system, together with the underlying applications and resources fulfil the agent/broker's requests. Our application sharing tool (ShAppliT) acts as the bridge between the clients and the server. Only one master session logs in to the application server and accesses the host Windows OS via terminal service. All the tasks are received by the broker from multiple clients, both remote and local computer users. Therefore, the server sees only one remote desktop session and does work for the agent/broker only. The agent/broker takes over the responsibility of negotiation with remote clients, forwards the input events to the server OS and redirects the display data back to the respective clients. In a way it shares a single-user application among multiple clients via a single log in to that application.



*Figure 14 Layered architecture of a cluster system*

As shown in Figure 14, there is a layer on top of all operating systems for multiple user management and resource management. It works as an agent/broker sitting in between clients and server to receive request from multiple users and manage the session for each user and provide only one access to the server operating system. The operating system is the actual worker to do all the tasks.

### 3.2.2 Use Case Diagram



*Figure 15 Application sharing use cases diagram*

Figure 15 is a use case diagram that elaborates the interaction between a user and App Share system. A user is able to perform five actions using App Share, searching for an application across the network, starting an application using App Share Client, ending an application, setting an application for sharing with peers in the network and removing an application for sharing with peers from the network.

The basic course of events when a user opens App Share is as following; assuming that the user is Alice and the peer in the network is Bob. They both have App Share running:

1. User Alice starts App Share
2. User Alice can choose whether to share/un-share a particular application.

3. Alice searches for an application
4. Alice's App Share will broadcast the request to all hosts in the cluster network through IP multicast
5. Bob's App Share receives request from Alice. If all conditions are met, he will fulfil the request by broadcasting the required information into the network.
6. If Alice's App Share receives Bob's reply, App Share will start remote application initialization with Bob's server; subsequently enter the maintenance state of remote application connection.
7. Bob's App Share server redirects the data stream from Alice to his Microsoft Terminal Service and streams the display data back to Alice's App Share client for display.
8. User Alice can close an application to terminate a particular remote application session.

### 3.3 Design and Methodology

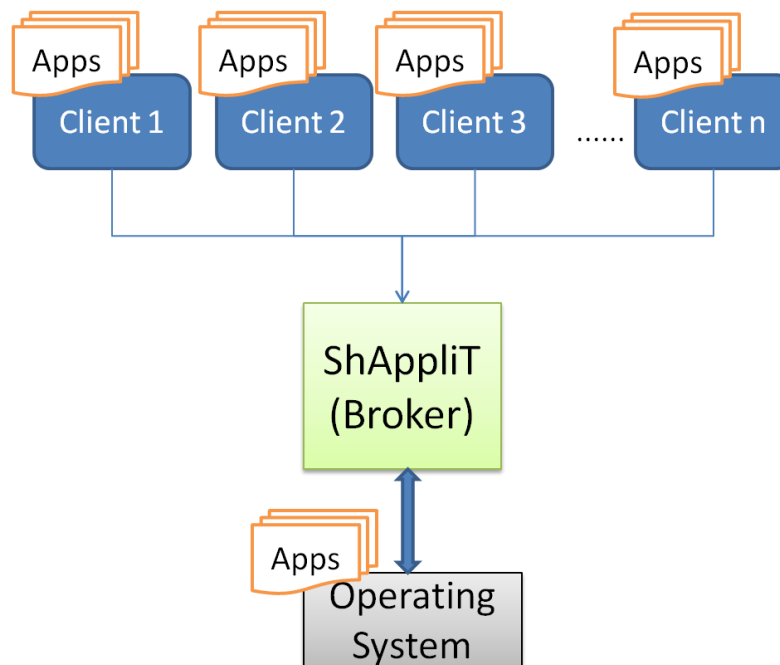
Unlike Linux which is a multi-user system designed to handle multiple concurrent users, Windows client systems are designed to be used by one person at a time [17] [57]. Windows XP is typically used by standalone users whereas Windows Server 2003 is normally deployed as a server operating system built to support multiple clients concurrently. However, Windows Server 2003 contains complex functionality and is mainly operated by programmers or administrators and it is many times costlier than XP, which make Windows Server 2003 not desirable for peer to peer usage. Since Windows XP is a single user operating system, it is an obstacle to the realization of peer-to-peer application sharing.

ShAppliT is divided into three parts:



- Establishment of multiple remote application sharing sessions
- Initialization and management of a cluster
- Incoming and outgoing packet management

### 3.3.1 *Establishing Multiple Remote Application Sessions*



*Figure 16 Broker mediated application sharing system architecture*

A broker-mediated solution is proposed and provided to extend single user software license for multiple-user usage and solve the problem of working on closed or proprietary Operating Systems.

Instead of managing multiple connections using Windows terminal service server, ShAppliT which sits in between the client and Windows TS server as a broker. It handles tasks from multiple clients and passes them to the TS server. Therefore, TS server sees only one Remote Desktop Protocol (RDP) session and does work for the ShAppliT only. And ShAppliT takes over the responsibility of negotiation with remote clients, forwards the input events to TS server and redirects the display data back to the respective clients.

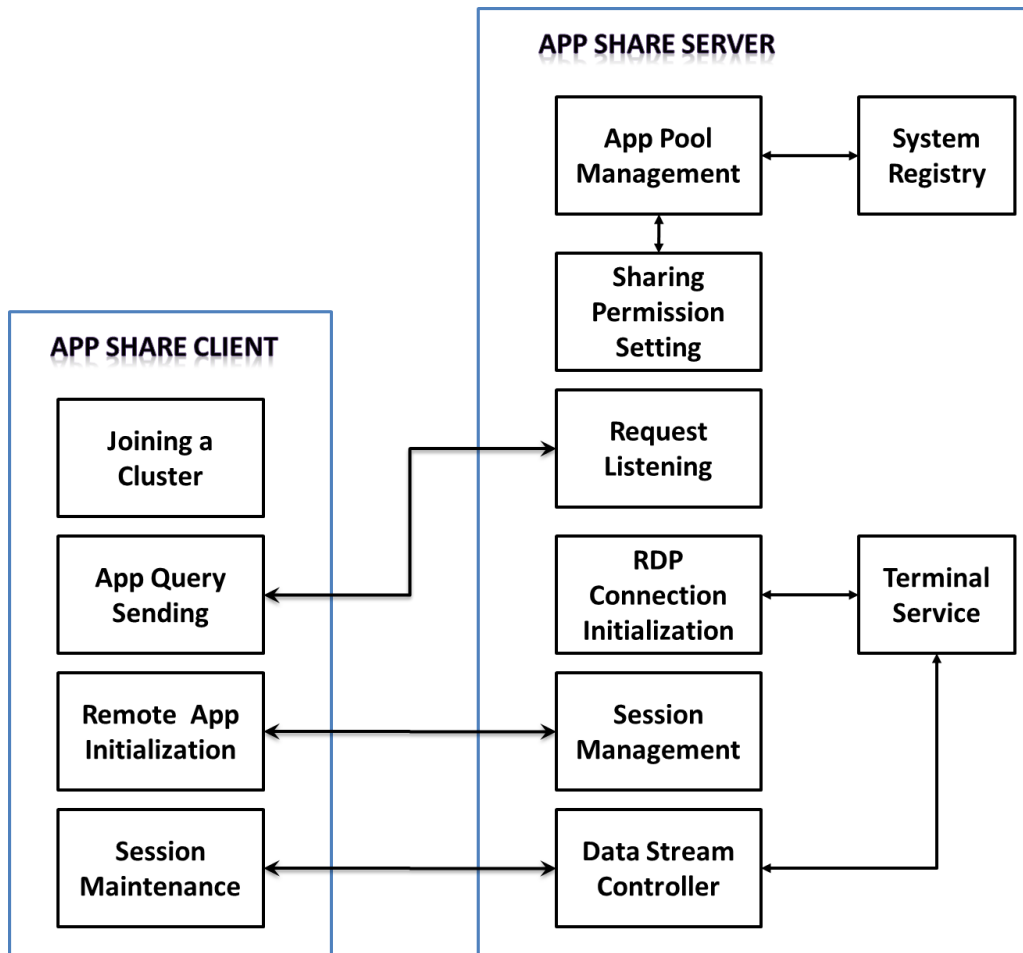


Figure 17 System architecture model of ShAppliT

Figure 17 shows the design system architecture model of ShAppliT. The left block is the App Share client that consists of the Cluster joining component, Query sending component, Remote session initialization component and Session maintenance management. The right block is the App Share server that consists of the Application pool management, Request listening component, RDP connection initialization component, Session management and Data stream controller.

The details of each component are described as follows. Sharing Permission Setting component allows the user to configure which applications to be offered for sharing via the Application Pool Management. Query Sending is capable of creating a query for application. Request Listening has an open port listening to

the requests broadcasted in the cluster. Request listening periodically processes the requests in the list by verifying whether all the relevant conditions are met. When all conditions are met, the App Share Client will launch a remote session. In the initialization phase, App Share client establishes a remote connection session with session manager in App Share Server. User session is an abstract venue on an App Share Server that is assigned to a user. Once the user session moves to an established state, user interacts with the server and applications from within this venue.

In the communication phase, keyboard and mouse input events are sent from the App Share Client to the remote endpoint on the App Share Server while graphic update data are received from an established graphics channel and is sent to the display adapter of the App Share Client. In App Share Server, Data stream controller is in charge of multiplexing and de-multiplexing the clients' and server's traffic. It maintains the smooth execution of multiple remote user sessions of App Share system.

### 3.3.1.1 App Share Client State Model

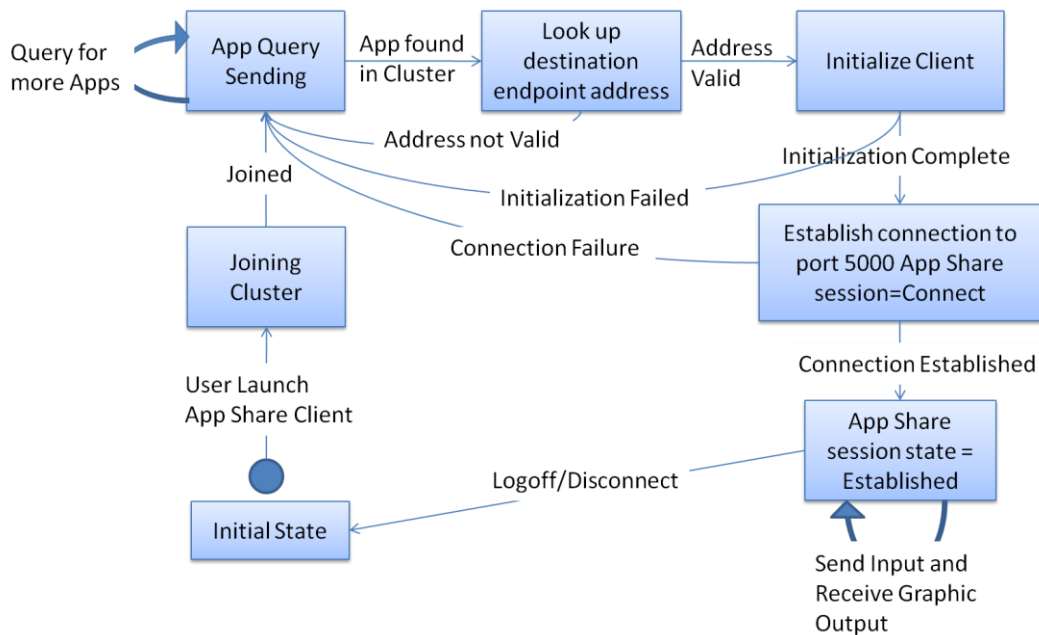


Figure 18 State diagram of App Share Client during connection sequence

The App Share Client state model for a basic connection scenario is illustrated in Figure 18. In this scenario, an App Share Client connects to an App Share Server in an intranet environment. The high-level state diagram that follows shows the connection states as the App Share Client transitions from an initial state to the state of an established connection.

After the App Share Client has joined the cluster, the connection process continues as follows:

1. The App Share Client acquires the destination IP address of the App Share Server by search an application in the cluster.
2. The App Share Client initiates the sequence to establish a Remote Desktop Protocol (RDP) connection as described in [MS-RDPBCGR] with App Share Server port 5000, starting with an X.224 exchange. [55]If the connection attempt fails due to authentication issues, the flow reverts to the state “Query for application” as shown in the following figure.
3. If the X.224 exchange is successful, the App Share Client supplies capability and license information to the App Share Server.
4. Once the license is validated, the user session moves to an established state. User session is an abstract venue on an App Share Server that is assigned to a user. The user interacts with the server and applications from within this venue.
5. While in this state, keyboard and mouse input is sent from the App Share Client to the remote endpoint on the App Share Server while graphics data is received from an established graphics channel and is sent to the display adapter of the App Share Client.
6. In addition, in the established state more applications can be spawned at the same sever by using slave mode of App Share Client. In slave mode, a command

of application will be sent to the master socket of App Share Client and then passed to App Share Server to spawn a new application.

### 3.3.1.2 App Share Server State Model

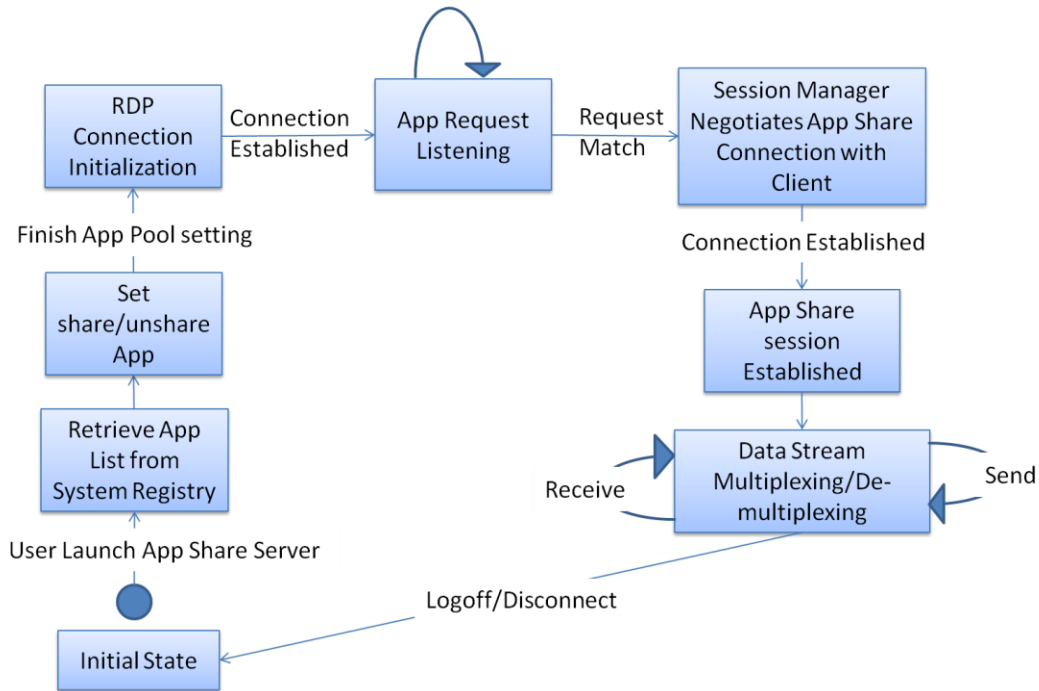


Figure 19 State diagram of App Share Server during connection sequence

The App Share Server state model for a basic connection scenario is illustrated in Figure 19. In this scenario, an App Share Server establishes one connection to TS Server on Windows OS, receives connections from remote clients and maintains the remote sessions. The high-level state diagram that follows shows the connection states as the App Share Server transitions from an initial state to the state of an established connection.

After the App Share Server has finished its internal initialization, the connection process continues as follows:

1. A registry crawler searches through the system registry to track all the applications installed in the host computer and generate a list.

2. Sharing permission setting component allows the user to configure which application to be offered for sharing and after the setting a sharable application pool is formed.
3. The App Share Server establishes one RDP connection to TS Server on Windows OS.
4. An App Share Server starts listening for an incoming connection request after initialization of RDP connection to TS Server on Windows OS.
5. After one RDP connection established, when an App Share Client attempts to establish a connection with App Share Server, the App Share Server starts processing the request by going through a sequence of steps.
6. If the user's application request matches, session manager negotiates the connection with App Client. It requests the user's credentials and if the user is a valid user, the App Share Server will attempt to authorize and validate the user.
7. After establishing the connection, an App Share user session is established for the App Share Client and allows App Share Client to display the remote application.
8. A stream controller multiplexes the display data from server to one selected client and forwards the input events from the selected client to server. The control signal on choosing the client is done by a scheduler.

#### 3.3.1.3 *Remote User Session Initialization and Management*

Before an App Share Server starts listening for an incoming connection request, it initiates a RDP connection to TS Server on Windows OS. When an App Share Server attempts to establish an RDP connection with TS Server, the App Share Server behaves like the RDP Client.

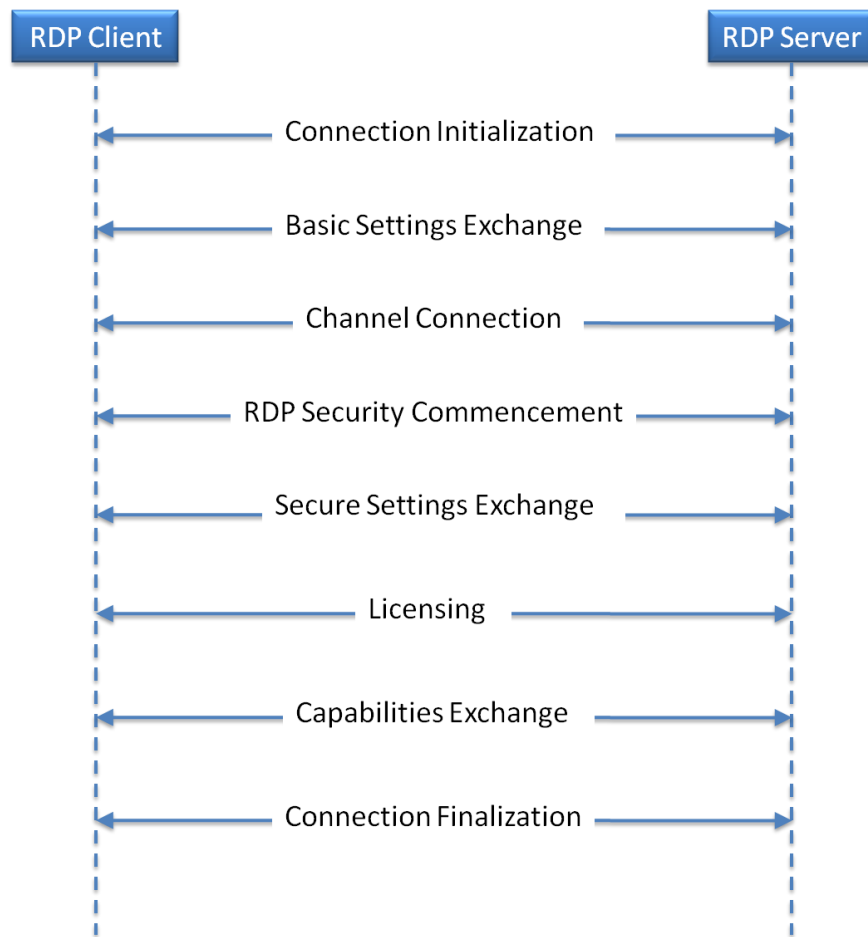
After initialization of the RDP connection to TS Server on Windows OS, App Share Server starts listening for an incoming connection request. When an App

Share Client attempts to establish a connection to App Share Server, the App Share Server behaves like a TS Server and the App Share Client behaves as an RDP Client.

The sequence of steps of processing the connection request in both cases above are the same as when an RDP Client attempts to establish a connection with a TS Server [55]:

1. The TS Server passes configuration and policy data to the RDP Client.
2. The TS Server requests information about the capability of the RDP Client.
3. The TS Server queries for data from the RDP Client that will be overridden by the configuration and policy data of the TS Server.
4. The TS Server will then start a licensing sequence, requesting a license from the RDP Client and attempting to validate the license. If a new or updated license is required, the TS Server will use licensing services to obtain a new or updated license and then will send the license back to the RDP Client. If the TS Server is configured in a per-user licensing mode, the TS Server will establish a connection without validating the license provided by the RDP Client.
5. The TS Server requests the user's credentials and if the user is a domain user, the TS Server will attempt to authorize and validate the user using directory services. If the user is not allowed to log on to the TS Server, the connection request will be terminated with an appropriate error message.
6. If the user is allowed to log on, the TS Server will query for the handles to the I/O objects and will construct a terminal object. The TS Server binds the terminal object to the session object, fully establishing the connection and allowing the RDP Client to display the remote desktop or remote application.

Figure 20 RDP connection sequence diagram illustrates one example of the messages that are exchanged between an RDP Client and a RDP Server.



*Figure 20 RDP connection sequence diagram [55]*

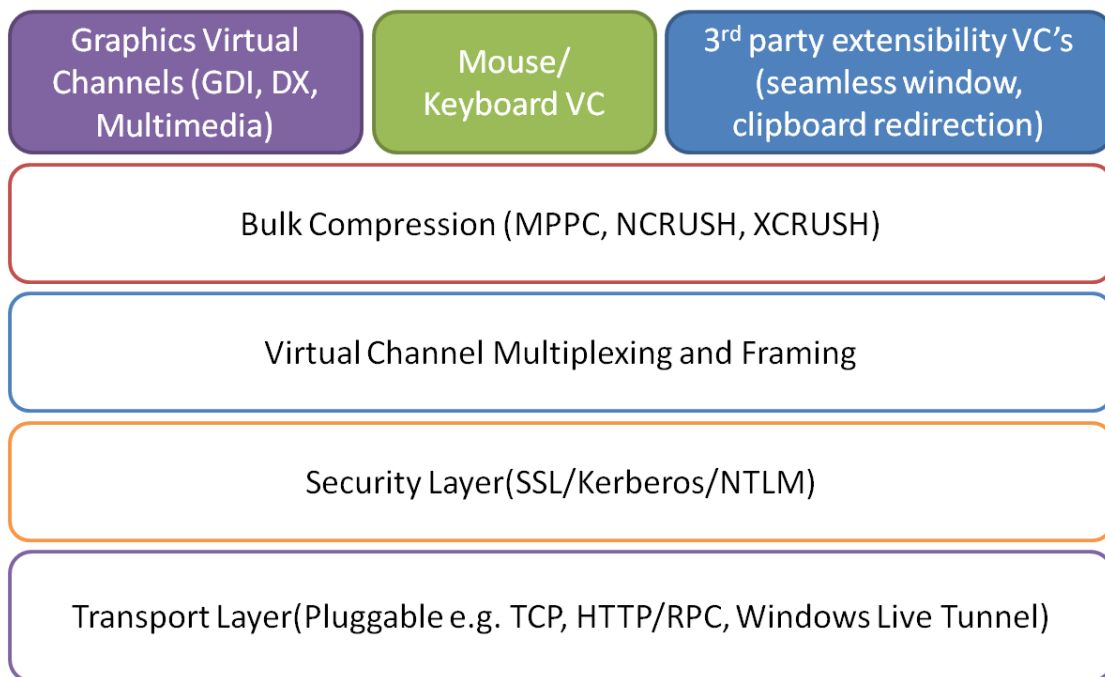
#### 3.3.1.4 Virtual Channel in Remote Desktop Protocol and SeamlessApp

The RDP protocol allows communication via up to 64,000 channels. The screen is transmitted as bitmap graphics from the server to the client or terminal. The client transmits the keyboard and mouse inputs and interactions to the server. Therefore, the communication is extremely asymmetric as most of the data are transmitted from the server to the client.

RDP was originally designed to support different network topologies. In its current state, it can be executed only via TCP/IP networks and is internally divided into several layers. The reason for this, at the lowest level, is that the T.120 protocol family, on which RDP is based, was optimized in accordance with



some rather complex specifications of the ISO model. These were mostly grade-of-service mechanisms. Because these cannot be mapped to the TCP/IP protocol, an X.224-compatible adaptation layer handled mapping the specified service primitive of the ISO layer to the service primitive of the TCP/IP protocol. RDP is used to tunnel graphical data, input data, and device data (and other communication) between an RDP Client and a TS Server. RDP also defines an extensible virtual channel mechanism. Each virtual channel acts as an independent data stream. The RDP Client and TS Server examine the data received on each virtual channel and route the data stream to the appropriate endpoint for further processing. The necessary static virtual channels are opened at the start of the session during handshaking, and remain open until the session is closed. Figure 21 is the legacy RDP Architecture [58]:



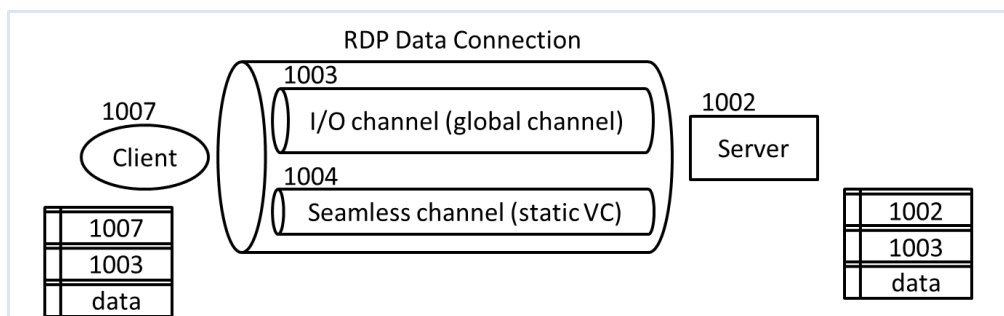
*Figure 21 RDP architecture*

The activity involved in sending and receiving data through the RDP stack is essentially the same as the seven-layer OSI model standards for common LAN networking today. Data from an application or service to be transmitted is passed

down through the protocol stacks, sectioned, directed to a channel (through MCS), encrypted, wrapped, framed, packaged onto the network protocol, and finally addressed and sent over the wire to the client. The returned data works the same way only in reverse, with the packet being stripped of its address, then unwrapped, decrypted, and so on until the data is presented to the application for use.

During RDP connection sequence, the RDP Client proceeds to join the user channel, I/O channel, and all virtual channels by using multiple MCS Channel Join Request PDUs and the TS Server confirms each channel with an MCS Channel Join Confirm PDU. All subsequent data sent from the RDP Client to the TS Server is wrapped in an MCS Send Data Request PDU, while data sent from the TS Server to the RDP Client is wrapped in an MCS Send Data Indication PDU. This is in addition to the data being wrapped by an X.224 Data PDU. [58]

The MCS PDU field encapsulates either an MCS Send Data Request PDU (if the PDU is being sent from client to server) or an MCS Send Data Indication PDU (if the PDU is being sent from server to client). In both of these cases, the embedded channel Id field must contain the server-assigned virtual channel ID. This ID must be used to route the data in the virtualChannelData field to the appropriate virtual channel endpoint after decryption of the PDU and any necessary decompression of the payload has been conducted. An illustration of virtual channel in RDP is shown in Figure 22 below:



*Figure 22 Virtual channel in RDP*

MCS I/O channel is to send and receive display update data and client's input events. A simplified version of PDU format is shown in the above figure. Each PDU has a channel initiator, channel ID and channel data. For example, if a PDU is sent from Client to Server via global channel it must consists of initiator = 1007 (0x03ef) and channel Id = 1003 (0x03eb); if a PDU is sent from Server to Client via global channel it must consists of initiator = 1002 (0x03ea) and channel Id = 1003 (0x03eb).

Static virtual channel provides application specific functions and features. It allows lossless communication between client and server components over the main RDP data connection and it is opaque to RDP [58]. Seamless window channel is a static virtual channel [59].

Virtual channels thus help add functions that are not yet specified in the RDP protocol. They represent a platform that future developments can be based on without having to modify the communication methods between a terminal server and its clients.

#### 3.3.1.5 *Data Stream Control for Multiple User Sessions*

After a new user session is created, a new client is added to data stream controller for starting additional application. There are three major programming modules inside data stream controller, namely the scheduler, MUX and DEMUX. And there are two important control signals: Allocated Client ID and Focused Win ID. Each client has at most one focused window. Data stream controller keeps track of the focused window ID for each client. According to the allocated client information determined by the scheduler, data stream controller sends over the focus window information to Server then followed by the client's input events.

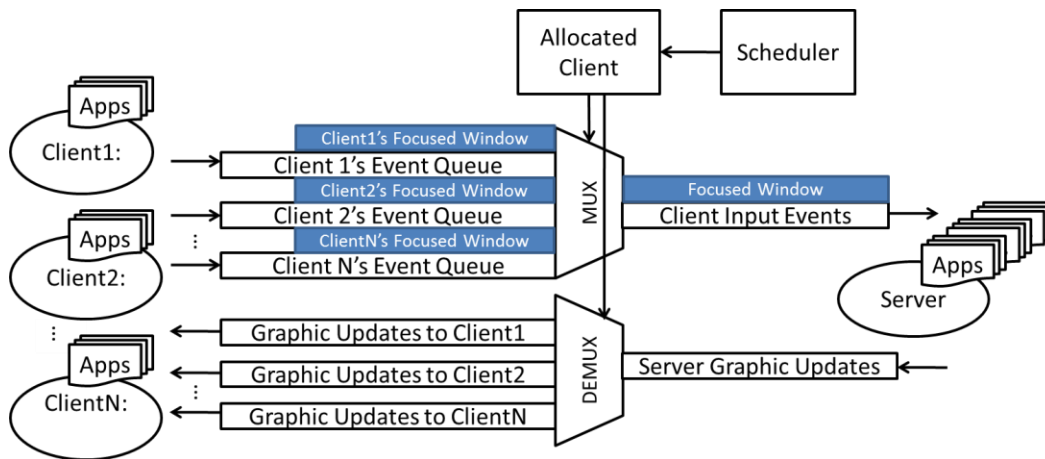


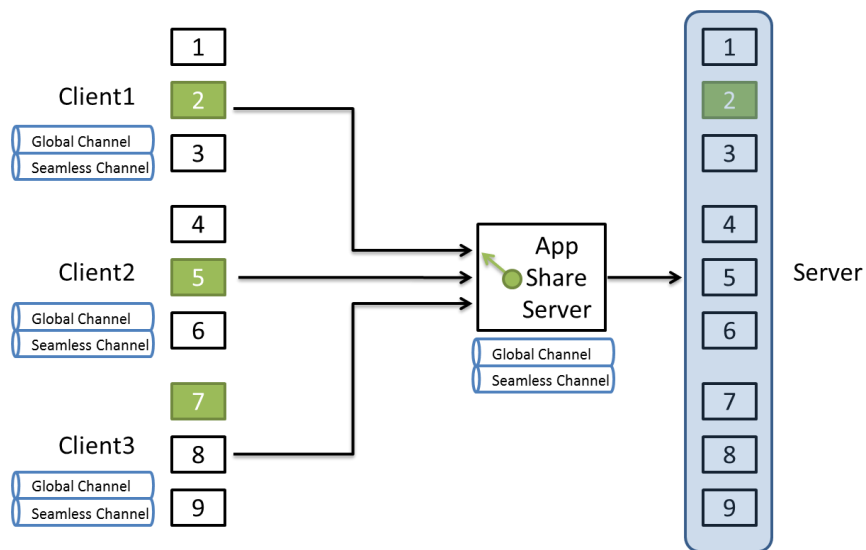
Figure 23 Data stream controller

Figure 23 shows the architecture of the data stream controller in App Share Server. Data stream controller is in charge of multiplexing and de-multiplexing the clients' and server's traffic. It maintains the smooth execution of multiple remote user sessions of ShAppliT system.

Allocated client is the control signal for the steam multiplexer and de-multiplexer. At a time only one client is enabled to transmit it input events and to receive the graphic updates from server. The allocated client is determined by a scheduler. Clients' input events in the global channel including mainly the keyboard and mouse inputs are buffered in an event queue of each client respectively. Currently our implementation of the scheduler uses a round-robin scheduling algorithm which assigns time slices to each client in equal portions and in circular order and handles all clients' events without priority.

Figure 24 illustrates the relationship of focused window and allocated client at both client and server sides. Each client may have one or more applications running from the same sever, but at a time there is only one window focused by the client. A focused window is the window the client is operating on currently. So, at the client side each client will have at most one window focused shown with filled colour box. At the sever side, only one window is focused each time shown with filled colour box. Therefore, it is important to keep track of the

focused window ID for each client. Allocated client is decided by the scheduler according to the scheduling algorithm as mentioned earlier. When it comes to a client's turn to send over its events the server will be notified about the current focused window by our ShAppliT Server. Then the TS server will perform operations on the focused window of the allocated client according to the input events received and send the server output graphic update data over to the allocated client.



*Figure 24 Illustration of focused window and allocated client*

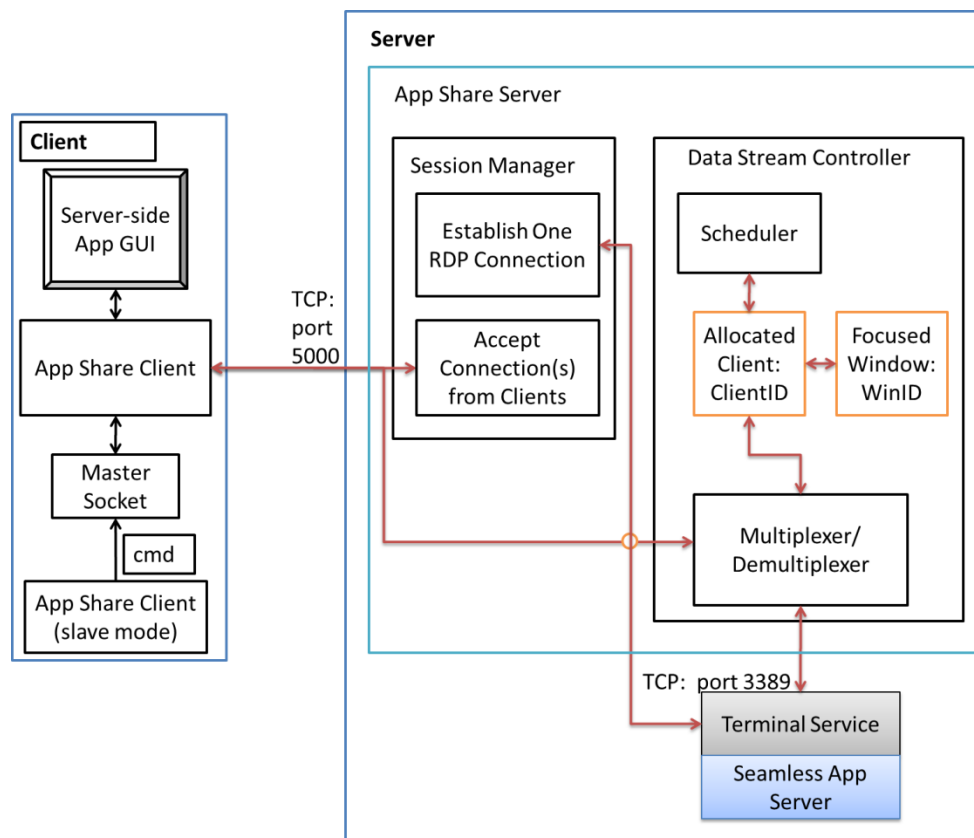
Focused window information is extracted from the network packet flow of the seamless virtual channel in RDP as mentioned in the previous section. The seamless channel ID is determined by the negotiation between client and server during the RDP connection sequence. Focused window information is carried in the seamless virtualChannelData field with the format "focus, win ID, flags". The seamless channel data is directed by the TS server to the SeamlessApp Server endpoint for further process [59].

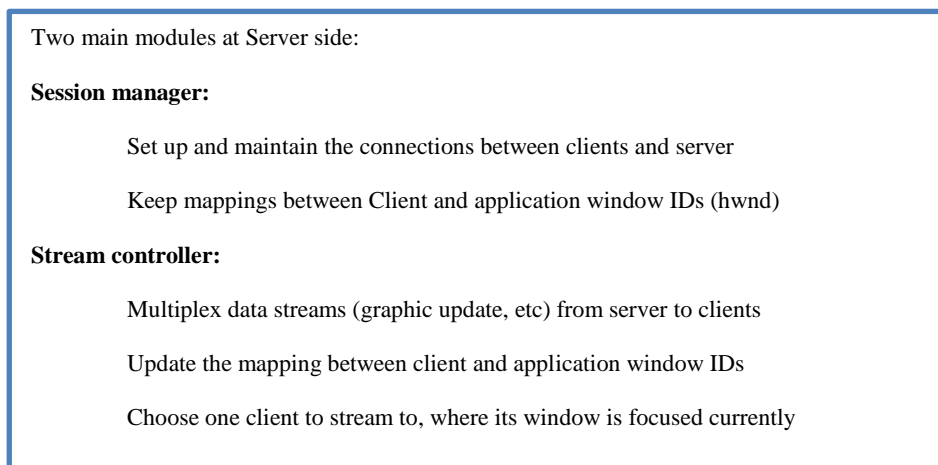
### 3.4 Implementation of a Demonstrating System

The application, ShAppliT realizes the proposed peer-to-peer application sharing on closed systems in a cluster. It is implemented on Windows XP X86 32-bit operating system in a local area network (LAN) environment. A clustering system using multicast and multiplexing approach have been implemented.

#### 3.4.1 Detailed Programming Model

Figure 25 shows the detailed programming model of ShAppliT.





*Figure 25 Programming model of ShAppliT system*

Master mode is the default mode of ShAppliT Client. When run in master mode, ShAppliT Client creates and listens on a master socket. After creation of a remote user session with a ShAppliT Server and maintenance of that connection, ShAppliT Client listens on the master socket and checks master socket each time when TCP layer receives packets.

When run in slave mode, ShAppliT Client notifies the master Client instance of a new command to be run by sending command (e.g. "mspaint") to the master socket and then exits. The master instance detects a command from a client and sends a client-to-server message (e.g. "spawn, mspaint") to the ShAppliT Server. The message will be directed to SeamlessApp server component at the Windows server, which runs the new command on the server machine. Finally, a remote application is launched at the Windows server and the application Graphic User Interface (GUI) will be received by ShAppliT Client. Moreover, the slave mode can be used multiple times to send more application commands. So, it provides connection sharing by allowing a single ShAppliT connection to launch multiple applications.

There are two components in the ShAppliT Server, namely the Session Manager and Data Stream Controller. The session manager component first establishes an

RDP connection with Microsoft Terminal Service Server. Then, it listens on TCP port 5000 and accepts connections from remote clients. It creates new user sessions for remote clients after successful connection negotiation. The connection sequence follows the RDP connection sequence mentioned in MS-RDPBCGR [55]. After a new user session is created, a new client is added to the data stream controller for starting additional application. Data stream controller is in charge of multiplexing and de-multiplexing the clients' and server's traffic. It maintains the smooth execution of multiple remote user sessions of ShAppliT system.

There are three major programming modules inside data stream controller, namely the scheduler, MUX and DEMUX. There are two important control signals: Allocated Client: Client ID and Focused Window: Win ID. Each client has at most one focused window. Data stream controller keeps track of the focused window for each client. According to the allocated client information determined by the scheduler, data stream controller sends over the focus window information to TS Server then followed by the client's input events. Our current scheduler uses a round-robin scheduling algorithm. Clients' input events are queued in a buffer of each client respectively. The scheduler assigns time slices to each client in equal portions and in circular order. The next client will be allocated after the timer expired. Allocated client is the control signal for the stream multiplexer and de-multiplexer. At a time only one client is enabled to transmit its input events and to receive the graphic updates from server. The allocated client is determined by a scheduler. Clients' input events in the global channel including the keyboard and mouse inputs are buffered in an event queue of each client respectively.

### 3.4.2 *App Share Client*

Figure 25 shows the programming Model of App Share System. App Share Client software is an application that establishes and maintains the connection between a



client and a server computer running App Share. Our App Share Client is implemented on top of rdesktop [60] which is a free, open source client for Microsoft's proprietary RDP protocol. Rdesktop is able to work with a number of Microsoft Windows versions such as NT 4 Terminal Server, 2000, XP, 2003, 2003 R2, Vista, 2008, 7, and 2008 R2. Rdesktop was initially written by Matthew Chapman. It is released under the GNU General Public License and is available on Unix-like systems such as BSD and Linux [61].

#### 3.4.2.1 *Master mode and slave mode of App Share Client*

- Master mode: Specify the path for the control socket that the rdesktop process listens on. By default, this is \$HOME/.rdesktop/seamless.socket
- Slave mode: Instead of starting a new rdesktop process, connect to an existing process' control socket and tell it to run a command on the server.

As shown on the left hand side of Figure 25 Programming model of ShAppliT system, master mode is the default mode of App Share Client; when run in master mode, App Share Client creates and listens on a master socket. After creation of a remote user session with App Share Server and maintenance of that connection, App Share Client keeps listening on the master socket and checks master socket each time when TCP layer receives packets.

When run in slave mode, App Share Client notifies the master App Share Client instance of a new command to be run by sending command (e.g. "mspaint") to the master socket and then exits. The master instance detects there is a command from client and sends a client-to-server message (e.g. "spawn, mspaint") to the App Share Server. Then the message will be directed to SeamlessApp server component at Windows server, which runs the new command on server machine. Finally, a remote application is launched at Windows server and the application GUI will be received by App Share Client. Moreover, the slave mode can be used multiple times to send more application commands. So, it provides connection

sharing by allowing a single App Share connection to launch multiple applications.

### 3.4.3 *App Share Server*

There are two components in the App Share Server, including Session Manager and Data Stream Controller.

The session manager component firstly establishes an RDP connection with Microsoft Terminal Service Server. Then, it listens on TCP port 5000 and accepts connections from remote clients. It creates new user sessions for remote clients after successful connection negotiation. The connection sequence follows the RDP connection sequence mentioned previously.

After a new user session is created, a new client is added to data stream controller for starting additional application. There are three major programming modules inside data stream controller, namely the scheduler, MUX and DEMUX. And there are two important control signals: Allocated Client: Client ID and Focused Window: Win ID. The relationship between allocated client and focused window at both the client and server sides is introduced previously. Each client has at most one focused window. Data stream controller keeps track of the focused window for each client. According to the allocated client information determined by the scheduler, data stream controller sends over the focus window information to TS Server then followed by the client's input events. Our current scheduler uses a round-robin scheduling algorithm. Clients' input events are queued in a buffer of each client respectively. The scheduler assigns time slices to each client in equal portions and in circular order. The next client will be allocated after the timer expired. So, it handles all clients' events without priority. Figure 26 Programming flow chart of App Share Server illustrates the programming flow chart of App Share Server.

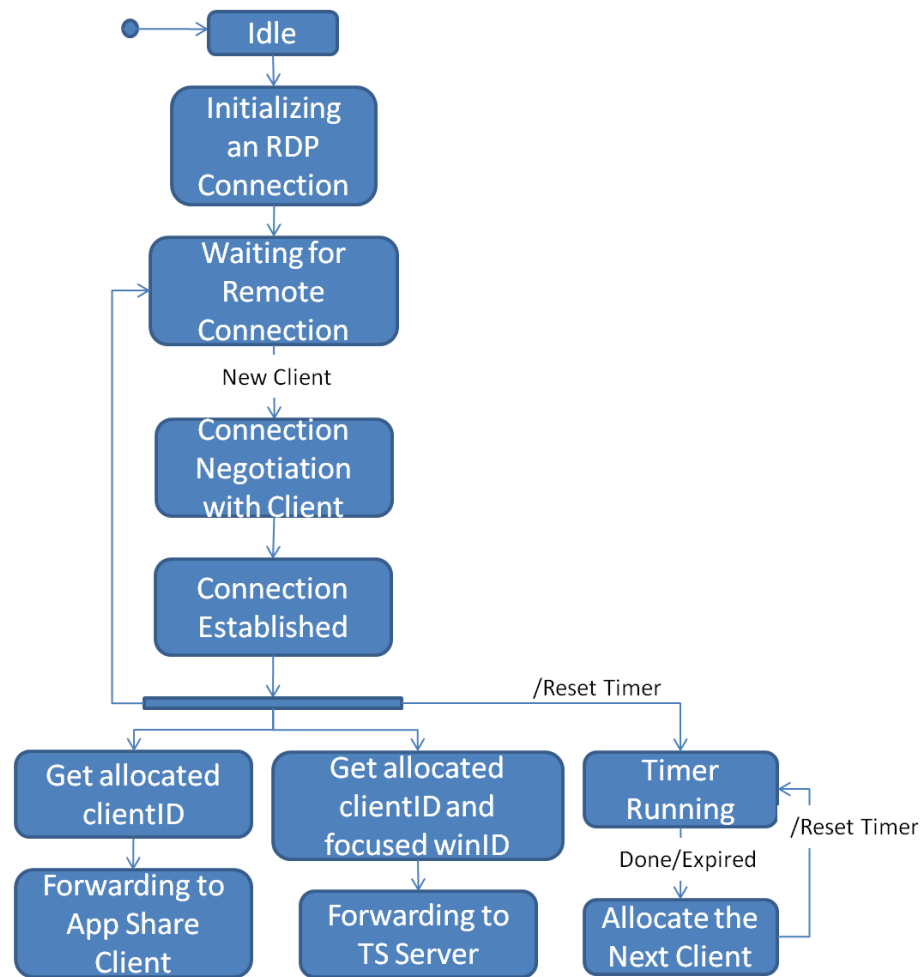


Figure 26 Programming flow chart of App Share Server

Sending focused window information is supported from client to server by seamless RDP feature of rdesktop at the client side and a seamless RDP server at the server side. They communicate end to end via the lossless seamless virtual channel. Details will be presented in the next section.

#### 3.4.3.1 *SeamlessApp*

The default way of deploying ShAppliT application has been set to seamless mode. It enables App Share Client to run individual applications rather than a full desktop. Also, the application itself looks as if it's been started from the local machine when it comes to the look and feel. In seamless mode, an end sees no difference between the remote application in App Share session and his/her local

application. The technology behind the seamless application basically cloaks or clips out the part of the window that shows the application in a normal Windows shell. There are three key features of SeamlessApp which facilitates the implementation:

1. Add a client-to-server message for starting an application: When run in slave mode, App Share Client notifies the master App Share Client instance of a new command to be run by sending command (e.g. "Ms Paint") to the master socket and then exits. The master instance detects there is a command from client and sends a client-to-server message (e.g. "spawn, Ms Paint") to launch a new application at the server.
2. Enhanced support for WM\_DELETE\_WINDOW: Instead of terminating the whole App Share connection when one client side window is closed, a client-to-server message is send to close the corresponding window on the server side.
3. Support for sending focus information from client to server: Focused window information is carried in the seamless virtualChannelData field with the format "focus, win ID, flags".

Figure 27 Control messages in seamless virtual channel shows the interaction between client and server about the above features in the SeamlessApp mode.

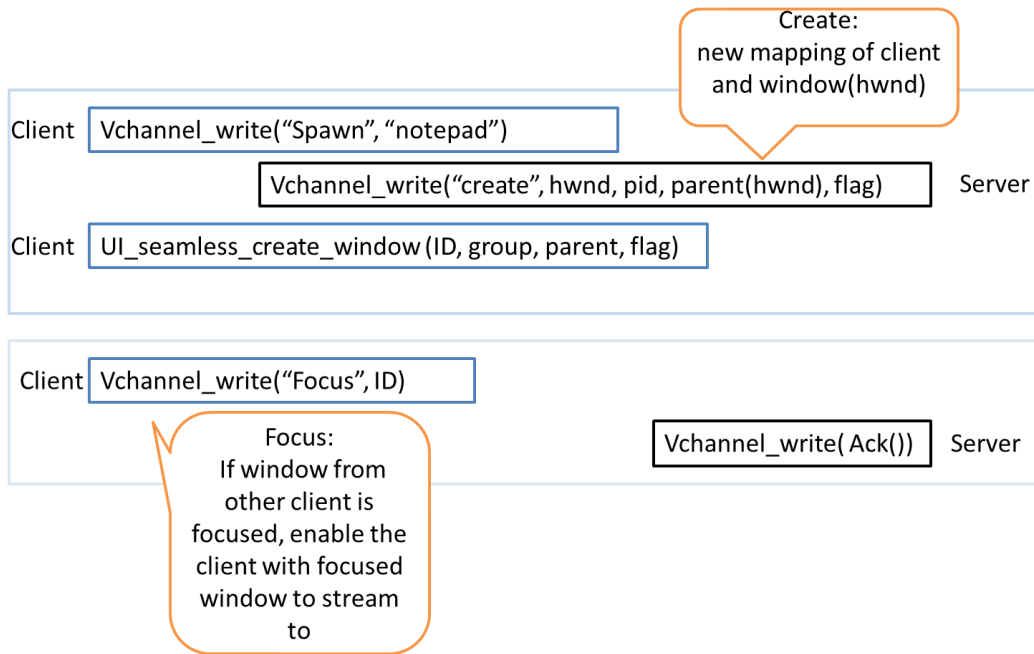


Figure 27 Control messages in seamless virtual channel

Table 3 Client ID and Window ID

Client ID	Application	Hwnd (WinID)	PID	Parent
8	explorer	0x00030022	0x00000fd8	0
	notepad	0x00030030	0x00000b20	0
		0x00030078		0x00030030
		0x00020078		0x00030030
9	mspaint	0x0001005e	0x00000f94	0

Table 3 shows the relationship among Client ID, application name, Process ID, Win ID, Parent Win ID. The Client ID in our implementation is assigned by the client socket number. Each client may have multiple applications running at the server. Each application is associated with a process ID and a main window ID. The main window without parent will have parent win ID 0. When a user operates

on certain applications, there are child windows created under the parent win ID. Therefore, each client has multiple windows to manage and each time at most one window is focused by the client. SeamlessApp supports for sending focus information from client to server. App Share Server will decode the focused window information carried in the seamless virtual channel "focus, win ID" and monitors each client. If it comes to a client's turn by the scheduler, its "focus, Win ID" will be send to SeamlessApp Server via seamless virtual channel by App Share Server. And subsequently, the input events of the client will be forwarded to TS Server by App Share Server.

### 3.5 Results and Discussion

Our test bed is set up in a LAN consisting of ten computers with ShAppliT application and identical system environment configurations. All PCs are Pentium4 3.0GHz machines with 512MB physical memory running Windows XP professional SP3. The performance evaluation is mainly focused on describing the impact of increasing the number of remote application sessions on the memory consumption of a host computer running ShAppliT. The load analysis shows that additional remote connection results in a linear increase of the commit charges on host computer.

#### 3.5.1 *User Interface*

The user interface of App Share and the screen shot of configuration of share/un-share an application are shown in Figure 28 Screen shot of the demonstrated App Share and Figure 29 Screen shot of the demonstrated App Share: setting share/un-share applications Figure 29 Screen shot of the demonstrated App Share: setting share/un-share applicationsbelow.

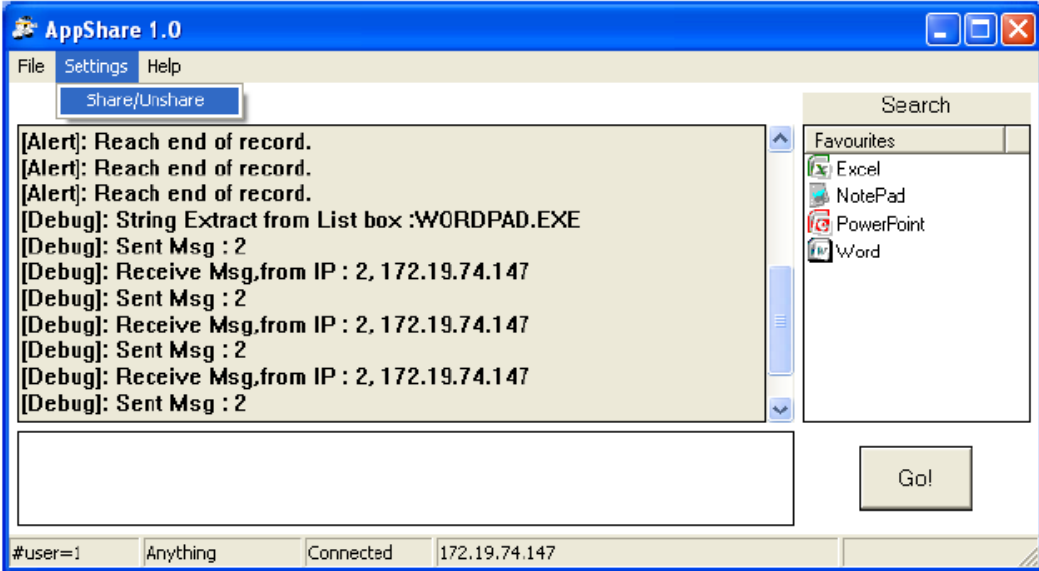


Figure 28 Screen shot of the demonstrated App Share



Figure 29 Screen shot of the demonstrated App Share: setting share/un-share applications

### 3.5.2 Multi-session Load Analysis

In the experiment, tests are carried out on a single host machine running multiple remote application sessions of WordPad.exe. This analysis helps us evaluate the memory performance of the computer and determine the maximum concurrent session to be accepted on the machine. This testing is conducted using a host computer with ShAppliT installed and deployed. The performance data is obtained using the performance analysis tool implemented in Windows Task Manager. Table 4 below gives an overview on detailed performance analysis of Windows Task Manager.

Table 4 Details on Windows Task Manager Performance analysis [62]

Parameter	Details
Commit Charge	Amount of virtual memory reserved by the operating system for the process. Memory allocated to programs and the operating system. Because of memory copied to the paging file, called virtual memory, the value listed under Peak may exceed the maximum physical memory. The value for Total is the same as that depicted in the Page File Usage History graph.
Physical Memory	The total physical memory, also called RAM, installed on your computer. Available represents the amount of free memory that is available for use. The System Cache shows the current physical memory used to map pages of open files.
Kernel Memory	Memory used by the operating system kernel and device drivers. The paged is memory that can be copied to the paging file, thereby freeing the physical memory. The physical memory can then be used by the operating system. Non-paged is memory that remains resident in physical memory and will not be copied out to the paging file.



In Windows Server 2008 R2, user can configure the number of simultaneous remote connections that are allowed for a connection. A client windows machine is converted to a windows server by implementation of ShAppliT V1.0. Our main goal is to compare out broker mediated solution (ShAppliT V2.0) with emulated Windows server (ShAppliT V1.0). ShAppliT V1.0 makes modifications on terminal service (TS) DLL file and the registry of Windows XP as described in references [63] and [64]. In this case, Windows terminal service server manages the connections sessions directly such that no broker is needed for exchange of information between server and client. A control session is used as a reference whereby the data is captured when no remote session is taking place. The data is being recorded every time an additional remote session is launched from a client computer and the result is shown in Table 5 Multi-session load analysis on host computer with ShAppliT V1.0 and Table 6 Multi-session load analysis on host computer with ShAppliT V2.0.

Table 5 Multi-session load analysis on host computer with ShAppliT V1.0

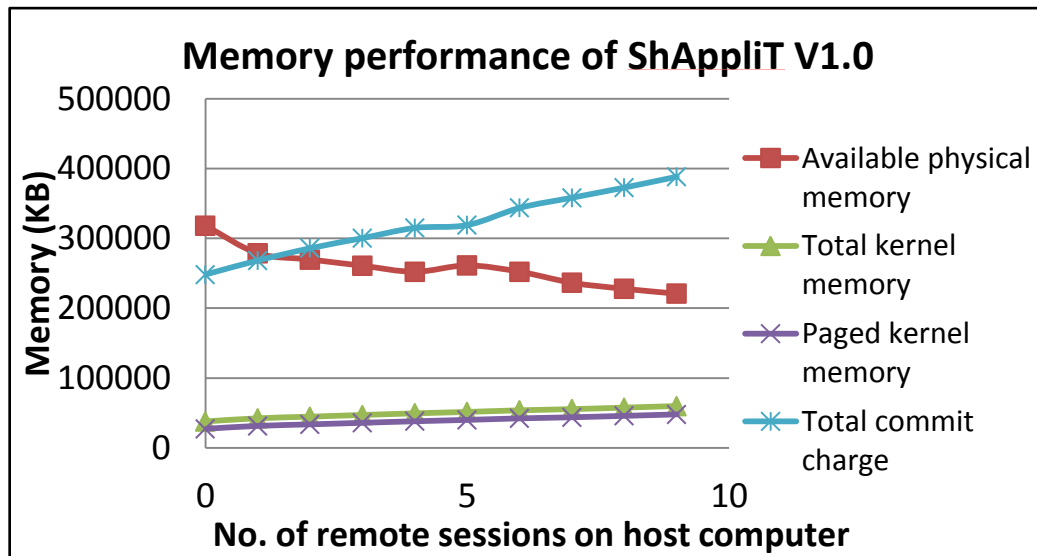
No. of remote sessions	Total physical memory(K B)	Available physical memory(K B)	Total kernel memory(K B)	Paged kernel memory(K B)	Total commit charge(KB)
0	514116	318056	37864	27276	248176
1	514116	278588	42312	31420	268272
2	514116	269660	44660	33600	285812
3	514116	260704	47076	35876	300284
4	514116	252144	49412	38056	314888
5	514116	261168	51440	40036	318972

6	514116	252172	53752	42188	343584
7	514116	236348	55448	43768	358156
8	514116	227700	57600	45776	372684
9	514116	220752	59864	47896	388188

Table 6 Multi-session load analysis on host computer with ShAppliT V2.0

No. of remote sessions	Total physical memory(K B)	Available physical memory(K B)	Total kernel memory(K B)	Paged kernel memory(K B)	Total commit charge(K B)
0	514116	318056	37864	27276	248176
1	514116	308092	38112	27524	249012
2	514116	306060	38244	27656	251132
3	514116	305416	38388	27800	251932
4	514116	303880	38532	27944	254080
5	514116	302248	38712	28124	255744
6	514116	300976	38868	28280	257356
7	514116	308312	39020	28432	258932
8	514116	308716	39172	28584	260468
9	514116	307864	39324	28736	261856

The load analysis of ShAppliT V1.0 (Figure 30 Memory performance of ShAppliT V1.0 when hosting multiple remote sessions) shows that additional remote connection results is a linear increase of the commit charge on the host computer. And the physical memory at the host computer decreases with increasing number of multiple remote sessions. As such, it is necessary to set a limit on the maximum number of concurrent sessions so that the host computer would not be burdened by excessive remote connections and experience laggings in the local session. This result also highlights that although this system provides certain degree of scalability but further performance optimization on memory consumption still need to be done.



*Figure 30 Memory performance of ShAppliT V1.0 when hosting multiple remote sessions*

The load analysis of ShAppliT V2.0 (Figure 31 Memory performance of ShAppliT V2.0 when hosting multiple remote sessions) shows that additional remote connection results in a linear increase of the commit charge on host computer. The increment of commit charge is very small with increasing number of concurrent remote sessions. In addition, the available physical memory of the host computer is affected very little by the multiple remote sessions.

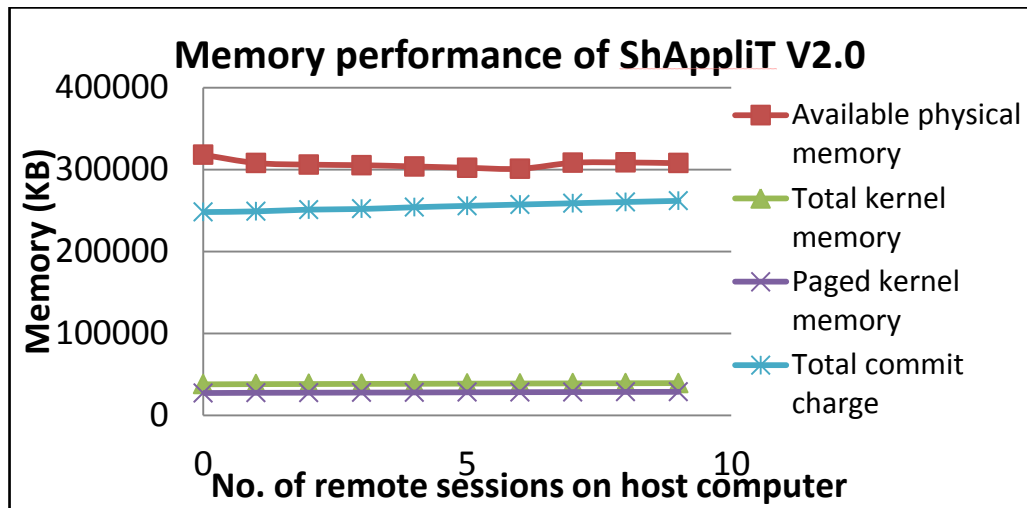


Figure 31 Memory performance of ShAppliT V2.0 when hosting multiple remote sessions

Memory management is more effective in ShAppliT V2.0 compared to ShAppliT V1.0 observed from Figure 32 Comparison between ShAppliT V1.0 and ShAppliT V2.0 on commit charge when hosting multiple remote sessions below. This is because in ShAppliT V2.0 there is only one RDP connection established and maintained by ShAppliT Server. Each additional application launched at host computer is invoked by SeamlessApp server in the same way as using cmd.exe at host computer. As a result, starting a new application session, the operating system only allocates the necessary memory resource to the application process running within the same user. While, in ShAppliT V1.0 multiple remote connections are made to Windows TS server directly and multiple RDP sessions are established. Each time any new request of application from client, the host computer launches an additional RDP session for the application. Therefore, the operating system reserves the memory for multiple RDP sessions in ShAppliT V1.0, which consumes much more memory than within one RDP connection session.

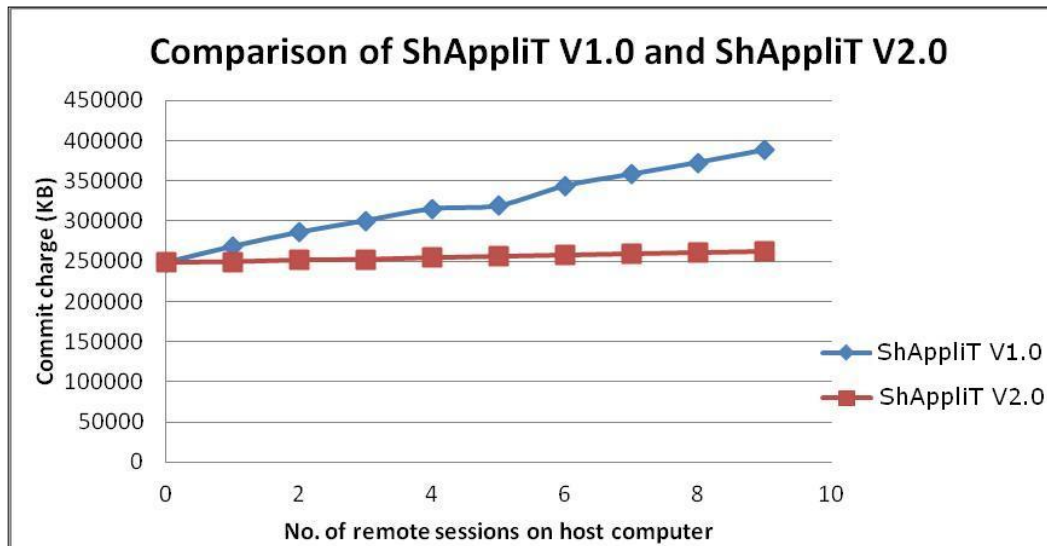


Figure 32 Comparison between ShAppliT V1.0 and ShAppliT V2.0 on commit charge when hosting multiple remote sessions

### 3.5.3 License Issue on Application Sharing

Most of the software installed in personal computers have a single user software license and cannot be transferred from one user to another. For example Microsoft office edition 2007 says the single primary user of a licensed device may access and use the software installed on the licensed device. Single user may use remote access technologies, such as the Remote Desktop features in Microsoft Windows or NetMeeting, to access and use the licensed copy of the Software, provided that only the primary user of the device hosting the remote desktop session accesses and uses the Software with a remote access device [65].

The single user licensing problem of application sharing is solved in our current approach ShAppliT V2.0 by establishing one RDP connection for multiple clients. The ShAppliT Server sits in between the ShAppliT client and TS server as a broker. It logs in to the host operating system via RDP, handles tasks from multiple clients, including the local user sitting in front of the computer and passes them to the TS server. Therefore, TS server sees only one RDP session and it feels that it works for the broker only. And the broker is in charge of connecting

to TS Server, creating remote user sessions and multiplexing/de-multiplexing the data streams. So, when a client want to launch a remote application from the server, the application will be open at the server under the same user account which is the one established by the broker. Therefore, with our application sharing tool ShAppliT, as long as there is one user license for the software, it can be shared among multiple clients without violating any licensing terms.

Furthermore, the client version of Microsoft Windows operating system (e.g. Windows XP Professional, Windows 7) terminal service limits the number of users logged in to one at a time. Two people cannot be logged on at the same time even if it includes just a physical, local-console login and a remote login. It has to be one or the other and only one user at a time. In ShAppliT V2.0 system, there is only one master user login by ShAppliT Server (broker). So, the problem of the closed system limitation on single user logged-in session is not an issue for broker mediated application sharing system.

#### 3.5.4 *Some Limitations of Our Implementations*

In our current implementation of application sharing cluster, Windows OS is chosen as the implementation platform. Currently, the architecture has been evaluated on Windows OS. However, in a cluster environment the operating systems are heterogeneous in general. More implementation and performance testing should be done on other OS as well for example Mac, Linux OS, to demonstrate the framework and the methodology are widely applicable.

More experiments can be done on other applications to collect more data and evaluate the performance. Currently only the basic applications are tested, e.g. MS Paint, MS word, notepad, calculator, etc.

### 3.6 Summary

A novel P2P application sharing system ShAppliT has been developed in a cluster which supports generic application sharing and concurrent multiple sharing sessions. The proposed architecture is a clever blend of cluster computing and peer-to-peer concepts. ShAppliT enables client remote access application resources that are not installed on the local computer. Also, a peer can host multiple remotely access sessions without any interference for his own experience. A broker-mediated solution has been provided to extend a single user licensed software resource for multiple user usage without modifying the operating system. Experiments also show that our application sharing system has good usability, scalability and a friendly user interface. Our broker-mediated system architecture has wide applicability on other closed systems.

# CHAPTER 4 BUILDING A RELIABLE FILE SYSTEM FOR FAULT-TOLERANT SERVICES

---

---

## 4.1 Introduction

As a cluster is scaled up to large number of nodes and disks, it becomes more risky that some components are working incorrectly from time to time. There is a need to handle component failures gracefully and keep operating in the presence of failures. [66] In computing, a file system can be regarded as a method to store and organize files and data so that there will be ease in finding and accessing these files. In other words, it can be viewed as a collection of files with directory structures, and a file system will provide an abstraction of accessing the files or directories. Due to the high possibilities of system and media failures, as well as the presence of user and application faults, hence this calls for a need to protect important file system data so that data loss can be minimized. A successful application sharing system should provide reliable services. In the current cluster file system literature, there are two main streams of research on addressing different applications or workflows, one is directed acyclic graph (DAG) structured workflow and the other one is Map-reduced workflow. For DAG structured workflows, they are mainly for scientific workflows and often re-use large datasets in multiple workflows. The scientific tasks consume whole files and replicate the whole files rather than striping files as in Hadoop. There are some examples of cluster file systems for DAG-structured workflows in the literature, namely Makeflow, Chirp and Confuga.

As compared with the DAG workflow, the Map-Reduce application or workflow has the following features, they are mainly leveraging with Hadoop distributed file system for Map Reduce workflows. In terms of accessing the files, they are



block oriented and no whole-file access. The Map-Reduce workflow is inefficient for single task whole-file access.

Therefore, based on the literature research, one chief technology to accomplish fault-tolerant application services in a cluster is file/data replication at client or server or third peer. Through literature review, several previous efforts have been done to protect file system data. A key idea would be to retain important older versions of the file systems followed by storage reclamation. Another concept that was implemented was to allow users to make and maintain multiple copies of data and avoid deletes whenever possible. In view of the rising efforts in this key area pertaining to operating systems, this led to the strong motivation behind this work, whereby the main aim is to create a reliable and secure file system. A failure-save solution has been designed and implemented which enables user to login to the file server from anywhere, synchronizes document to last saved state on server and provides certain degree of portability. Through this implementation, it is hoped to establish appropriate techniques that can be used for the actual implementation of a reliable file system to accomplish fault-tolerant application services.

## **4.2 Portable File System (PFS) on Filesystem in User Space (FUSE)**

In this thesis, a file system called “PFS” was built on top of FUSE [67]. As such, this section serves to provide some background information and basic description of FUSE. FUSE or File system in User Space is a loadable kernel module for Unix-like operating systems, and it is a platform that allows users to create their own file systems without editing kernel codes. This is achievable by the running of the file system codes in user space (which also explains the name), while the FUSE module bridges to the actual kernel interfaces.

The overview of the reliable file system architecture is as shown in Figure 33:

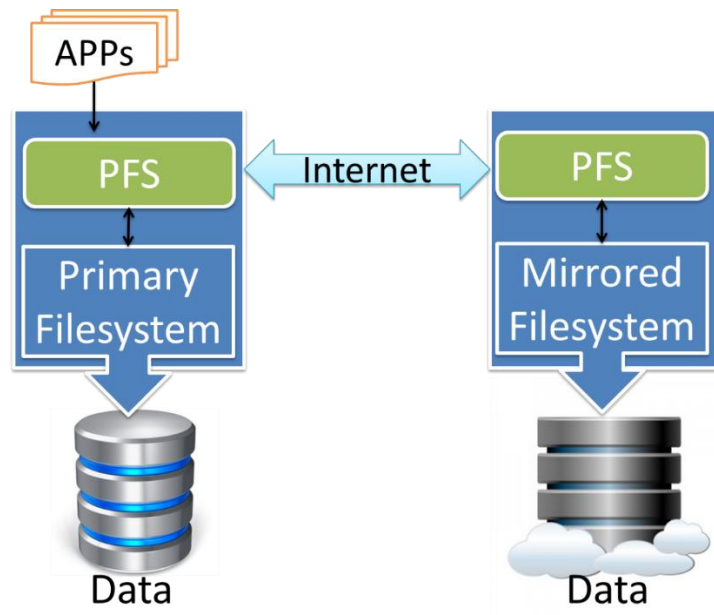


Figure 33 Overview of reliable file system architecture

FUSE was originally part of “A Virtual File System” (AVFS) [68], but it is now a separate project on SourceForge.net. It is free software as it is released under the terms of the GNU General Public License and the GNU Lesser General Public License. Also, FUSE is available for Linux [69] as well, and is officially merged into the mainstream Linux kernel tree in kernel version 2.6.14. Due to these mentioned characteristics, FUSE is decided for usage in this work. A FUSE file system is a program that listens on a socket for file operations to perform, and performs them. The FUSE library (*libfuse*) provides the communication with the socket, and passes the requests on to the user’s code. This is accomplished using a “call-back” mechanism. The call-backs are a set of functions that need to be written for the implementation of file operations.

### 4.3 Implementation of PFS

#### 4.3.1 Set-up of FUSE and Host Computers

First of all, FUSE was set up for the work by ensuring that most of the required file operations are functioning well before implementing additional source codes for the research purposes. As a reference, the "Big Brother File System (BBFS)" [70], was used as the skeleton that the work built upon. This file system is mainly a logging file system, and was utilized for debugging purposes in this work.

A basic file system called "PFS" was then built based on FUSE with reference to the BBFS; the reason for the name is that "Big Brother is watching." The file system simply passes every operation down to an underlying directory, but logs the operation. This file system will support the following minimum specification:

- 1) It can accommodate about 5000 documents, and each file has a maximum size of 50MB. A maximum of 50 characters will be supported for each file name.
- 2) The file system will support most of the major operations, including open, close, read, write, create, rename, delete, mkdir, rmdir on top of other basic calls like getattr and mknod.

The BBFS was being studied and but almost all the system call-backs were not suitable for our implementation, and many were not functioning in an appropriate manner as well. As such, new source codes have to be developed in order to derive a fully working PFS, which is the core of our implementation.

In this work, there will be two host computers, namely the server and the client. The client will be the site whereby the originating activities are done, which means that there will be user involvement at the client. Hence all the file operations will be originated at the client site. This means that the client is the primary file system.

As for the server, it will be the location whereby the file operations will be mirrored to. In this way, such a form of implementation will be able to represent a real-life application in which the client acts like a user workstation, and the server

is the host purely for mirroring purposes. Hence, the server will be the mirrored file system. Both the server and client will be set up and running and it is assumed that they are both in good working conditions. The assumption here is the original file operations is always performed at the client due to user involvement, and this will be followed by the same set of similar file operations being performed on the server. Hence, the state of the server will always be “behind” that of the client, and this would call for the need to perform the appropriate mirroring on the server as the implementation.

### 4.3.2 *Logging of File Operations*

In order to keep track of the file operations in the client, there will be a log file to record down the activities at the client. The log file has three fields and the description of the fields is as follows:

- ID – This field is in a number format, and is used to track the sequence of the file operations at the client site.
- CmdID – This field is also in a number format, and is used to represent the command types for the particular file operation that is being performed.
- Parameters – This field is related to the command type for the particular file operation, and it differs from command to command. However in general, this field will contain all the parameters/arguments involved in the command. For example, if a read operation is executed, then the parameters logged down will be offset, size and path name. As for a write operation, it will be the write buffer that is being logged.

### 4.3.3 *Client-Server Communication*

As seen from Figure 33 depicting the system architecture in the above subsection, network connection between client and server was established, and there is client-server interaction to allow network real-time mirroring to take place. The concept of the client-server communication protocol is quite similar to that of a

TCP/IP protocol. The client will send signals in the form of ID via the network whenever there are any file operations performed. As for the server, it will always be in the “listening” mode, so as to detect any signals sent by the client.

In detail, whenever a file operation is being performed on the client, the command type and the parameters involved in the file operation will be logged down in its log file. This means that there will be a new entry, and new entries are all added with increasing ID number. The client which is the primary file system will then tell the server (mirrored file system) of the ID number of the entry that it is going to send over. The server will then check whether that ID number sent by the client is equal to the very last ID + 1 in its own log file. If this is true, the client will proceed to send over the appropriate command type (in the form of CmdID) to the server. Otherwise, error recovery will be performed whereby the server will request the client to resend all the commands numbering from the previous ID at the server to the current ID at the client side. Under situations when the client or server is down, the error recovery mentioned above will take place. In this way, there will be assurance that there will be consistency in the data stored in both client and server. Hence the mirroring on the server will be up to date and similar to what was being performed on the client. These set of actions are clearly illustrated in Figure 34.

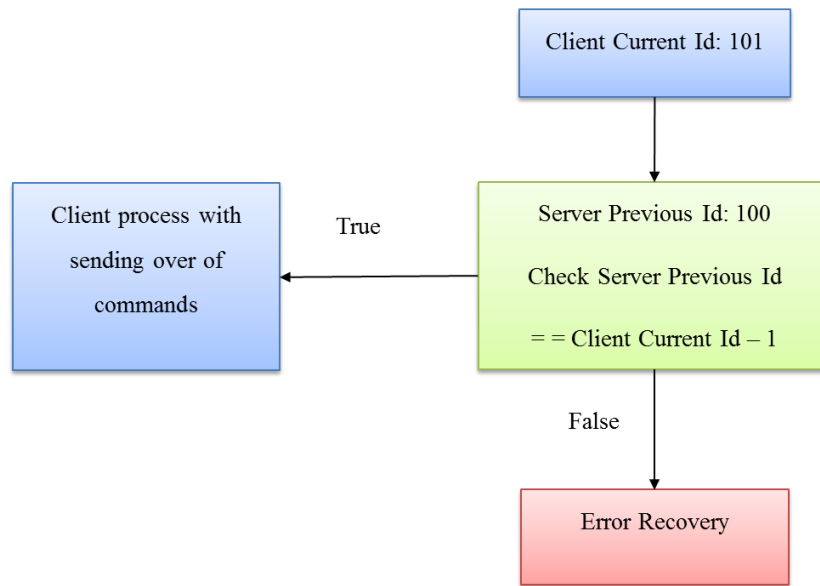


Figure 34 Flow for ID checking on server site

#### 4.3.4 Explanation of Callback Functions

Table 7 describes all the call-back functions that were implemented for this project.

Table 7 Call-back functions implemented in PSF

Function Name	Function
server_start client_start	To set up the basic connection on the server side and the client side respectively.
server_check client_check error_recovery	server_check and client_check determine the if the id numbers of the server and client match. error_recovery will synchronise the client and server if the id numbers did not match.
sendcmdid sendrmdir sendmkdir sendreaddir sendopendir sendopen sendread sendaccess	sendcmdid is called by the client to send the id number and command number to the server and perform appropriate checks.  The other functions perform the necessary communications with the server for that individual command.

sendgetattr sendrename sendunlink sendmknod sendcreate sendwrite sendlink	
pfs_getattr pfs_readlink pfs_mknod pfs_mkdir pfs_unlink pfs_rmdir pfs_rename pfs_link pfs_open pfs_read pfs_write pfs_release pfs_opendir pfs_readdir pfs_readdir pfs_access pfs_create	Individual call-back functions of the file system.
pfs_fullpath pfs_logpath	Provide the path to the appropriate files or the log files.
logread logwrite	Functions use to read and write to the log files required by pfs_write
servermain	The server main loop which wait and listen to request by the client
main	Main function to initialize the variables and parse the command line

The following flow-charts in Figure 35 and Figure 36 demonstrates the detailed explanation of a selected call-back function namely the write operation from both the server and client perspective.

Client side:

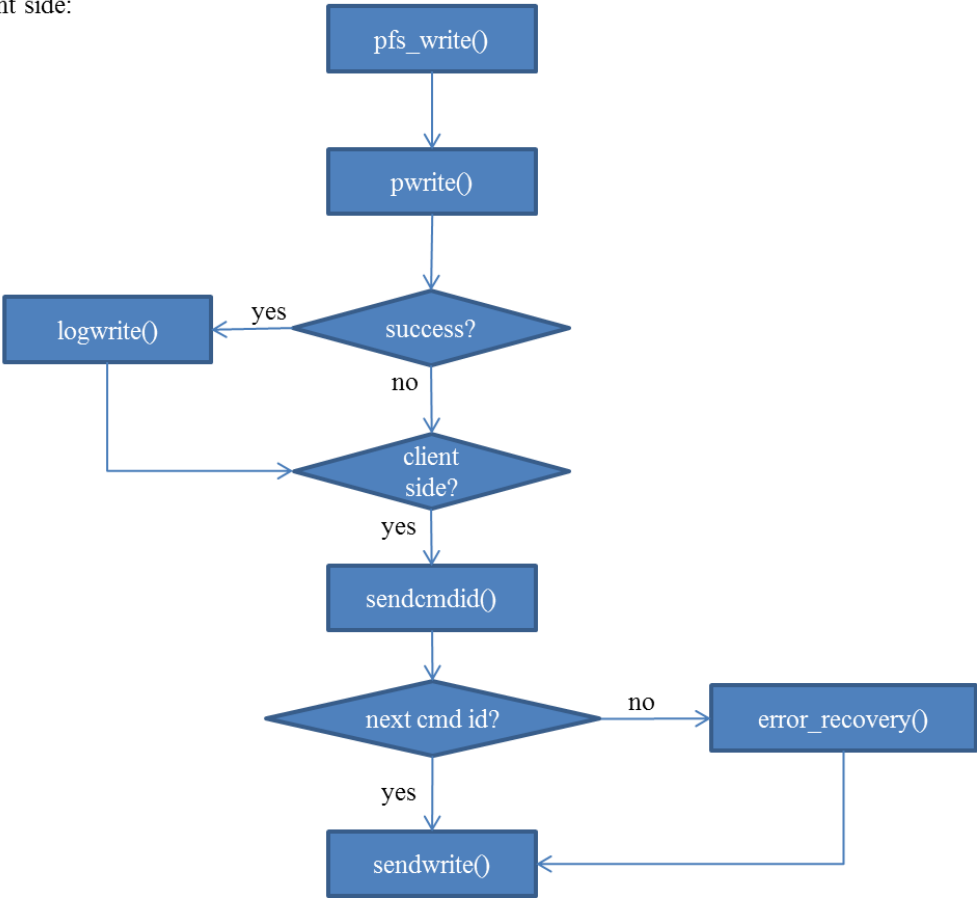
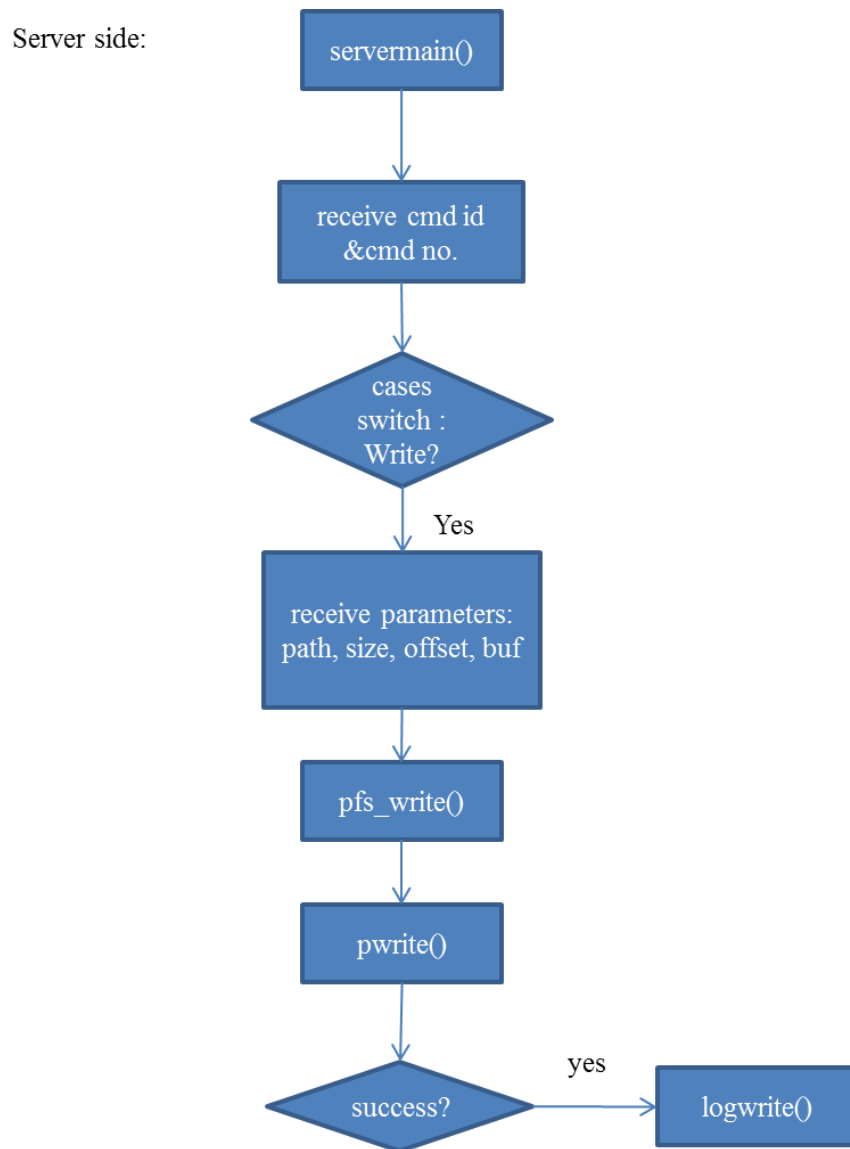


Figure 35 Flow-chart for write operation at client





*Figure 36 Flow-chart for write operation at server*

#### 4.4 Testing and Evaluation

In order to test the functionality of the implemented file system, two main approaches were thought of and used in this project. The first approach is to test the read and write latencies of the PFS and this was compared against the default file system in Linux. For the second test, the purpose is to ensure integrity in the PFS. This was done by performing numerous file creations at the client computer,

and checks were done on the server (which is the mirrored site) to observe whether all the same files were seen at the server after the file creations were performed on the client. In other words, this test aims to uncover any discrepancies between the files of both client and server.

#### 4.4.1 *Latency Test*

A test script (In Appendix G) was written to measure the latencies experienced during a read operation under the above two mentioned conditions, and the latencies was benchmarked against the default file system in Linux. For this latency test, files of various sizes ranging from 1KB to 50MB were being read by the client computer, and this was followed with the writing of these files as well. The graphs in Figure 37 and Figure 38 display the results for both the read and write latency tests.

#### 4.4.2 *Integrity Test for File System*

This integrity test was intentionally done on the PFS under vigorous operating condition so as to unveil the reliability of the PFS in a certain way. The test script (Detailed scripts are shown in the Appendix F) was run and this involved 10,000 file creations with varying sizes on the PFS.

Upon completion of the file creations and the mirroring, the Linux command:

```
diff -r -N </Path on Client> <Path on Server>
```

was issued to detect any discrepancies between the files in the client and server. However, no discrepancies were found, and all 10,000 files written on the client were mirrored on the server. This demonstrates that PFS is indeed reliable as it ensures file operations performed on the client are being mirrored accurately onto the server.

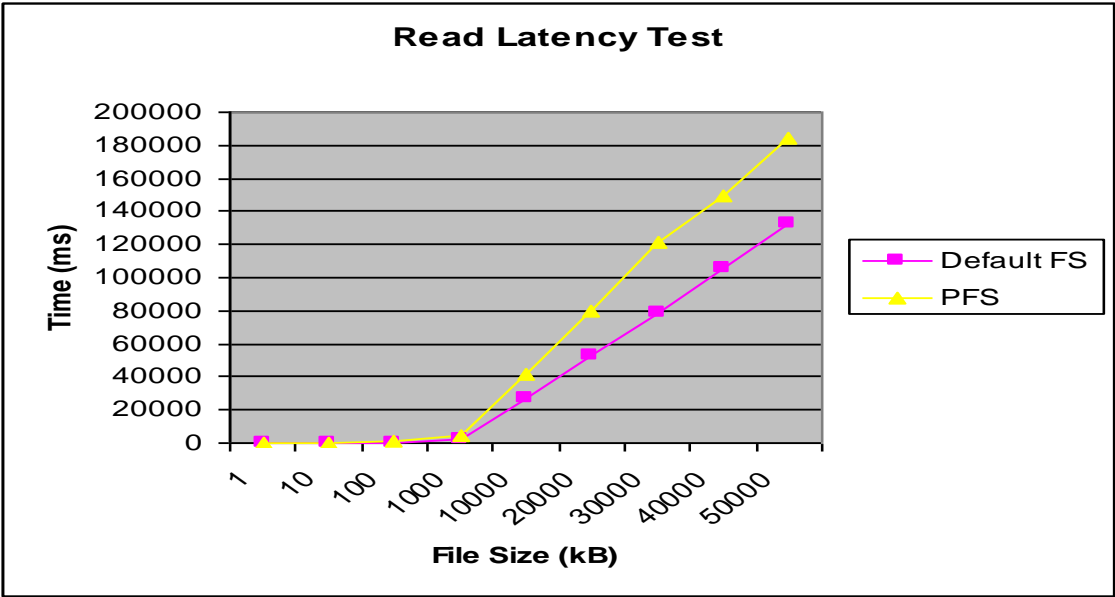


Figure 37 Graph for read latency test results

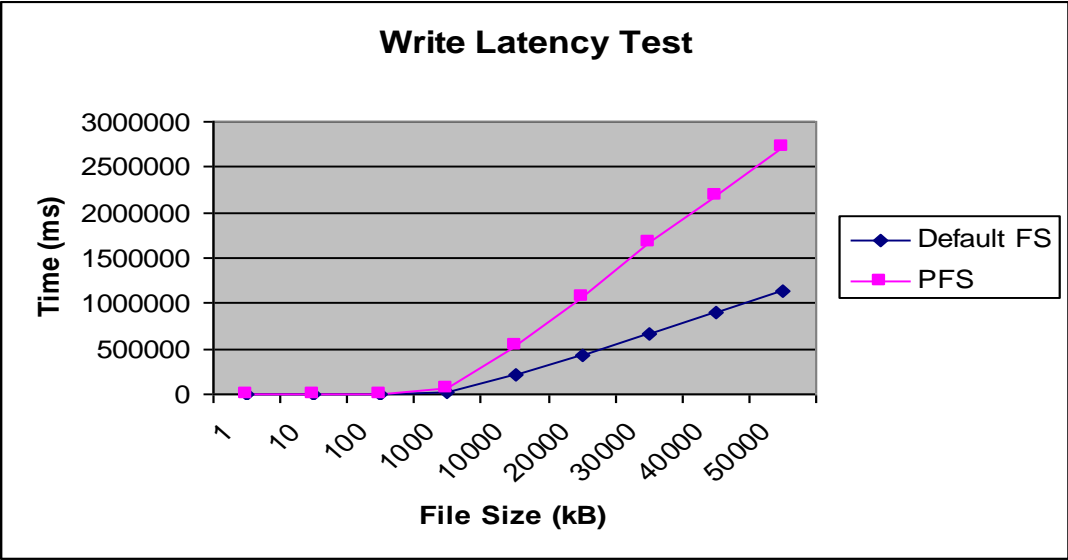


Figure 38 Graph for write latency test results

## 4.5 Summary

As a cluster is scaled up to large numbers of nodes and disks it becomes increasingly unlikely that all components are working correctly at all times. This implies the need to handle component failures gracefully and continue operating in the presence of failures. The proposed idea of implementing a reliable file system was implemented successfully in this work. Upon the completion of the development of the file system, testing and evaluation of the system were also performed and results showed that the implemented has reached a reasonable level of reliability.

# CHAPTER 5 IMPRECISE COMPUTATION SCHEDULING ALGORITHMS FOR REAL- TIME CLUSTER COMPUTING

---

---

## 5.1 Introduction

Cluster computing has attracted attention for large scale computing using idle CPU cycles of personal computers connected in local area network. In this thesis, a broker with imprecise computation scheduling is proposed for large scale computing in clusters. Imprecise computation techniques provide scheduling flexibility by trading off result quality to meet computation deadlines. It provides a technique to enhance QoS for real-time systems and improve the energy efficiency for large scale computing in clusters with lower carbon emission. Measurements of simulation on a large number of task sets show that imprecise computation improves the system reliability when scheduling intensive workloads. With less schedule faults, CPU cycles are saved and energy-efficiency is improved for large scale computing in clusters.

Many parallel applications have been developed to be running on cluster computing platforms. However, scheduling large scale data intensive work load in cluster computing is a challenging task. Scheduling strategies deployed in clusters have great impacts on overall system performance as it involves coordinating multiple computational nodes for resource sharing and scheduling in an efficient manner [71]. The jobs executed in cluster computing comprise many tasks. These tasks are allocated to PCs and processed in parallel. SETI@home [72] and distributed.net [73] are two well-known projects in this area.

Real time applications are required to perform their functions under strict timing constraints. A task missing its deadline may cause other tasks to miss their deadlines resulting in a system failure [74]. System failures and fault tolerant solution e.g. job replication in general result in wastage of CPU cycles. Imprecise computation technique is proposed as a natural means for enhancing fault tolerance and graceful degradation of real-time systems. Imprecise computation techniques provide scheduling flexibility by trading off result quality to meet computation deadlines [75]. For real time applications such as image processing, the user may accept timely fuzzy and approximate results. Therefore, the imprecise computation workload model has to adjust the trade-off between computation time and result quality. It is assumed that every task can be logically divided into two tasks: a mandatory task and an optional task. They are treated as tasks rather than subtasks. The ready times and deadlines of the tasks are the same as the job therefore any delay in a single task will affect the completion time of the whole job [76]. The broker has to monitor the task execution at each host, make sure all the tasks finish at the required deadline and perform appropriate actions according any change in execution [76]. The system will schedule and execute all the mandatory loads before their deadlines and then the optional loads to refine the result. In order to complete a job, all mandatory tasks that are executed on various hosts should be completed [76].

Gartner Report 2007 shows that IT industry contributes 2% of world's total CO<sub>2</sub> emissions. And U.S. EPA Report 2007 shows that 1.5% of total U.S. power consumption used by data centres which has more than doubled since 2000 and costs \$4.5 billion [77]. In the last decade, the issue of energy conservation for parallel application running on large-scale clusters has attracted little attention. Recently energy saving techniques has made it possible to develop energy efficient cluster computing platforms [77]. For example, dynamic voltage scaling scheme (DVS) and dynamic link shutdown (DLS), proposed by Kim et al. [78], cluster-based Energy-Saving Routing Algorithm (CERA) developed by Juan et al.

[79] and optimized buffer design to reduce energy consumption in cluster interconnects by Kim et al. [80].

In this research, a broker with imprecise computation scheduling is proposed for large scale computation in cluster computing. The imprecise computation application model (consisting a mandatory part and optional part) can be applied in many scenarios or use cases such as,

- Resource allocation in cluster to achieve load balancing and flexible use of cluster resource, which is to avoid overload certain nodes which are heavily loaded. If the source site is overloaded, that can be adopted that only mandatory part will be executed
- Fine-grained QoS specification: this model allows the administrator or user to specify the QoS by deciding the mandatory task ratio of the overall task, e.g. 90% or 80%. A user can describe a cluster application's QoS in more detail using the proposed application model. The minimum required quality is specified by the mandatory part
- Multimedia services in the cluster: application (e.g. multimedia application, image processing task) may require real time response. By using the imprecise computation model, the scheduling or timing fault could be reduced.
- A user can describe a cluster application's QoS in more detail using the proposed application model. The minimum required quality is specified by the mandatory part. The mandatory part is corresponded to the minimum quality. The optional part is to enhance the multimedia quality.

A technique to enhance QoS for real-time systems is provided to improve the energy efficiency for large scale computing in clusters with lower carbon emission. Green broker has two objectives in task scheduling of cluster computing: minimize job completion time and improve system energy efficiency

such that the carbon emission is reduced. Also four imprecise scheduling algorithms are designed and simulated, namely earliest deadline first (EDF), rate monotonic scheduling (RMS), least execution time first (LEF) and most execution time first (MEF) under varying system workload from 0 to 100% loading. Measurements of simulation on a large number of task sets show that imprecise computation improves the system reliability when scheduling intensive workloads. With less schedule timing faults, CPU cycles are saved and energy-efficiency is improved for computation of intensive work loads in clusters. It proves that our green broker is energy efficient by saving the CPU cycles wasted in the timing faults and gives user acceptable results approximately by using imprecise computation scheduling algorithms. The performance among four algorithms also shows that the EDF scheduling algorithm is the best scheduling algorithm in the real time system environment with intensive workloads. EDF scheduling algorithm is able to schedule 100% of the tasks when system is fully loaded. Using imprecise computation, when system is 100% loaded, RMS scheduling algorithm is able to schedule 62.3% more tasks; LEF scheduling algorithm is able to schedule 77.6% more tasks; MEF scheduling algorithm is able to schedule 10.3% more tasks.

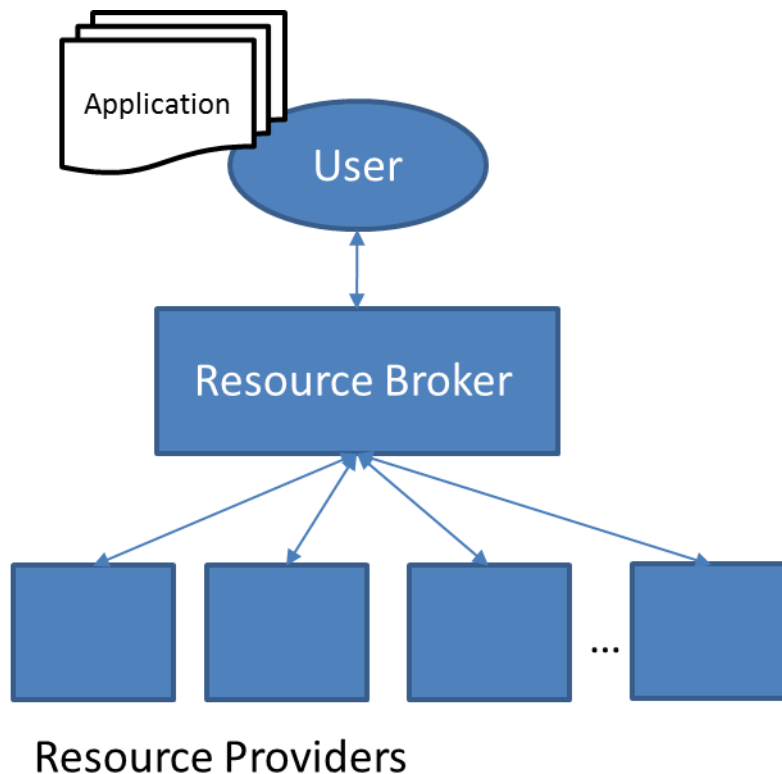
## 5.2 System Model

A generic cluster computing system architecture is proposed. A green broker works as a middle layer on top of operating systems for multiple user management and resource management in Figure 39 Cluster computing system overview. And it behaves like an agent to receive and manage tasks from multiple clients and provide a single view for them. Also, it allows resources to be remotely accessed by multiple clients without interfering with the person sitting at the computer where the application is installed [81] [82] [83]. Green broker is responsible to process the computationally intensive loads and schedule its load



on all the sinks including itself. Sinks are the computing nodes for processing workloads or data [71].

The cluster computing system interconnection topology is modelled as a single-level tree network shown in Figure 40 Cluster computing system model. The cluster system consists of a broker which is the master node denoted by  $P_0$  and  $m$  sinks (processing nodes) denoted by  $P_1 \dots P_m$ . Each node is a processor with front-end [84] that means every node is capable in job admission, assignment and processing. It is assumed there are  $m+1$  processor ( $p_0, p_1, p_2 \dots p_m$ ) and  $m$  links. The root processor receives the arrival load and partition and distribute of the load to all the processors.



*Figure 39 Cluster computing system overview*

A divisible load is one that can be arbitrarily partitioned among the processors in a system.  $\omega$  is a processor's computation speed parameter respect to a standard

processor.  $z$  is a parameter for the communication link speed in a distributed computing system. All the nodes have the different computational speeds and are fully connected by communication links with different speeds [84]. Notations and definitions of the cluster computing system model are shown in Table 8 Notations and definitions of the System.

Our system assumes that every task can be logically divided into two tasks: a mandatory task and an optional task. The system will schedule and execute all the mandatory loads before their deadlines and then the optional loads to refine the result. Each task  $T$  can be decomposed into two subtasks: the mandatory subtask  $M$  and the optional subtask  $O$ .  $M$  and  $O$  are treated as tasks rather than subtasks. The processing times of  $M$  and  $O$  are  $m$  and  $o$ , respectively and  $m + o = \tau$  (Tau). The ready times and deadlines of  $M$  and  $O$  are the same as those of  $T$ .

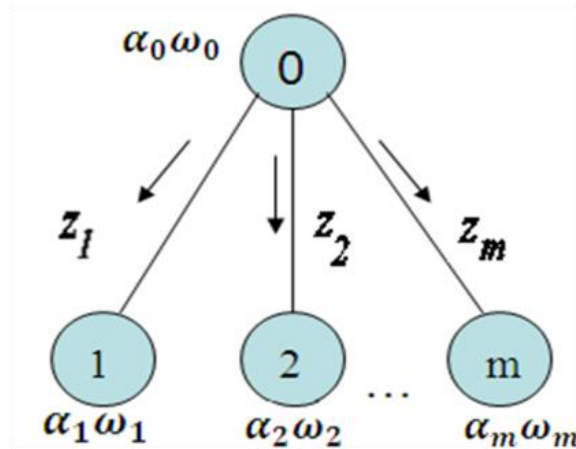


Figure 40 Cluster computing system model

Table 8 Notations and definitions of the System

T	Task
M	Mandatory task
O	Optional task

$\tau$	Processing time of $T_m + o = \tau$ (Tau)
$m$	Processing time of M
$o$	Processing time of O
P	Processor
$\omega$	Processor's computation speed parameter
$z$	Communication link speed parameter
$\alpha$	Load fraction assigned to processor, $\sum \alpha_i = 1$
$T_{cp}$	Time taken to process a unit load by the standard processor
$T_{cm}$	Time taken to communicate a unit load on a standard link

### 5.3 Scheduling Method and Modelling

#### 5.3.1 Scheduling Algorithms

In our system four scheduling algorithms are designed. They are all priority driven and pre-emptive. Earliest Deadline First (EDF) scheduling is the dynamic priority driven scheduling algorithm used in real time systems. The system checks the deadline of the tasks at any task arrival time and the task with the earliest deadline will be chosen. Rate Monotonic Scheduling (RMS) algorithm is a static scheduling algorithm. The priority is assigned according to the periods of the tasks. Most Execution Time First (MEF) and Least Execution Time First (LEF) will assign the priorities according to the amount of execution time taken by the mandatory tasks. MEF will schedule and execute the task with longest execution time of mandatory part first. While LEF will schedule and execute the

task with the shortest execution time of mandatory part first. The system will schedule and execute all the mandatory tasks first before their deadlines and then the optional tasks as shown in Figure 41 Timing diagram of the system. T1, T2 and T3 are three periodic tasks. P0, P1, P2 and P3 are four parallel processing nodes in the cluster. M denotes the mandatory part of a task and O denotes the optional part of a task. And time intervals are defined by all the deadlines of the tasks which are the same as next tasks' arrival time. In Figure 41, in a time interval mandatory tasks are scheduled based on their priorities. Priority assignment varies for different scheduling algorithms. In the example, mandatory tasks (T1.M, T2.M and T3.M) are executed in order followed by the optional tasks (T1.O, T2.O and T3.O). Optional tasks can be left uncompleted when the deadline comes or a new task arrives. Since our tasks are periodic only the scheduling in the least common multiple of all the periods in the task set will be considered.

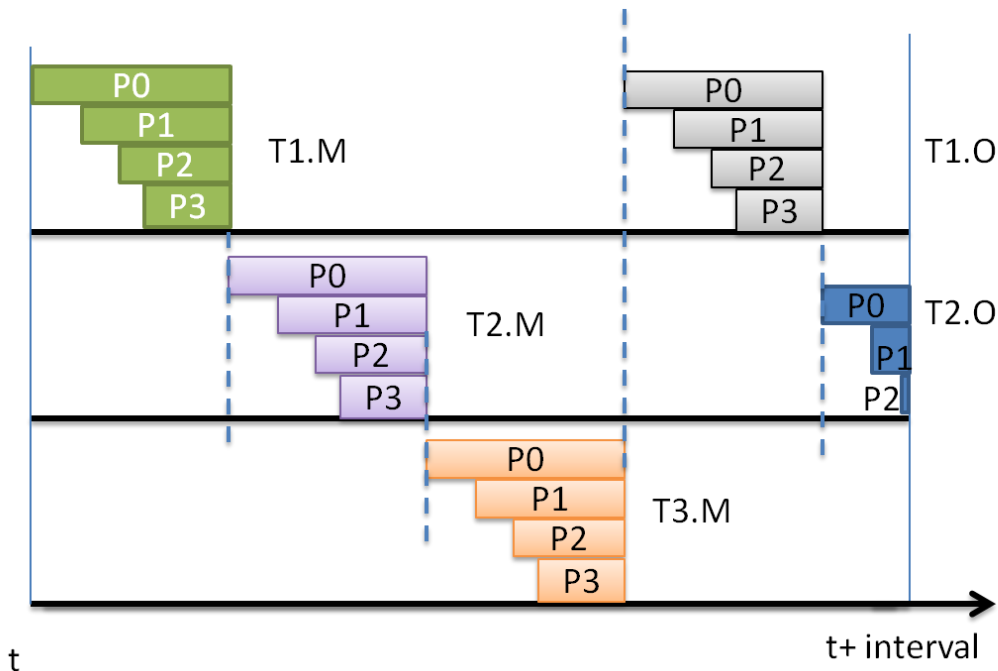


Figure 41 Timing diagram of the system

### 5.3.2 Optimal Load Distribution

In order to achieve optimal processing time in a cluster computing network or linear network, the processing load must be scheduled such that all the processors stop computing their loads at the same time, which is named the principle of optimality [84]. Timing diagram for optimal load divisible for a cluster of processing nodes and its equivalent network are shown in Figure 42 and Figure 43.

Original Cluster

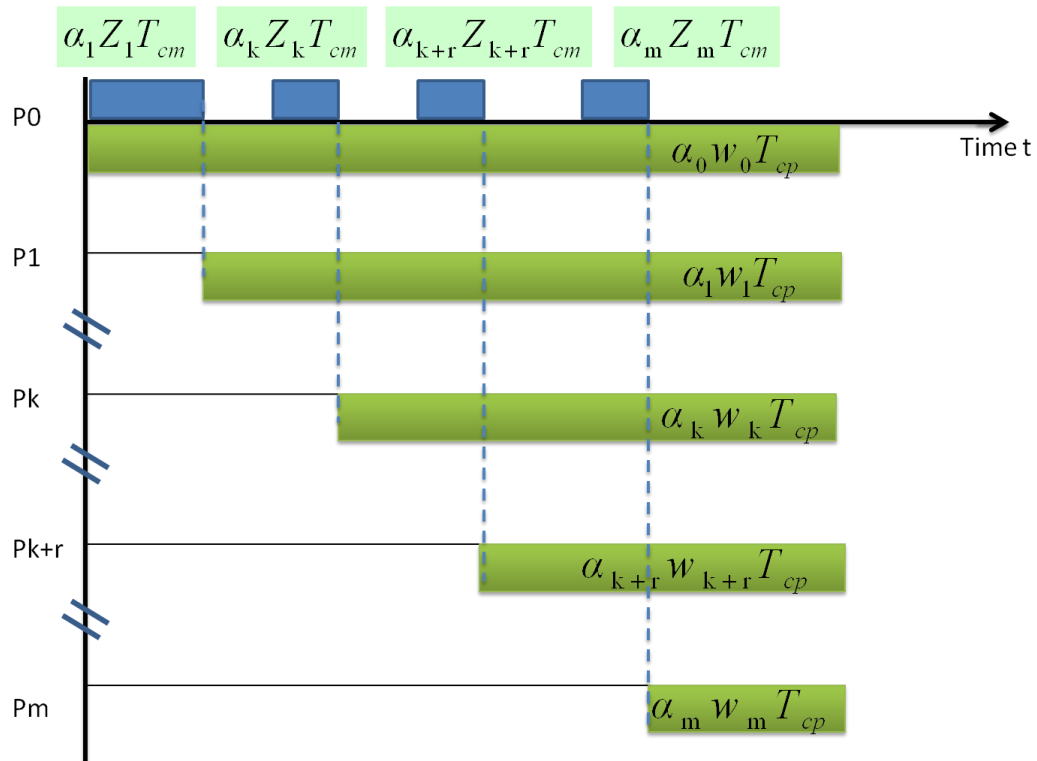


Figure 42 Timing diagram: optimal load divisible for a cluster of processing nodes [84]

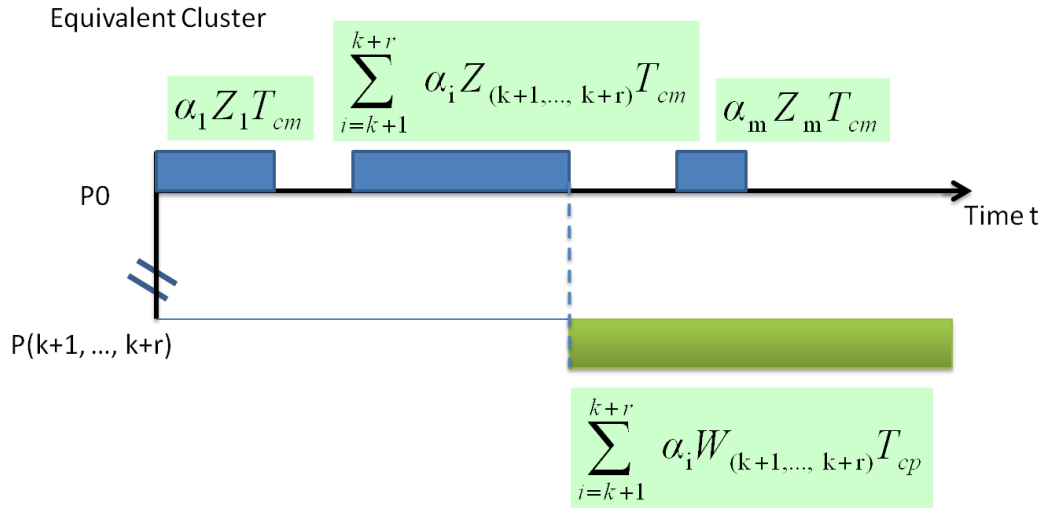


Figure 43 Timing diagram: optimal load divisible for equivalent cluster network [84]

The optimal load distribution can be solved by recursive equations provided in the scheduling divisible load book by B. Veeravalli et al. [84].

Equivalent W:

$$W(k+1, \dots, k+r) = \left( \frac{1}{1 + \sum_{i=k+2}^{k+r} \prod_{p=i}^{k+r} f_p} \right) W_{k+r} \quad (1)$$

Recursive function:

$$a_0 w_0 T_{cp} = a_1 Z_1 T_{cm} + a_1 w_1 T_{cp} \quad (2)$$

$$a_{i-1} Z_{i-1} T_{cm} + a_{i-1} w_{i-1} T_{cp} = a_i Z_i T_{cm} + a_i w_i T_{cp}$$

$$i = 2, 3, \dots, m \quad (3)$$

Normalization equation:

$$a_0 + a_1 + a_3 + \dots + a_m = 1 \quad (4)$$

#### 5.4 ICSCluster Simulator

ICSCluster is designed as the simulation platform of the system. The simulator is programmed using GNU Octave which is a high-level language, primarily

intended for numerical computations [85]. The block diagram of the simulator is shown as in Figure 44.

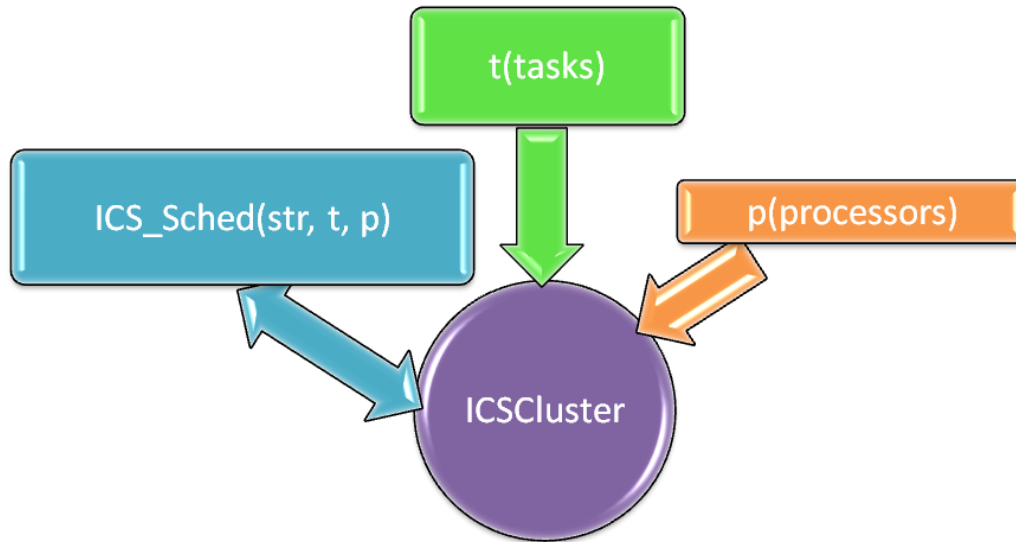


Figure 44 Block diagram of the simulator

In Table 9 Structure array and fields' definition, task set  $t$  (tasks) can be user specified or generated from our task generator.  $t$  (tasks),  $p$  (processors) and scheduling algorithm  $str$  are passed to scheduling module  $ICS\_sched(str, t, p)$ .  $ICS\_sched$  is doing the actual schedule work and returns solution ( $sol$ ). Solution contains all the information of the system at a time interval  $t1$  to  $t2$  including the communication information and the computation information of each task on each processor.

Table 9 Structure array and fields' definition

Structure Array	Fields
$t(tasks)$	$tid, m, \tau, \pi, p(priority)$
$p(processors)$	$pid, w, z$
$sol(solution)=simics\_sched(str, t, p)$	$t1, t2, type, pid, tid, amount$

In Table 9 Structure array and fields' definition, task is specified by the task identification number  $tid$ , mandatory portion execution time  $m$ , task execution time  $\tau$ , task period  $p_i$  and the task priority  $p$  used in error calculation. Processor is defined by processor identification number  $pid$ , processor's computation speed parameter  $\omega$  and communication link speed parameter  $z$ . The scheduler will return the solution in the format with time interval  $(t_1, t_2)$ , communication or computation carried out type, which processor  $pid$  is processing which task  $tid$  and the amount of processing time given to the task.

The system schedules and executes the mandatory tasks first based on the scheduling algorithms introduced in previous section. For mandatory tasks, it will use EDF, RMS, LEF and MEF. The optional tasks are scheduled based on their weights or priorities. Our schedule system is to achieve the maximum task schedulable rate and to improve the energy efficiency. Figure 45 Flow chart of simulation imprecise computation scheduling demonstrates the flow of simulation imprecise computation scheduling. When the simulation starts, the system first determine which scheduling algorithm is chosen to be simulated from EDF, RMS, LEF and MEF. Table 10 summarizes all the scheduling algorithms used in the simulator.



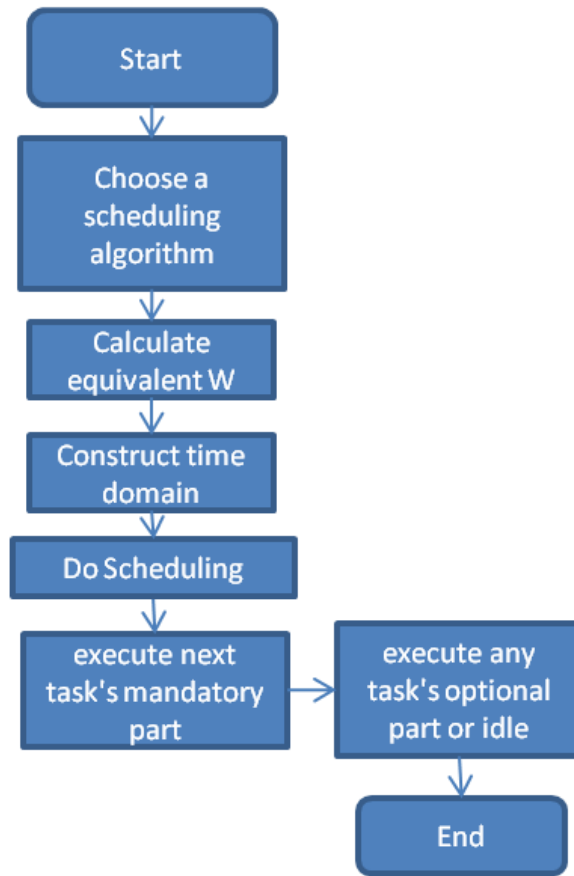


Figure 45 Flow chart of simulation imprecise computation scheduling

Table 10 Scheduling algorithms for mandatory and optional tasks

String(str)	Mandatory	Optional
Algo 1	EDF	Highest Priority First
Algo 2	RMS	Highest Priority First
Algo 3	LEF	Highest Priority First
Algo 4	MEF	Highest Priority First

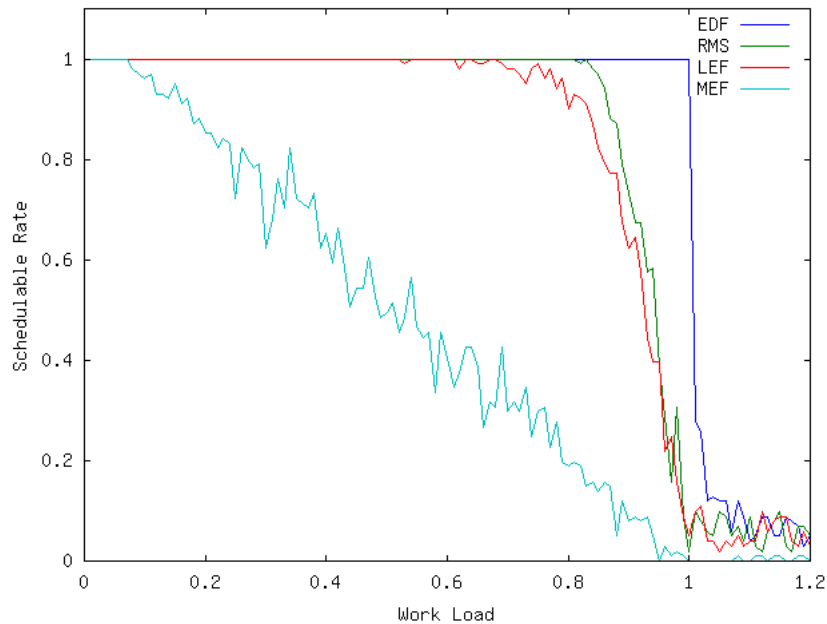
Then equivalent  $W$  is calculated based on optimal load distribution equations (1), (2), (3) and (4). After that, time domain is constructed according to all the tasks' periods in the task sets and their least common multiple. The scheduling will start if all the previous jobs are successful. Mandatory tasks are scheduled before

optional tasks. If all the mandatory tasks are finished before their deadlines, the optional tasks will execute based on their priorities. If the tasks are successfully scheduled, a solution returns to the scheduler with all the information of the system within a time domain including the communication information and the computation information of each task on each processor.

## 5.5 Results and Analysis

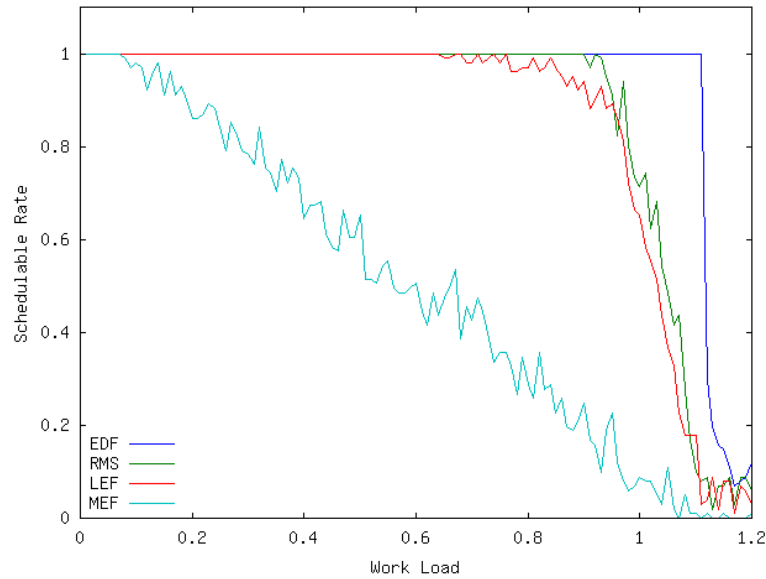
In our evaluation, the program randomly generates 100 task sets each time for a certain workload and increase workload from 0.01 to 1.20 with increment step of 0.01 to test the schedulable rate for each algorithm. If the workload cannot be schedule within the constrain or deadline, that means it will create a timing fault and lead to energy wastage. Work load is defined as the sum of computation time required over period for a task set,  $\sum\tau/\pi$ , against the equivalent computation capacity of the multiprocessor system. Work load describes the intensity of system loads. Schedulable rate is defined as the percentage of task sets which pass the scheduling test.

Firstly, a comparison is done on the behaviour of scheduling intensive workloads among four different scheduling algorithms. The schedulable rate vs. work load is plotted in Figure 46.



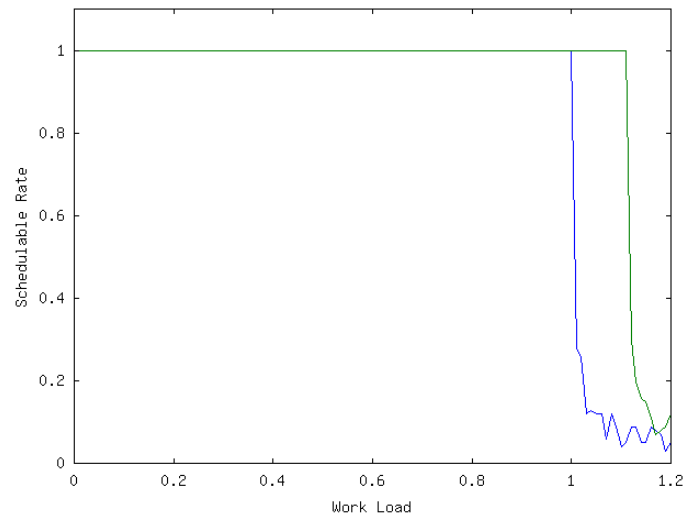
*Figure 46 Schedulable rates vs. work load for precise scheduling*

Next, the performance evaluation between different scheduling algorithms of imprecise computation is shown in Figure 47. It can be observed that when work load increases, schedulable rate behaves differently among the four scheduling algorithms. It is obvious that the differences among the four scheduling algorithms lie in their capabilities of scheduling intense workloads. Results show that EDF has the best performance among these algorithms. Its schedulable rate doesn't drop until work load reaches 1, and maintains the highest schedulable rate in four algorithms. RMS, which is considered as the best static priority-driven algorithm, shows satisfactory results as well. MEF as a comparison algorithm performs the worst among four algorithms.



*Figure 47 Schedulable rates vs. workload for imprecise computation*

Next, for the four algorithms introduced above the performances of imprecise computation against precise computation are shown in Figure 48-Figure 51. The effect and benefit of imprecise computation are investigated.



*Figure 48 Comparison between precise and imprecise computation on schedulable rates for EDF scheduling algorithms*

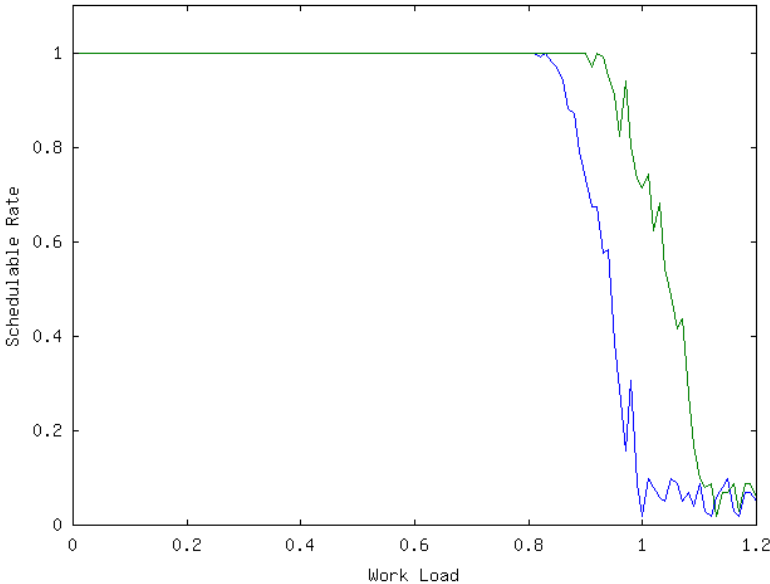


Figure 49 Comparison between precise and imprecise computation on schedulable rates for RMS scheduling algorithms

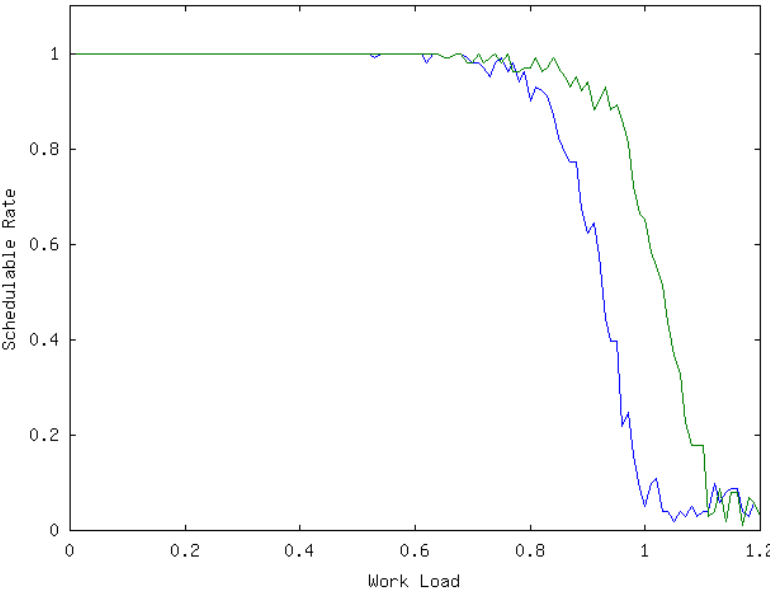
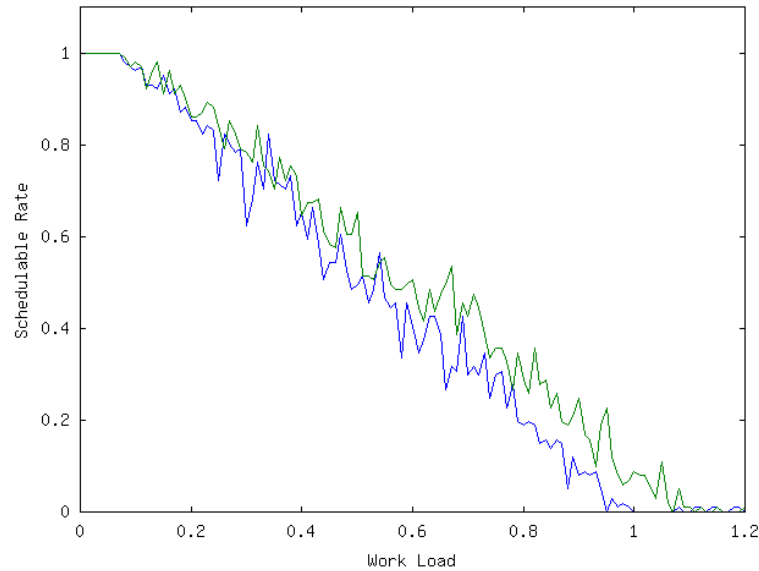


Figure 50 Comparison between precise and imprecise computation on schedulable rates for LEF scheduling algorithms



*Figure 51 Comparison between precise and imprecise computation on schedulable rates for MEF scheduling algorithms*

In these evaluations, it is assumed that a task can be logically decomposed into a mandatory part which takes 90% of execution time  $\tau$  and an optional part which takes 10% of  $\tau$ . Clearly, for imprecise computation a system with certain workload can reach a higher schedulable rate in contrast to precise computation. This is because imprecise computation can left some optional work unfinished and return an acceptable solution. So, imprecise computation is possible to 1.11 times higher schedulable workloads as before. Therefore with the green broker, our cluster computing system provides higher QoS and lowers the timing faults which lead to lower energy consumption. In the figures, the minor spikes of precise computation beyond the work load of 1 and imprecise computation beyond 1.11 are considered as experimental errors. As an observation of the four algorithms stated above, they only differs in the scheduling strategy on mandatory parts as the same priority driven algorithm is used for optional tasks. The performance among four algorithms also shows that the EDF scheduling algorithm is the best scheduling algorithm in the real time system environment with intensive workloads. EDF scheduling algorithm is able to schedule 100% of

the tasks when system is fully loaded. Using imprecise computation, when system is 100% loaded, RMS scheduling algorithm is able to schedule 62.3% more tasks; LEF scheduling algorithm is able to schedule 77.6% more tasks; MEF scheduling algorithm is able to schedule 10.3% more tasks.

## 5.6 Summary

A green broker has been proposed in the thesis with imprecise computation scheduling for large scale computation in cluster computing. A technique to enhance QoS for real-time systems has been provided to improve the energy efficiency for large scale computing in clusters with lower carbon emission. Green broker has achieved objectives: minimize job completion time, improve system energy efficiency and reduce the carbon emission. Also four scheduling algorithms with imprecise computation task model under varying system workload from 0 to 100% loading are designed and simulated. Measurements of simulation on a large number of task sets show that imprecise computation improves the system reliability when scheduling intensive workloads. With less schedule timing faults, CPU cycles are saved and energy-efficiency is improved for computation of intensive work loads in clusters. It proves that our green broker is energy efficient by saving the CPU cycles wasted in the timing faults and gives user acceptable results approximately by using imprecise computation scheduling algorithms. The performance comparisons among four algorithms show that EDF scheduling algorithm is the best algorithm in the real time system environment when dealing with intensive workloads.

## CHAPTER 6 CONCLUSIONS AND FUTURE WORK

---

---

### 6.1 Conclusions

In this research, a generic application sharing architecture was proposed for users' application sharing in a cluster of closed operating systems such as Microsoft Windows. The broker-mediated solution allows multiple users to access a single user software license on a time multiplex basis through a single logged in user. An application sharing tool called ShAppliT has been introduced and implemented in Microsoft Windows operating system. Their performance has been evaluated on CPU usage and memory consumption when a computer is hosting multiple concurrent shared application sessions

Moreover, imprecise computation scheduling was modelled and simulated to enhance QoS for real-time systems and improve the energy efficiency for large scale computing in clusters. Measurements of simulation on a large number of task sets showed that imprecise computation improved the system reliability when scheduling intensive workloads with less schedule timing faults, CPU cycles and energy-efficiency improvement.

Finally, a failure-save solution was implemented for fault-tolerant application services in clusters which enabled user to login to the file server from anywhere, synchronize document to last saved state on server and provide certain degree of portability. The proposed idea of building a reliable file system was implemented successfully. Testing and evaluation of the system were also performed and results showed that the implemented had reached reasonable level of reliability.



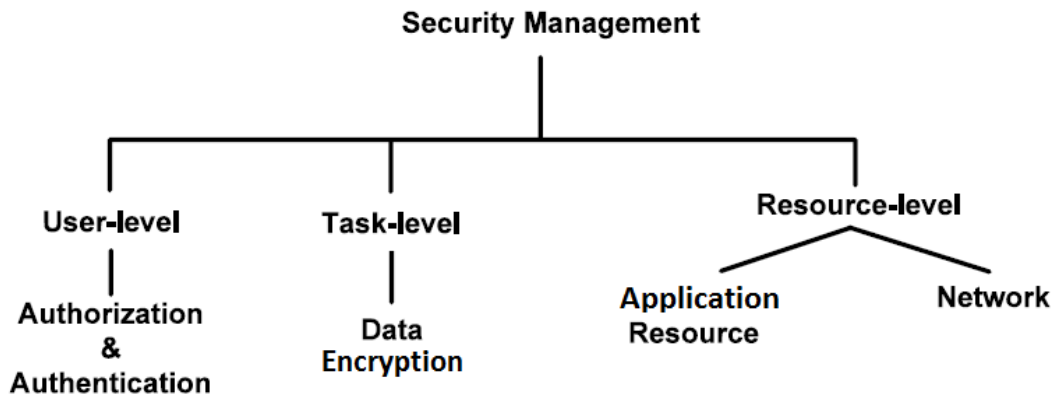
## 6.2 Future Work

Future research works are able to be carried out on security management, reliability and resource management for P2P application sharing in a cluster environment. User identification, data encryption algorithms and incentive mechanisms are ways to prevent free-riding and promote cooperation across distrustful peers [89]. In addition, a successful application sharing system should also provide reliable services. Peers can build up coordinated checkpoints [90] for fault recovery and establish redundant links across the peers in case of network failures. In addition, resource management plays a critical rule in P2P application sharing [91]. The research problem for resource discovery is matchmaking [92] that locates resources subject to certain constraints.

Next each of the possible future work will be discussed in detail.

### 6.2.1 *Security Management*

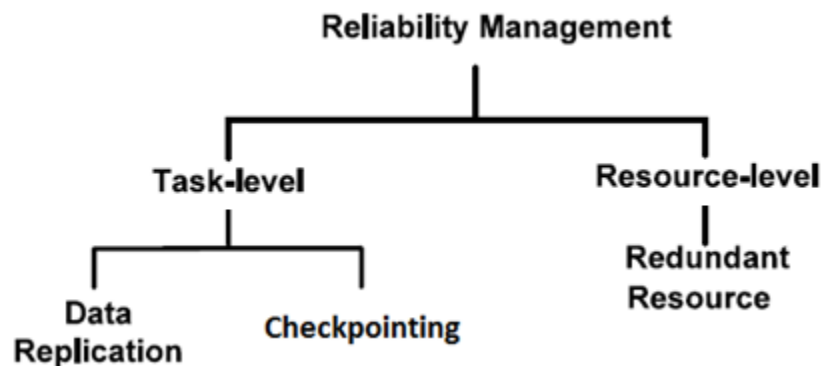
In this part categorize security protection technologies can be applied to various levels in a P2P App Share system. Security management in user level mostly relies on user identity verification. User identification provides a screening process for certified peer management. At task level security management concerns the aspect of task data privacy protection, commonly achieved through data encryption algorithms. In addition, systems must provide effective incentive mechanisms to prevent free-riding and promote cooperation across distrustful peers. Finally, network disconnection can be used if remote attacks from certain source are identified.



*Figure 52 Taxonomy for security management*

### 6.2.2 Reliability Management

In addition to secure management, a successful application sharing system should also provide reliable services. One chief technology to accomplish fault-tolerant application services is data replication at client or server or third peer. Moreover, peers can build up coordinated checkpoints for fault recovery and establish redundant links across the peers in case of network failures. So, process at the failed peer can be migrated to a peer with redundant resource for fault tolerance.

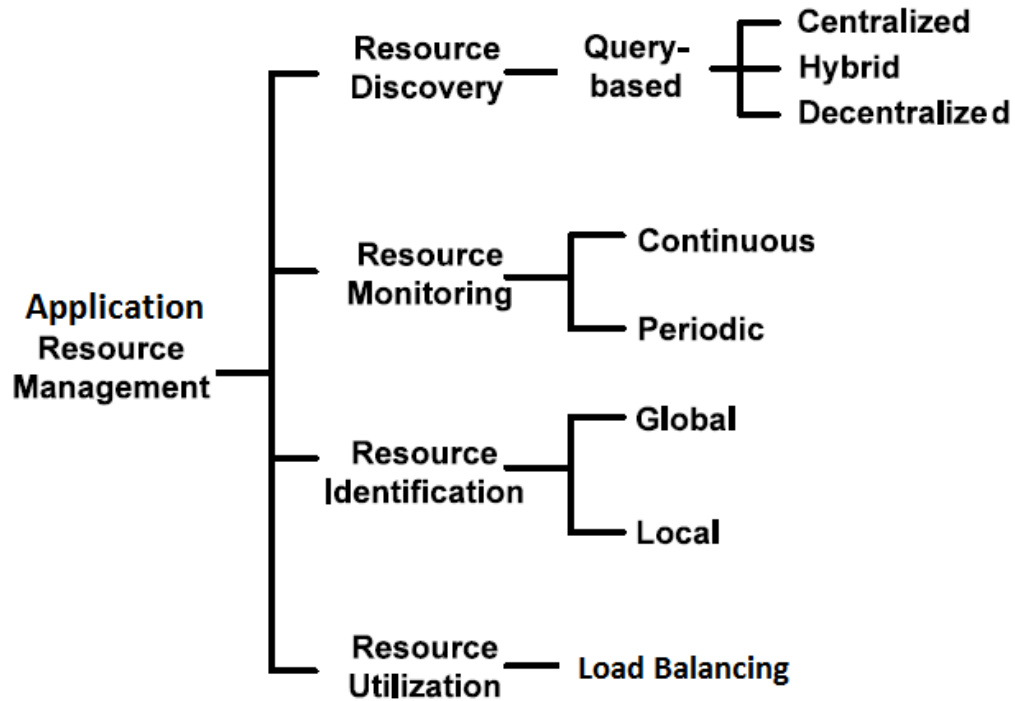


*Figure 53 Taxonomy for reliability management*

### 6.2.3 Resource Management

Resource management plays a critical rule in P2P application sharing. For computing resources four core functions are generalized: resource discovery,

resource monitoring, resource identification and resource utilization. The research problem for resource discovery is matchmaking that locates resources subject to certain constraints. Resource utilization concerns how the management functions affect resource providers. Load balancing can be applied for better resource utilization.



*Figure 54 Taxonomy for resource management*

## BIBLIOGRAPHY

---

- [1] R. Buyya, "A Proposal for Creating a Computing Research Repository (CoRR, <http://www.arXiv.org/>) on Cluster Computing," Monash University, Melbourne, Australia, 2000.
- [2] B. Mark, B. Rajkumar, H. Ken, J. Heath and J. and Hai, "Cluster Computing R&D in Australia," 2000. [Online]. Available: <http://www.cloudbus.org/papers/ClusterComputingAU.pdf>. [Accessed 1 Jun 2013].
- [3] B. Mark, B. Rajkumar and H. Dan, "Cluster Computing: A High-Performance Contender," [Online]. Available: <http://arxiv.org/ftp/cs/papers/0009/0009020.pdf>. [Accessed 1 Jun 2013].
- [4] F. Ian, Z. Yong, R. Ioan and L. Shiyong, "Cloud Computing and Grid Computing 360-Degree Compared," in *Grid Computing Environments Workshop GCE '08*, 2008.
- [5] B. Rajkumar, J. Hai and C. Toni, "Cluster computing," *Elsevier Science: Future Generation Computer Systems*, vol. 18, 2002.
- [6] Digipede Technologies, "Grid and Cluster Computing: Options for Improving Windows Application Performance," 2003. [Online]. Available: [http://www.digipede.net/downloads/Digipede\\_CCS\\_Whitepaper.pdf](http://www.digipede.net/downloads/Digipede_CCS_Whitepaper.pdf). [Accessed 1 Jun 2013].
- [7] K. Hwang, "Network-Based Cluster Computing," 2000. [Online]. Available: <http://www-classes.usc.edu/engr/ee-s/657h/clusterbasic.pdf>. [Accessed 1 Jun 2013].
- [8] K. Chander, "Linux Compute Clusters," 2003. [Online]. Available: <http://linuxclusters.com>. [Accessed 1 Jun 2013].
- [9] P. David A. and H. John L., "Computer Organization and Design," ISBN 0-12-374750-3, 2011, pp. 641-642.
- [10] F. Armando, G. Steven D., C. Yatin, B. Eric A. and G. Paul, "Cluster-Based Scalable Network Services," in *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, Saint-Malo, France, 1997.
- [11] K. Shirahata, "Hybrid Map Task Scheduling for GPU-Based Heterogeneous Clusters in: Cloud Computing Technology and Science," in *CloudCom*, 2010.

- [12] B. Omer and S. Henning, "Application and Desktop Sharing," in *CoNEXT'07*, New York, NY, U.S.A, 2007.
- [13] "Windows NetMeeting," [Online]. Available: [www.microsoft.com/windows/netmeeting](http://www.microsoft.com/windows/netmeeting). [Accessed 1 Jun 2013].
- [14] B. Omer and S. Henning, "BASS Application Sharing System," in *10th IEEE International Symposium on Multimedia*, 2008.
- [15] T. Kasame, K. Pakit and M. Veera, "Jamjuree Cluster: A Peer-to-Peer Cluster Computing System," *Lecture Notes in Computer Science*, Vols. Network-Based Information Systems, pp. pp 375-384, 2007.
- [16] H. Park, R. Izhak-Ratzin and M. van der Schaar, "Peer-to-Peer Networks - Protocols, Cooperation and Competition," *Streaming Media Architectures, Techniques, and Applications: Recent Advances*, 2010.
- [17] "Technical Overview of Windows Server 2003 Terminal Services, Microsoft Corporation," [Online]. Available: <http://www.microsoft.com/windowsserver2003/techinfo/overview/termserv.msp>. [Accessed 1 Jun 2013].
- [18] [Online]. Available: <http://www.beowulf.org/>. [Accessed 1 Jun 2013].
- [19] S. Mulyadi and S. Andreas, "Build a heterogeneous cluster with coLinux and openMosix," *IBM Developer Works*, 2005.
- [20] H. Forrest M. and H. William W., "High Performance Computing: An Introduction to Parallel Programming With Beowulf," [Online]. Available: <http://www.climate modeling.org/~forrest/osdj-2000-11/>. [Accessed 1 6 2013].
- [21] [Online]. Available: <http://research.cs.wisc.edu/htcondor/>. [Accessed 1 Jun 2013].
- [22] "DMTCP: Distributed MultiThreaded CheckPointing," [Online]. Available: <http://dmtcp.sourceforge.net/>. [Accessed 1 Jun 2013].
- [23] "TightVNC," [Online]. Available: <http://www.tightvnc.com/>. [Accessed 1 Jun 2013].
- [24] "Open MPI: Open Source High Performance Computing," [Online]. Available: <http://www.open-mpi.org/>. [Accessed 1 Jun 2013].
- [25] "MPICH," [Online]. Available: <http://www.mpich.org/>. [Accessed 1 Jun 2013].
- [26] "python," [Online]. Available: <http://www.python.org/>. [Accessed 1 Jun 2013].
- [27] R. Ma, C.-L. Wang and F. C. M. Lau, "M-JavaMPI: A Java-MPI Binding with Process Migration Support," in *2nd IEEE International Symposium on Cluster Computing and the Grid*, 2002.
- [28] M. Dobber, R. van der Mei and G. Koole, "Dynamic Load Balancing and Job Replication in a Global-Scale Grid Environment: A Comparison," 2009.

- [29] "HP Insight Cluster Management Utility," [Online]. Available: [http://h20311.www2.hp.com/HPC/cache/412128-0-0-0-121.html?jumpid=ex\\_r1459\\_w1/en/large/tsg/go\\_cmu](http://h20311.www2.hp.com/HPC/cache/412128-0-0-0-121.html?jumpid=ex_r1459_w1/en/large/tsg/go_cmu). [Accessed 1 Jun 2013].
- [30] H. Victor, "Cluster Computing: A Survey and Tutorial," [Online]. Available: [http://www.sdsc.edu/~victor/Projects/NQE/SysAdmin/SysAdmin\\_Batch.html](http://www.sdsc.edu/~victor/Projects/NQE/SysAdmin/SysAdmin_Batch.html). [Accessed 1 Jun 2013].
- [31] P. Ziewer and H. Seidl, "Transparent TeleTeaching," in *ASCILITE*, Auckland, New Zealand, 2002.
- [32] G. Lewis, S. M. Hasan, V. N. Alexandrov and M. T. Dove, "Facilitating collaboration and application sharing with MAST and the access grid development infrastructures," in *E-SCIENCE*, 2006.
- [33] "Apple Remote Desktop 3," [Online]. Available: <http://www.apple.com/sg/remotedesktop/>. [Accessed 1 Jun 2013].
- [34] "GoToMyPC," [Online]. Available: [http://www.gotomypc.com/remote\\_access/remote\\_access](http://www.gotomypc.com/remote_access/remote_access). [Accessed 1 Jun 2013].
- [35] "Cendio ThinLinc," [Online]. Available: <http://www.cendio.com/products/thinlinc/>. [Accessed 1 Jun 2013].
- [36] "The VNC family of Remote Control Applications," [Online]. Available: [http://ipinfo.info/html/vnc\\_remote\\_control.php](http://ipinfo.info/html/vnc_remote_control.php). [Accessed 1 Jun 2013].
- [37] "Citrix System," [Online]. Available: [www.citrix.com](http://www.citrix.com). [Accessed 1 Jun 2013].
- [38] "VNC," [Online]. Available: [www.realvnc.com](http://www.realvnc.com). [Accessed 1 Jun 2013].
- [39] G. Wallace and K. Li, "Virtually shared displays and user input devices," in *USENIX Annual Technical Conference*, 2007.
- [40] "The X Windows System," [Online]. Available: <http://www.opengroup.org/tech/desktop/x/>. [Accessed 1 Jun 2013].
- [41] J. E. Baldeschwieler, T. Gutekunst and B. Plattner, "A survey of x protocol multiplexors," *SIGCOMM Comput. Commun. Rev.*, vol. 23, no. 2, pp. 16-24, 1993.
- [42] "FreeRDP/xrdp," [Online]. Available: <https://github.com/FreeRDP/xrdp>. [Accessed 1 Jun 2013].
- [43] "Chrome Remote Desktop," [Online]. Available: <https://chrome.google.com/webstore/detail/chrome-remote-desktop/gbchcmhahfdphkxkmpfmihenigmpp?hl=en>. [Accessed 1 Jun 2013].
- [44] MSDN, "Remote Desktop Protocol," [Online]. Available: [http://msdn.microsoft.com/en-us/library/cc240446\(v=PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/cc240446(v=PROT.10).aspx). [Accessed 1 Jun 2013].
- [45] "Ericom Blaze RDP," [Online]. Available:

- [http://www.ericom.com/ericom\\_blaze.asp](http://www.ericom.com/ericom_blaze.asp). [Accessed 1 Jun 2013].
- [46] "N.Able," [Online]. Available: <http://www.n-able.com/products/n-central/>. [Accessed 1 Jun 2013].
- [47] "RapidSupport - Remote Support Solution," [Online]. Available: <http://www.rapidsupport.net/pages/index.php>. [Accessed 1 Jun 2013].
- [48] "Team Viewer," [Online]. Available: <http://www.teamviewer.com/en/index.aspx>. [Accessed 1 Jun 2013].
- [49] "xrdp," [Online]. Available: <http://www.xrdp.org/>. [Accessed 1 Jun 2013].
- [50] T. Richardson, "The RFB Protocol," RealVNC Ltd, 2009.
- [51] "ITU-T T.128," [Online]. Available: <http://www.itu.int/rec/recommendation.asp>. [Accessed 1 Jun 2013].
- [52] "SunForum," [Online]. Available: [www.sun.com/desktop/products/software/sunforum](http://www.sun.com/desktop/products/software/sunforum). [Accessed 1 Jun 2013].
- [53] L. Hui-Chieh, C. Yen-Ping, S. Ruey-Kai and L. Win-Tsung, "A Generic Application Sharing Architecture Based on Message-Oriented Middleware Platform," in *Proceedings of the 10th International Conference on Computer Supported Cooperative Work in Design*, 2006.
- [54] MSDN, "The T.120 Standard," [Online]. Available: [http://msdn.microsoft.com/en-us/library/ms709084\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms709084(VS.85).aspx). [Accessed 1 Jun 2013].
- [55] MSDN, "MS-RDPBCGR," [Online]. Available: [http://msdn.microsoft.com/en-us/library/cc240445\(v=prot.10\).aspx](http://msdn.microsoft.com/en-us/library/cc240445(v=prot.10).aspx). [Accessed 1 Jun 2013].
- [56] "Windows Terminal Service," [Online]. Available: [www.microsoft.com/windowsserver2003/technologies/terminalservices/default.mspx](http://www.microsoft.com/windowsserver2003/technologies/terminalservices/default.mspx). [Accessed 1 Jun 2013].
- [57] "Chapter 1: Functional Comparison of UNIX and Windows.," [Online]. Available: <http://technet.microsoft.com/en-us/library/bb496993.aspx>.
- [58] C. Hameed, "Windows 7 / Windows Server 2008 R2: Remote Desktop Services Architecture," 13 October 2009. [Online]. Available: <http://blogs.technet.com/b/askperf/archive/2009/10/13/windows-7-windows-server-2008-r2-remote-desktop-services-architecture.aspx>. [Accessed 10 July 2011].
- [59] "SeamlessRDP," [Online]. Available: <http://www.cendio.com/seamlessrdp/>. [Accessed 1 Jun 2013].
- [60] "rdesktop: A Remote Desktop Protocol Client for accessing Windows Remote Desktop Services," [Online]. Available: <http://www.rdesktop.org/>. [Accessed 1 Jun 2013].

- [61] "rdesktop," [Online]. Available: <http://www.fontis.com.au/rdesktop>. [Accessed 1 Jun 2013].
- [62] "Task Manager overview," [Online]. Available: [http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/taskman\\_whats\\_there\\_w.mspx?mfr=true](http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/taskman_whats_there_w.mspx?mfr=true). [Accessed 1 Jun 2013].
- [63] "Enabling Multiple Remote Desktop Sessions in Windows XP,," [Online]. Available: [fawzi.wordpress.com/2008/02/09/enabling-multiple-remote-desktop-sessions-in-windows-xp/](http://fawzi.wordpress.com/2008/02/09/enabling-multiple-remote-desktop-sessions-in-windows-xp/). [Accessed 1 Jun 2013].
- [64] "How to Enable Multiple Remote Desktop Sessions on XP or Vista," [Online]. Available: [remotedesktoprdp.com/Multiple-Remote-Desktop-Sessions.aspx](http://remotedesktoprdp.com/Multiple-Remote-Desktop-Sessions.aspx). [Accessed 1 Jun 2013].
- [65] "License Terms," [Online]. Available: [www.microsoft.com/About/Legal/EN/US/IntellectualProperty/UseTerms/Default.aspx](http://www.microsoft.com/About/Legal/EN/US/IntellectualProperty/UseTerms/Default.aspx). [Accessed 1 Jun 2013].
- [66] S. Frank and H. Roger, "GPFS: A Shared-Disk File System for Large Computing Clusters," in *Proceedings of the Conference on File and Storage Technologies (FAST'02)*, Monterey, CA, 2002.
- [67] "Filesystem in Userspace," [Online]. Available: <http://fuse.sourceforge.net/>. [Accessed 1 Jun 2013].
- [68] "AVFS: A Virtual Filesystem," [Online]. Available: <http://sourceforge.net/projects/avf/>. [Accessed 1 Jun 2013].
- [69] D. Jeff, "User-Mode Linux," [Online]. Available: [www.kernel.org/doc/ols/2001/uml.pdf](http://www.kernel.org/doc/ols/2001/uml.pdf). [Accessed 1 Jun 2013].
- [70] "Writing a FUSE Filesystem: a Tutorial," [Online]. Available: <http://www.cs.nmsu.edu/~pfeiffer/fuse-tutorial>. [Accessed 1 Jun 2013].
- [71] V. Sivakumar, V. Bharadwaj and R. Thomas G., "Resource-Aware Distributed Scheduling strategies for Large-Scale Computational Cluster/Grid Systems," *IEEE Trans. Parallel Distrib. Syst*, no. DOI=10.1109/TPDS.2007.1073 <http://dx.doi.org/10.1109/TPDS.2007.1073>, pp. 1450-1461, 2007.
- [72] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky and D. Werthimer, "SETI@home: An Experiment in Public-Resource Computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56-61, 2002.
- [73] "distributed.net, "Node Zero,," [Online]. Available: [http://www.distributed.net/..](http://www.distributed.net/) [Accessed 1 Jun 2013].
- [74] A. Hamdy, "Scheduling real-time indivisible loads with special resource allocation requirements on cluster computing," *Journal of Systemics, Cybernetics and Informatics*, vol. 8, no. 5, pp. 34-39, 2010.
- [75] J. Liu, K.-J. Lin, W.-K. Shih, A.-s. Yu, J.-Y. Chung and W. Zhao, "Algorithms for



- scheduling imprecise computations," *Computer*, vol. 24, no. 5, pp. 58-68, 1991.
- [76] H. Tada, M. Imase and M. Murata, "On the robustness of the soft state for task scheduling in large-scale distributed computing environment," in *International Multiconference on Computer Science and Information Technology IMCSIT 2008*, doi: 10.1109/IMCSIT.2008.4747285, 2008.
- [77] K. Saurabh and B. Rajkumar, "Green Cloud Computing and Environmental Sustainability," *Harnessing Green It: Principles and Practices (eds S. Murugesan and G. R. Gangadharan)*, no. UK. doi: 10.1002/9781118305393.ch16, 2012.
- [78] E. Kim, G. Link, K. Yum, V. N., K. M., I. M.J. and D. C.R., "A holistic approach to designing energy-efficient cluster interconnects," *IEEE Transactions on Computers*, vol. 54, no. 6, pp. 660-671, 2005.
- [79] S. Bansal, P. Kumar and K. Singh, "An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 6, pp. 533-544, 2003.
- [80] C. Juan-Carlos, K. Dongkyun and M. Pietro, "Cera: Cluster-based energy saving algorithm to coordinate routing in short-range wireless networks," in *ICOIN*, 2003.
- [81] G. Chen, Z. Cen zhe and T. Teng Tiow, "ShAppliT: A Novel Broker-mediated Solution to Generic Application Sharing in a Cluster of Closed Operating Systems," *International Journal of Soft Computing and Software Engineering [JSCSE]*, vol. 2, no. 6, pp. 16-32, 2012.
- [82] G. Chen, Z. Cen zhe and T. Teng Tiow, "Sharing of Generic Single-user Application without Interference in a Cluster," in *International Conference on Software and Computer Applications, IPCSIT*, Singapore, 2012.
- [83] Z. Cen zhe, G. Chen, W. Jin and T. Teng Tiow, "Towards Scalability Issue in Ontology-based Context-aware Systems," in *International Conference on Software and Computer Applications, IPCSIT*, Singapore, 2012.
- [84] V. Bharadwaj, G. Debasish, M. Venkataraman and R. Thomas G., *Scheduling Divisible Loads in Parallel and Distributed Systems*, Los Almitos, California: IEEE Computer Society Press, 1996.
- [85] "Octave," [Online]. Available: <http://www.gnu.org/software/octave/index.html>. [Accessed 1 Jun 2013].
- [86] K. W. Kurose, *Computer Networking: A Top-down Approach*, Pearson.
- [87] "cplusplus.com," [Online]. Available: <http://wwwcplusplus.com/reference/stl/set/>. [Accessed 1 Jun 2013].
- [88] "'Remote Desktop Protocol Settings in Windows Server 2003 and Windows XP'", Microsoft Support., [Online]. Available: <http://support.microsoft.com/kb/885187>. [Accessed 1 Jun 2013].

- [89] L. Ni, A. Harwood and P. Stuckey, "Realizing the e-science desktop peer using a peer-to-peer distributed virtual machine middleware," in *MCG '06: Proceedings of the 4th International Workshop on*.
- [90] J.-S. Kim, B. Nam, M. Marsh, P. Keleher and B. Bhattacharjee, "Creating a robust desktop grid using peer-to-peer services," in *IPDPS'07: Proceedings of IEEE International Parallel and Distributed Processing Symposium*, 2007.
- [91] Z. Han, X. Liu and X. Li, "A taxonomy of peer-to-peer desktop grid paradigms," *Cluster Computing*, pp. 129-144, 2010.
- [92] S. Choi, M. Baik, J. Gil, C. Park, S. Jung and C. Hwang, "Group based dynamic computational replication mechanism in peer-to-peer grid computing," in *CCGRID 06' the Sixth IEEE International Symposium on Cluster Computing and the Grid*, 2006.

# GLOSSARY

**Remote Desktop Protocol (RDP):** The protocol used to implement remote connections (Terminal Services) on Windows operating systems.

**Protocol data unit (PDU):** Information that is delivered as a unit among peer entities of a network and that may contain control information, address information, or data.

**Remote application:** An application running on a remote server.

**Remote Desktop Protocol (RDP) Client:** The client which initiated the remote desktop connection.

**Remote Desktop Protocol (RDP) Server:** The server to which the client initiated the remote desktop connection.

**Virtual channel:** A communication channel available in a Terminal Services (TS) server session between applications running at the server and applications running on the TS client.

**Static virtual channel:** The Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension is designed to operate over static virtual channels, as specified in [MS-RDPBCGR], using the acronym DRDYVNC. The Remote Desktop Protocol (RDP) layer manages the creation, setup, and data transmission over the virtual channel.

**Multipoint Communication Service (MCS):** A data transmission protocol and set of services defined by the ITU T.120 standard, specifically [T122] and [T125].

**User session:** An abstract venue on a server that is assigned to a user. The user interacts with the server and applications from within this venue.

**Remoting:** A server sending graphical data or application data from a server-based application to a remote client.

**Hosting:** The assignment, management, and operation of a user-dedicated session on a server for a user accessing the server, for example, when a user runs an application on a server, the application is running within a user session that the server is hosting.

**Terminal server:** A computer on which Terminal Services is running.

**Terminal Services:** A service on a server computer that allows delivery of applications, or the desktop itself, to various computing devices. When a user runs an application on a terminal server, the application execution takes place on the server computer and only keyboard, mouse, and display information is transmitted over the network. Each user sees only his or her individual session, which is managed transparently by the server operating system and is independent of any other client session.

**Network Level Authentication (NLA):** Refers to the usage of CredSSP [MS-CSSP] within the context of an RDP connection to authenticate the identity of a user at the network layer before the initiation of the RDP handshake. The usage of NLA ensures that server resources are only committed to authenticated users.

**Server Authentication:** The act of proving the identity of a server to a client while providing key material that binds the identity to subsequent communications.

**Firewall:** A firewall is a software component typically implemented on an Internet gateway device that is a part of a private network. The firewall is configured to either block or allow external access to resources within the private network.

**Client Data Block:** A collection of related client settings that are encapsulated within the user data of a Generic Conference Control (GCC) Conference Create Request. Only four Client Data Blocks exist: Core Data, Security Data, Network Data, and Cluster Data. The set of Client Data Blocks is designed to remain static.

**Server Data Block:** A collection of related server settings that are encapsulated within the user data of a Generic Conference Control (GCC) Conference Create Response. Three Server Data Blocks exist: Core Data, Security Data, and Network Data.

# APPENDICES

## A. RDP Connection Sequence and PDU

### a. RDP Connection Sequence

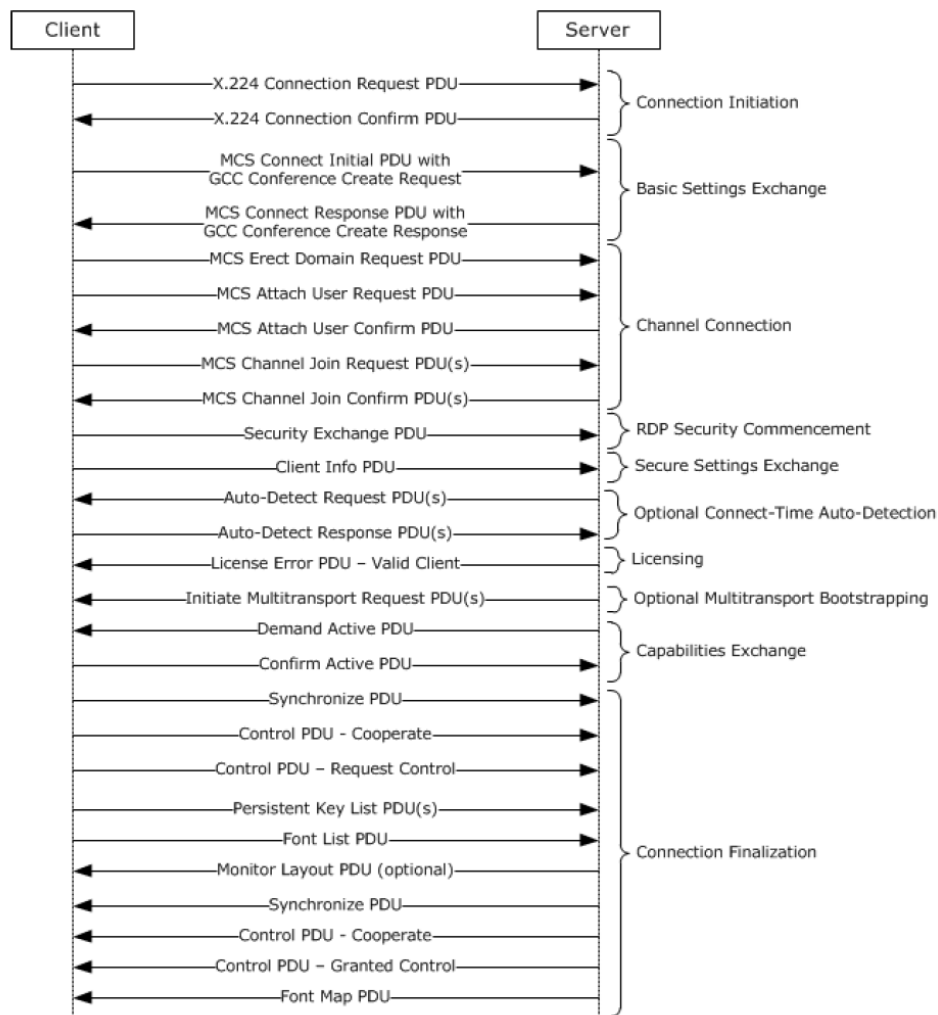


Figure 55 Connection sequence of RDP [55]

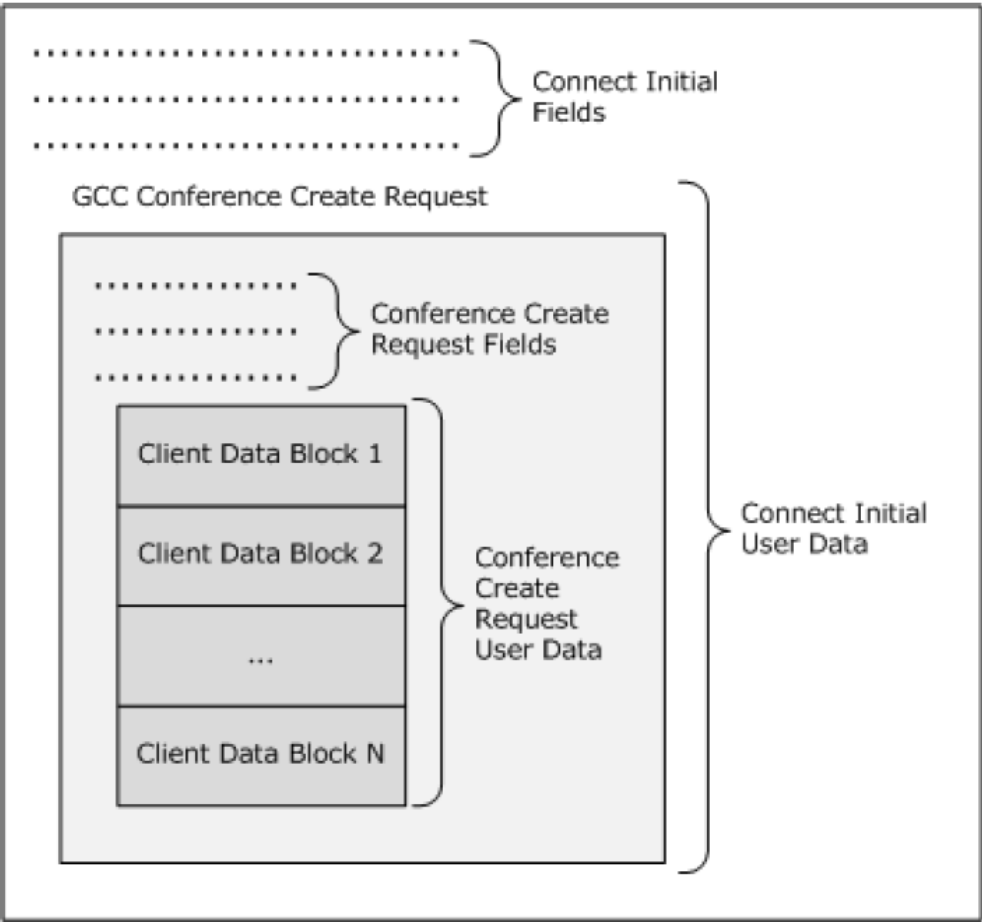


Figure 56 MCS connect initial PDU [55]

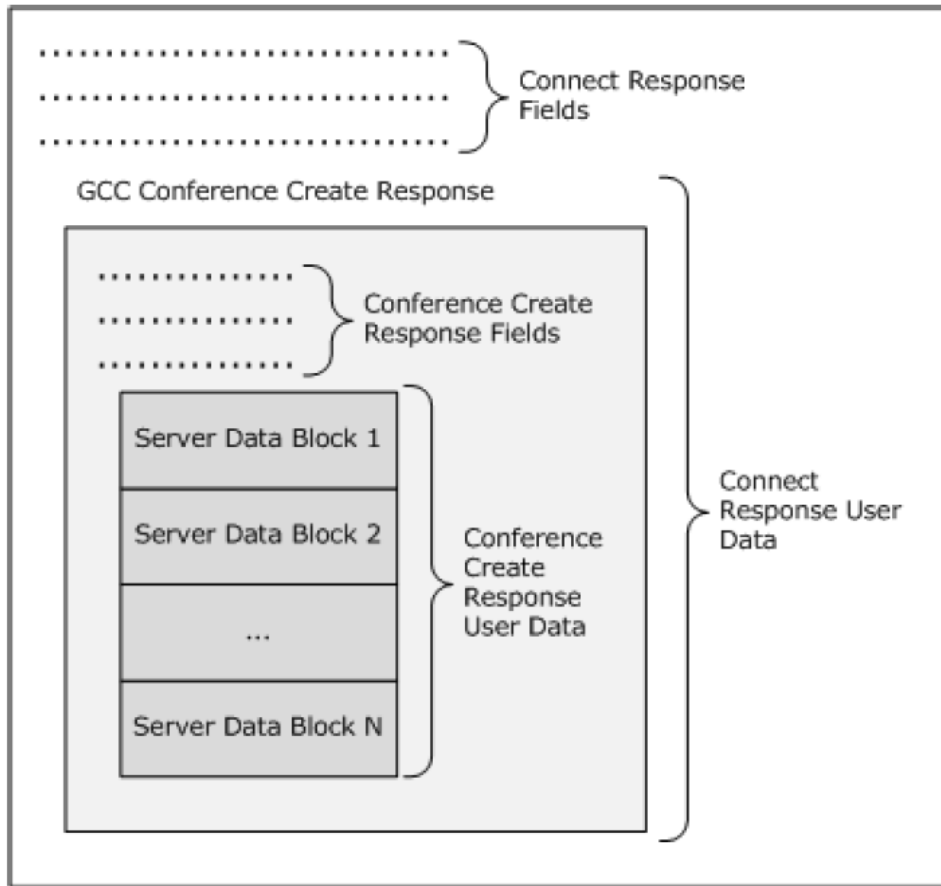


Figure 57 MCS connect response PDU [55]

*b. Protocol Data Unit (PDU)*

Protocol Data Unit, PDU is information delivered through network layers. Connection Initiation: After the client initiates the connection by sending the server a Class 0 X.224 Connection Request PDU and the server responds with a Class 0 X.224 Connection Confirm PDU. From this point, all subsequent data sent between client and server is wrapped in an X.224 Data Protocol Data Unit (PDU). [52]

**Client X.224 Connection Request PDU Example**

```
00000000 03 00 00 2c 27 e0 00 00 00 00 43 6f 6f 6b 69 ..., '.....Cooki
00000010 65 3a 20 6d 73 74 73 68 61 73 68 3d 65 6c 74 6f e mstshash=a
00000020 6e 73 0d 0a 01 00 08 00 00 00 00 00 ns.....
03 -> TPKT Header: version = 3
```

```

00 -> TPKT Header: Reserved = 0
00 -> TPKT Header: Packet length - high part
2c -> TPKT Header: Packet length - low part (total = 44 bytes)
27 -> X.224: Length indicator (39 bytes)
e0 -> X.224: Type (high nibble) = 0xe = CR TPDU; credit (low nibble) = 0
00 00 -> X.224: Destination reference = 0
00 00 -> X.224: Source reference = 0
00 -> X.224: Class and options = 0
43 6f 6f 6b 69 65 3a 20 6d 73 74 73 68 61 73 68
3d 65 6c 74 6f 6e 73 -> "Cookie: mstshash=a"
0d0a -> Cookie terminator sequence
01 -> RDP_NEG_REQ::type (TYPE_RDP_NEG_REQ)
00 -> RDP_NEG_REQ::flags (0)
08 00 -> RDP_NEG_REQ::length (8 bytes)
00 00 00 00 -> RDP_NEG_REQ: Requested protocols (PROTOCOL_RDP)

```

### Server X.224 Connection Confirm PDU Example

```

00000000 03 00 00 13 0e d0 00 00 12 34 00 02 00 08 00 01 .....4.....
00000010 00 00 00 ...
03 -> TPKT Header: TPKT version = 3
00 -> TPKT Header: Reserved = 0
00 -> TPKT Header: Packet length - high part
13 -> TPKT Header: Packet length - low part (total = 19 bytes)
0e -> X.224: Length indicator (14 bytes)
d0 -> X.224: Type (high nibble) = 0xd = CC TPDU; credit (low nibble) = 0
00 00 -> X.224: Destination reference = 0
12 34 -> X.224: Source reference = 0x1234 (bogus value)
00 -> X.224: Class and options = 0
02 -> RDP_NEG_RSP::type (TYPE_RDP_NEG_RSP)
00 -> RDP_NEG_RSP::flags (0)
08 00 -> RDP_NEG_RSP::length (8 bytes)
00 00 00 00 -> RDP_NEG_RSP: Selected protocols (PROTOCOL_RDP)

```

### *c. Protocol Packet Analysis for Initializing the Connection*



No.	Time	Source	Destination	Protocol	Length	Info
1371	24.226157	172.19.78.64	172.19.72.228	TCP	54	2293->3389 [ACK] Seq=13439 Ack=89538 win=64978 Len=0
1372	24.230696	00:64:40:3a:52:80	Broadcast	ARP	60	who has 172.19.79.175? Tell 172.19.64.1
1373	24.236416	172.19.72.228	172.19.78.64	TPKT	105	continuation
1374	24.240734	172.19.67.52	255.255.255.255	DB-LSP-	150	Dropbox LAN sync Discovery Protocol
1375	24.241399	172.19.67.52	172.19.79.255	DB-LSP-	150	Dropbox LAN sync Discovery Protocol
1376	24.247106	172.19.78.64	172.19.72.228	RDP	107	
1377	24.255038	172.19.78.64	172.19.72.228	RDP	107	
1378	24.255183	172.19.72.228	172.19.78.64	TCP	60	3389->2293 [ACK] Seq=89589 Ack=13545 win=65535 Len=0
1379	24.263032	172.19.78.64	172.19.72.228	RDP	107	
1380	24.271039	172.19.78.64	172.19.72.228	RDP	107	
1381	24.271153	172.19.72.228	172.19.78.64	TCP	60	3389->2293 [ACK] Seq=89589 Ack=13651 win=65429 Len=0
1382	24.271319	00:64:40:3a:52:80	Broadcast	ARP	60	who has 172.19.76.94? Tell 172.19.64.1
1383	24.279012	172.19.78.64	172.19.72.228	RDP	107	
1384	24.286210	172.19.75.145	172.19.79.255	NBNS	92	Name query NB EEA0508<20>
1385	24.287074	172.19.78.64	172.19.72.228	RDP	107	
1386	24.287189	172.19.72.228	172.19.78.64	TCP	60	3389->2293 [ACK] Seq=89589 Ack=13757 win=65323 Len=0
1387	24.295019	172.19.78.64	172.19.72.228	RDP	107	
1388	24.303036	172.19.78.64	172.19.72.228	RDP	107	
1389	24.303170	172.19.72.228	172.19.78.64	TCP	60	3389->2293 [ACK] Seq=89589 Ack=13863 win=65217 Len=0
1390	24.311023	172.19.78.64	172.19.72.228	RDP	107	
1391	24.319046	172.19.78.64	172.19.72.228	RDP	107	
1392	24.319149	172.19.72.228	172.19.78.64	TCP	60	3389->2293 [ACK] Seq=89589 Ack=13969 win=65111 Len=0
1393	24.326154	172.19.75.173	172.19.79.255	NBNS	92	Name query NB SIP.EXAMPLE.COM<00>
1394	24.327101	172.19.78.64	172.19.72.228	RDP	107	
1395	24.335043	172.19.78.64	172.19.72.228	RDP	107	
1396	24.335147	172.19.72.228	172.19.78.64	TCP	60	3389->2293 [ACK] Seq=89589 Ack=14075 win=65005 Len=0
1397	24.343006	172.19.78.64	172.19.72.228	RDP	107	
1398	24.351045	172.19.78.64	172.19.72.228	RDP	107	
1399	24.351149	172.19.72.228	172.19.78.64	TCP	60	3389->2293 [ACK] Seq=89589 Ack=14181 win=64899 Len=0

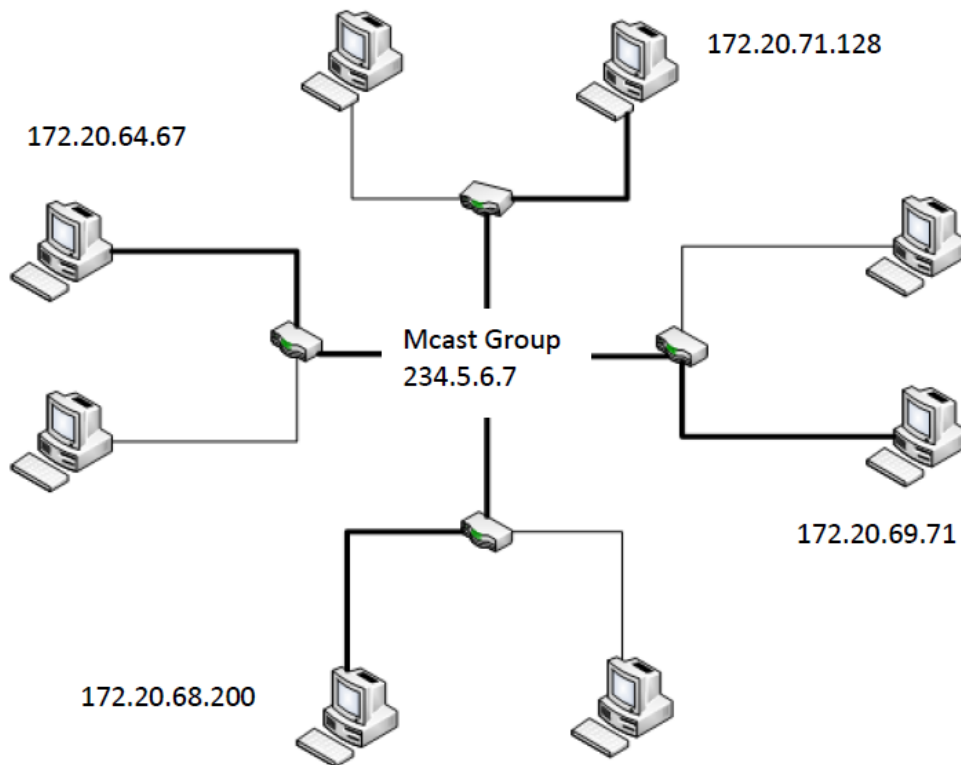
  

Frame 1376: 107 bytes on wire (856 bits), 107 bytes captured (856 bits)						
Ethernet II, Src: 00:1a:a0:70:89:91 (00:1a:a0:70:89:91), Dst: 00:1a:a0:70:86:57 (00:1a:a0:70:86:57)						
Destination: 00:1a:a0:70:86:57 (00:1a:a0:70:86:57)						
Address: 00:1a:a0:70:86:57 (00:1a:a0:70:86:57)						
.....0. .... = LG bit: Globally unique address (factory default)						
.....0 ..... = IG bit: Individual address (unicast)						
Source: 00:1a:a0:70:89:91 (00:1a:a0:70:89:91)						
Address: 00:1a:a0:70:89:91 (00:1a:a0:70:89:91)						
..... = LG bit: Globally unique address (factory default)						
0000 00 1a a0 70 86 57 00 1a a0 70 89 91 08 00 45 00 ...p.W...p...E.						
0010 00 5d 39 51 40 00 80 06 d1 fe ac 13 4e 40 ac 13 ...j9Q8...NB..						
0020 48 e4 08 f5 0d 3d 3d 13 b2 86 e3 81 56 50 18 H...#8...VP.						
0030 fd 9f ef 9a 00 00 03 00 00 35 02 f0 80 64 00 05 .....\$.d...						
0040 03 eb 70 80 26 00 00 00 00 22 00 17 00 ee 03 ea ..p.&...".						
0050 03 01 00 00 14 00 1c 00 00 00 01 00 00 00 97 .....M.....						
0060 2a cd 4d 01 80 00 08 7f 02 bf 01						

## B. Cluster Management

In our first attempt to create a peer-to-peer application sharing cluster, Microsoft Peer Name Resolution Protocol (PNRP) is implemented as our base protocol [86]. However, the result is not satisfactory because of an excessive delay in the connection. Therefore, a new system using multicast approach is implemented. Multicast packet is addressed using a single identifier for a group of receivers. This address indirection allows a copy of the packet that is addressed to the group to be delivered to all the multicast receivers associated with that group.

Class network as used by multicast is succeeded by classless inter-domain routing. However, multicasting address is still considered as Class D address. Classless inter-domain routing used significant bits to represent host and network. For an example, 192.168.0.0/16 means that there are  $2^{(32-16)}$  host in the network and they start from 192.168.0.0 to 192.168.255.255. The figure shows a class D identifier, 234.5.6.7, which is used to associate a group of receivers. This group is referred as a multicast group. The flow chart describes the implementation of multicast clustering using Win32 APIs.



*Figure 58 Multicast group*

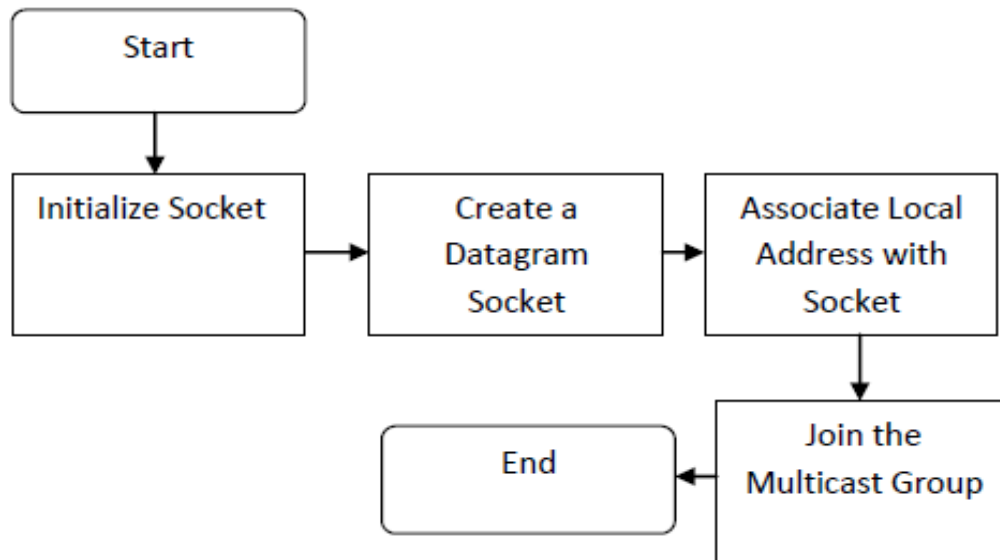


Figure 59 Flow chart of joining a multicast group

### C. Incoming and Outgoing Packet Management

The message passed within the cluster determines the sender, the message type and the application requested. In the example below, suppose Alice request WinWord from the multicast group. Bob replies to Alice's request. Charlie discards Alice's request because request has been fulfilled by Bob. The definition of four message header types:

1. Type 0: request an application by Alice, example: 0//winword.exe
2. Type 1: reply a particular request, example:  
1//Alice\_IP//winword.exe//C:\\Program Files\\Microsoft Office\\winword.exe//guest2
3. Type 2: handshake between all hosts to notify each other their existence in the cluster
4. Type 3: graceful disconnection if a host is to leave the cluster

When ShAppliT receive datagram from the network, these packets are stored in the list. There are 3 kind of list:

1. A list of all requests by the host
2. A list of all incoming replies to the request of the host
3. A list of all incoming requests from other clients

The information in the lists must be unique. This uniqueness can be enforced by using STL (Standard Template Library) set [87]. Sets are associate containers that store unique elements or keys. The uniqueness of the structure is enforced by the operator of the structure. A thread is used to process incoming datagram stored in the set.

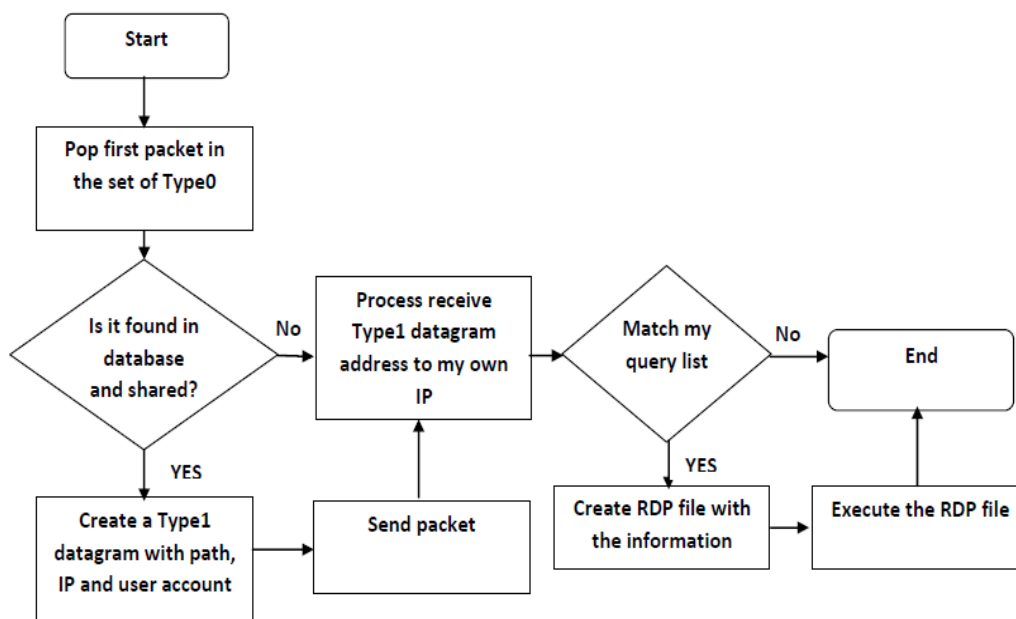


Figure 60 Flow chart of processing datagram

QueryPacket structure: there is no duplicate application name. Example: Alice cannot request winword.exe twice until the previous request is timed out or is satisfied by other peer in the cluster.

RecReplyPac structure: this structure is used to store replies of all the requests made. The uniqueness of the structure is enforced by “a peer does not satisfy any

request twice”. Example: Alice will not store the reply packet on winword.exe from Bob twice.

KeyValueRec structure: this structure is used to process incoming request packets. It stores a temporary list for later processing. This structure’s uniqueness is enforced by “the same IP should not associate to the same application”. Example: Bob receives Alice request on winword.exe and powerpnt.exe, Bob should not receive Alice’s request on winword.exe twice. Figure 61 C++ codes of message structures used to store the receiving packet from the cluster shows C++ codes for message structures used to store the receiving packets from the cluster

```

typedef struct _RecReplyPac{
wstring strAppName;
wstring strIpv4;
wstring strFullPathName;
wstring strUsername;
bool operator<(const _RecReplyPac& A) const
{
return (strIpv4.compare(A.strIpv4) < 0 &&
strAppName.compare(A.strAppName) < 0 );
}
}RecReplyPac;
typedef struct _QueryPacket{
SYSTEMTIME systemTime;
wstring strAppName;
bool operator<(const _QueryPacket& A) const
{
return (strAppName.compare(A.strAppName) < 0 );
}
}QueryPacket;
typedef struct _KeyValueRec{
wstring strIpv4;
wstring strAppName;
bool operator<(const _KeyValueRec& A) const
{
return (strIpv4.compare(A.strIpv4) <0) ^
(strAppName.compare(A.strAppName) < 0);
}
}KeyValueRec;

```

*Figure 61 C++ codes of message structures used to store the receiving packet from the cluster*

## D. Demonstrations

### a. Rdesktop as the Client program

Run from Xwin server (Xwin allows you to run linux sessions inside windows)

```
./rdesktop -s "notepad" hostIP
```

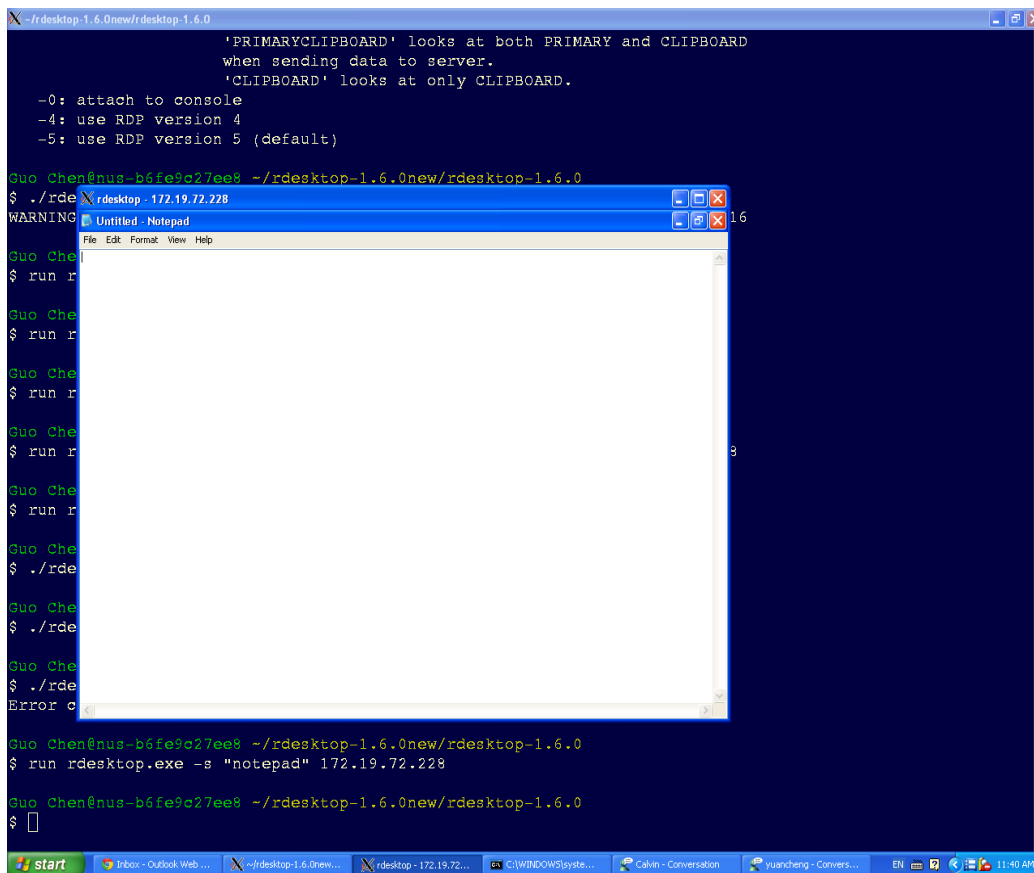


Figure 62 Run Linux sessions inside Windows

## b. Compile Rdesktop for Windows

```
tar -xf /rdesktop-1.6.0.tar.gz

cd rdesktop-1.6.0/

./configure --with-x --with-sound=oss; make; strip rdesktop.exe

mkdir /Rdesktop-1.6.0-Win32

ldd rdesktop.exe | perl -ane 'print "cp \"\$F[2]\" \"/Rdesktop-1.6.0-Win32/\"\\n" if
!/cygdrive/i;' | sh

cp rdesktop.exe /Rdesktop-1.6.0-Win32

cp -r keymaps /Rdesktop-1.6.0-Win32

zip -9rq /Rdesktop-1.6.0-Win32.zip /Rdesktop-1.6.0-Win32/*
```

If you should see the following error message:

```
ERROR: Failed to open display:
```

Set the needed variable with this command:

```
set DISPLAY=127.0.0.1:0
```



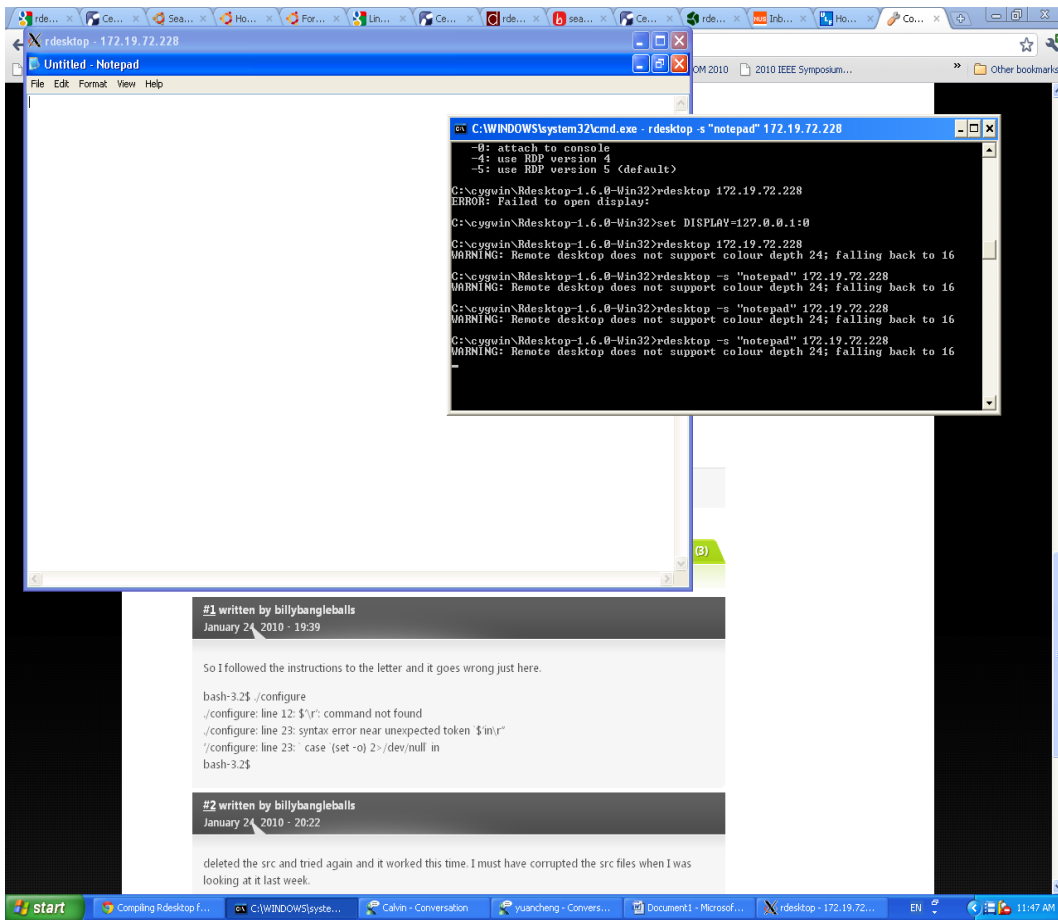
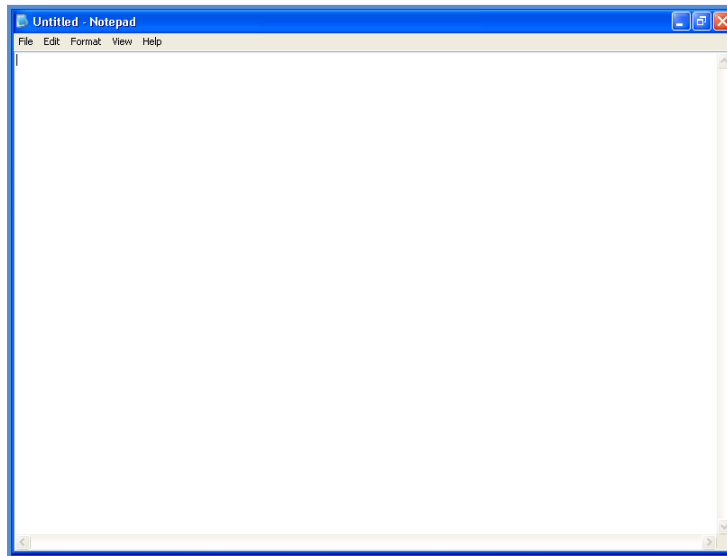


Figure 63 Compile rdesktop for Windows

### *c. SeamlessRDP and accessing remote applications*

One of the features of ShAppliT is to use an application in seamless mode called SeamlessApp [59]. That means the application itself looks as if it's been started from the local machine when it comes to the look and feel. Running seamless applications is the least confusing way for an end user to experience an application over an App Share session, as he/she sees no difference between the remote application and his/her local application. The default way of deploying a ShAppliT application has been set to seamless [59]. The technology behind the seamless application basically cloaks or clips out the part of the window that shows the application in a normal Windows shell.

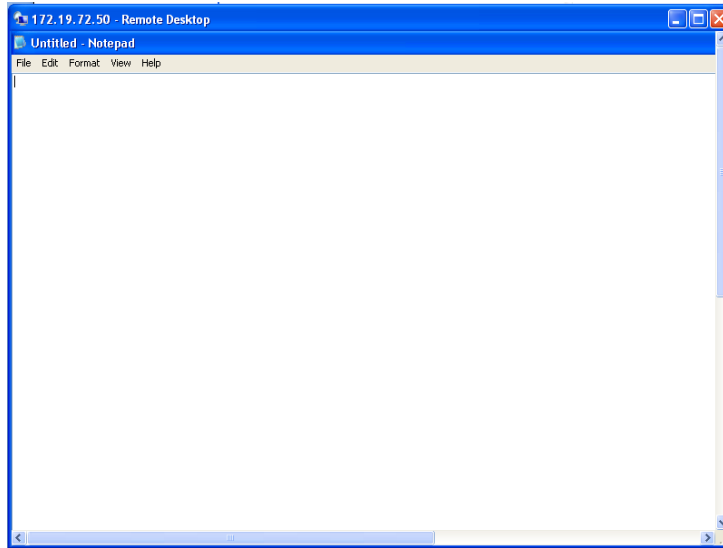
For example, when opening the notepad on your local Windows XP machine, it will look like this in **Error! Reference source not found.:**



*Figure 64 Screen shot of notepad on local machine*

However, if a user remote accesses the application using remote desktop connection normally user can see the window frame like the min/max/close button section, the title bar, etc. The parts which are not supposed to be visible are made invisible using the clipping technology. ShAppliT uses this technology for

all applications to give them a seamless look as in Figure 64 Screen shot of notepad on local machine.



*Figure 65 Screen shot of notepad on remote desktop connection*

With the release of rdesktop 1.5.0, a feature known as seamless RDP was contributed by Cendio [35] allowing rdesktop to run individual applications rather than a full desktop. Fontis [61] has been working on a number of patches to the seamless RDP feature, adding support for rdesktop session connection-sharing, icon support, improved handling of always-on-top windows and more.

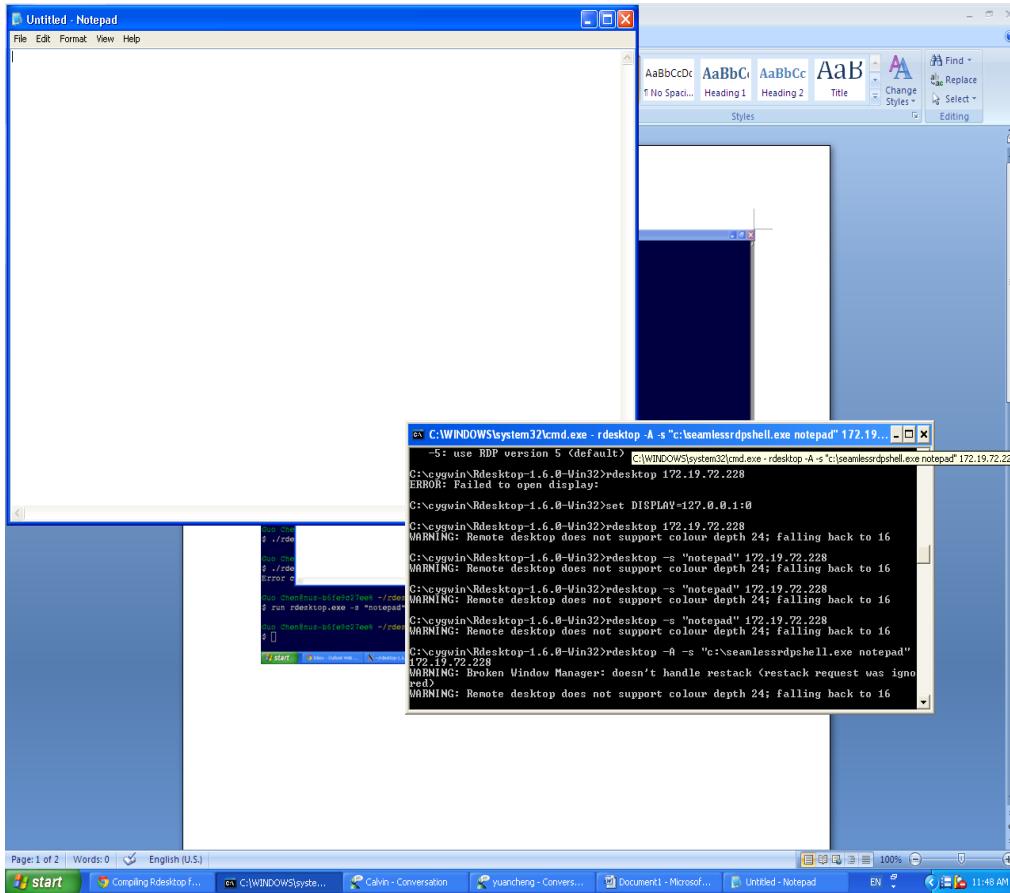


Figure 66 Screenshot of seamless application

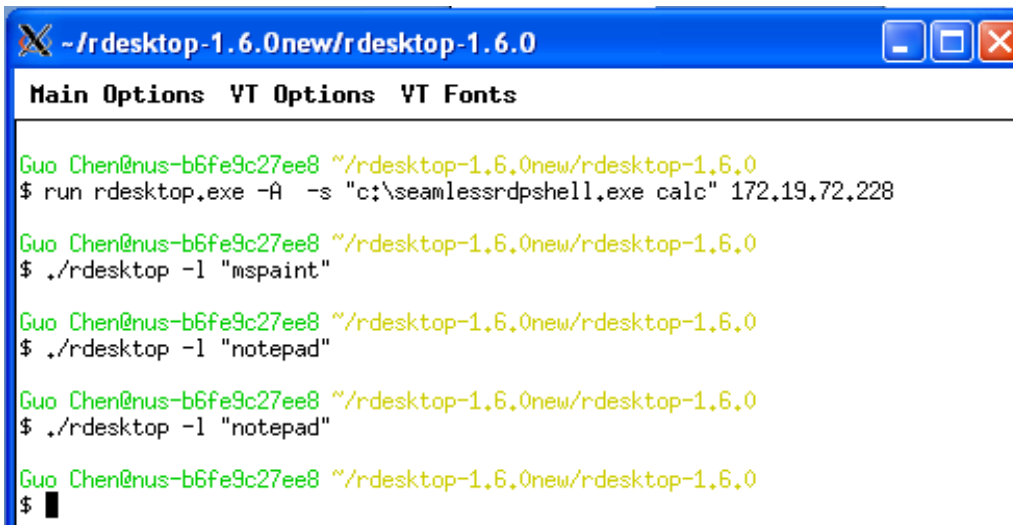


Figure 67 Command of seamless remote applications

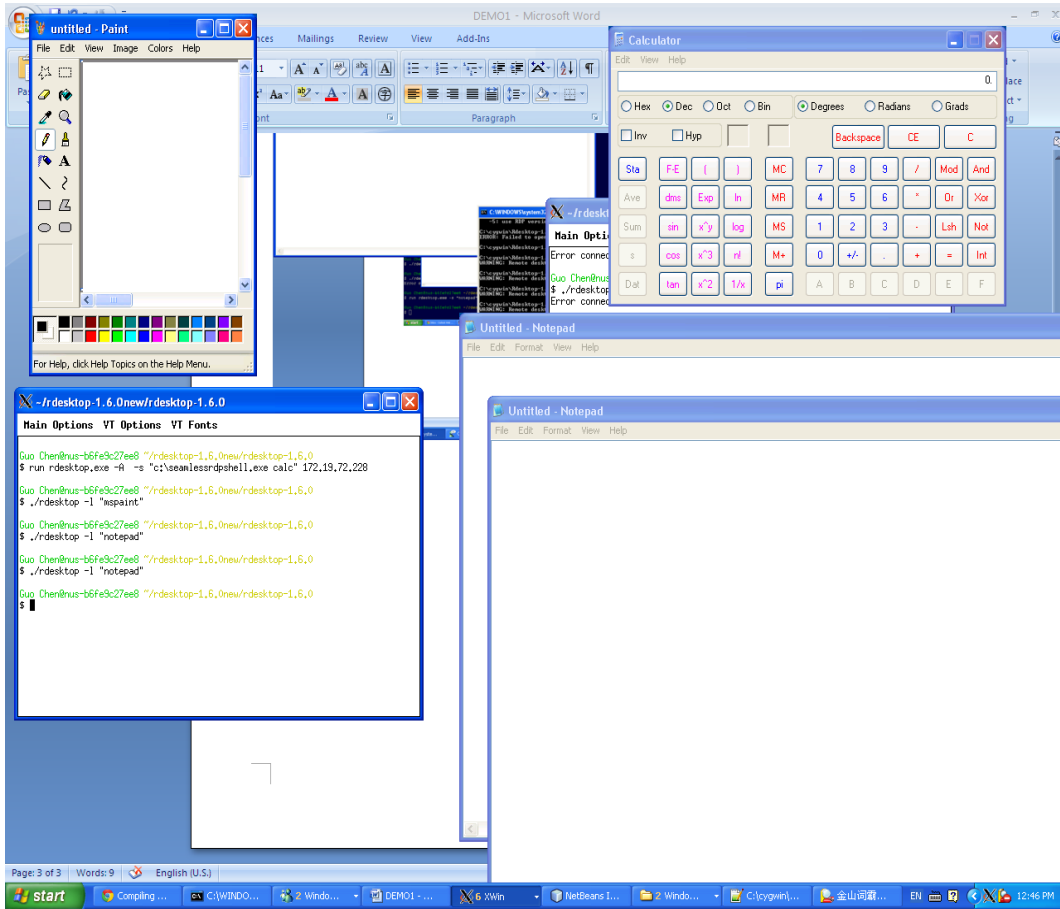


Figure 68 Screenshot of opening more remote applications

```
18 audiomode:i:0
19 redirectprinters:i:1
20 redirectcomports:i:0
21 redirectsmartcards:i:1
22 redirectclipboard:i:1
23 redirectposdevices:i:0
24 drivestoredirect:s:
25 autoreconnection enabled:i:1
26 authentication level:i:2
27 prompt for credentials:i:0
28 negotiate security layer:i:1
29 remoteapplicationmode:i:1
30 remoteapplicationprogram:s:Notepad
31 disableremoteappcapscheck:i:1
32 alternate shell:s:rdpinit.exe
33 alternate shell:s:c:\windows\system32\notepad.exe
34 shell working directory:s:
35 gatewayhostname:s:
36 gatewayusagemethod:i:4
```

*Figure 69 Editing an RDP file*

Run `rdesktop.exe -A -s "c:\seamlessrdpshell.exe explorer" 172.19.72.228`

Use a `V_channel` for seamlessRDP; if “-A” seamlessRDP enabled, `seamless_create_socket(master_socket)`; If “-l” slave mode, send command line “mspaint” to the master process (send spawn command to server-side seamlessRDP) then exit.

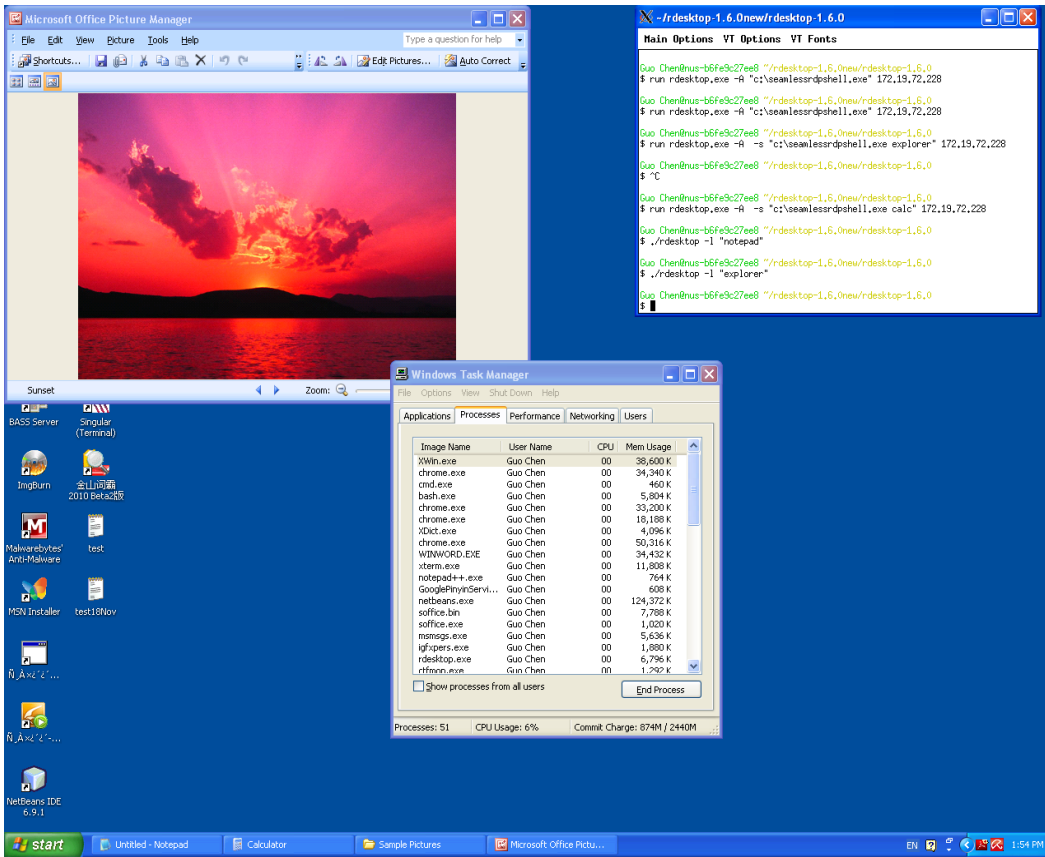
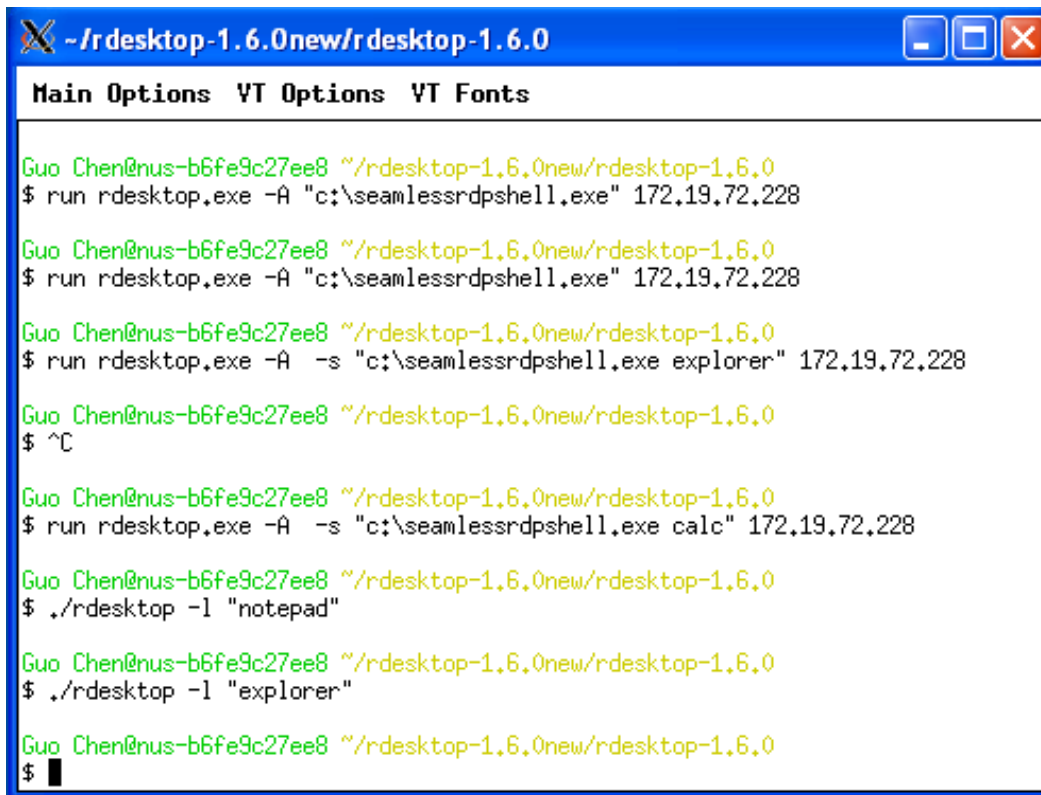


Figure 70 Remote accessing explorer.exe

The image shows a Windows command prompt window titled "~ /rdesktop-1.6.0new/rdesktop-1.6.0". The window has a blue title bar with standard minimize, maximize, and close buttons. Below the title bar, there are three tabs: "Main Options", "VT Options", and "VT Fonts". The main area of the window contains a series of command-line interactions. Each line starts with a green prompt "Guo Chen@nus-b6fe9c27ee8 ~/rdesktop-1.6.0new/rdesktop-1.6.0" followed by a white prompt "\$". The commands and their outputs are: 1. "run rdesktop.exe -A "c:\seamlessrdpshell.exe" 172.19.72.228" 2. "run rdesktop.exe -A "c:\seamlessrdpshell.exe" 172.19.72.228" 3. "run rdesktop.exe -A -s "c:\seamlessrdpshell.exe explorer" 172.19.72.228" 4. "^C" 5. "run rdesktop.exe -A -s "c:\seamlessrdpshell.exe calc" 172.19.72.228" 6. "./rdesktop -l "notepad"" 7. "./rdesktop -l "explorer"" 8. A blank line with a cursor at the end of the prompt "\$".

*Figure 71 Local (client) command window*

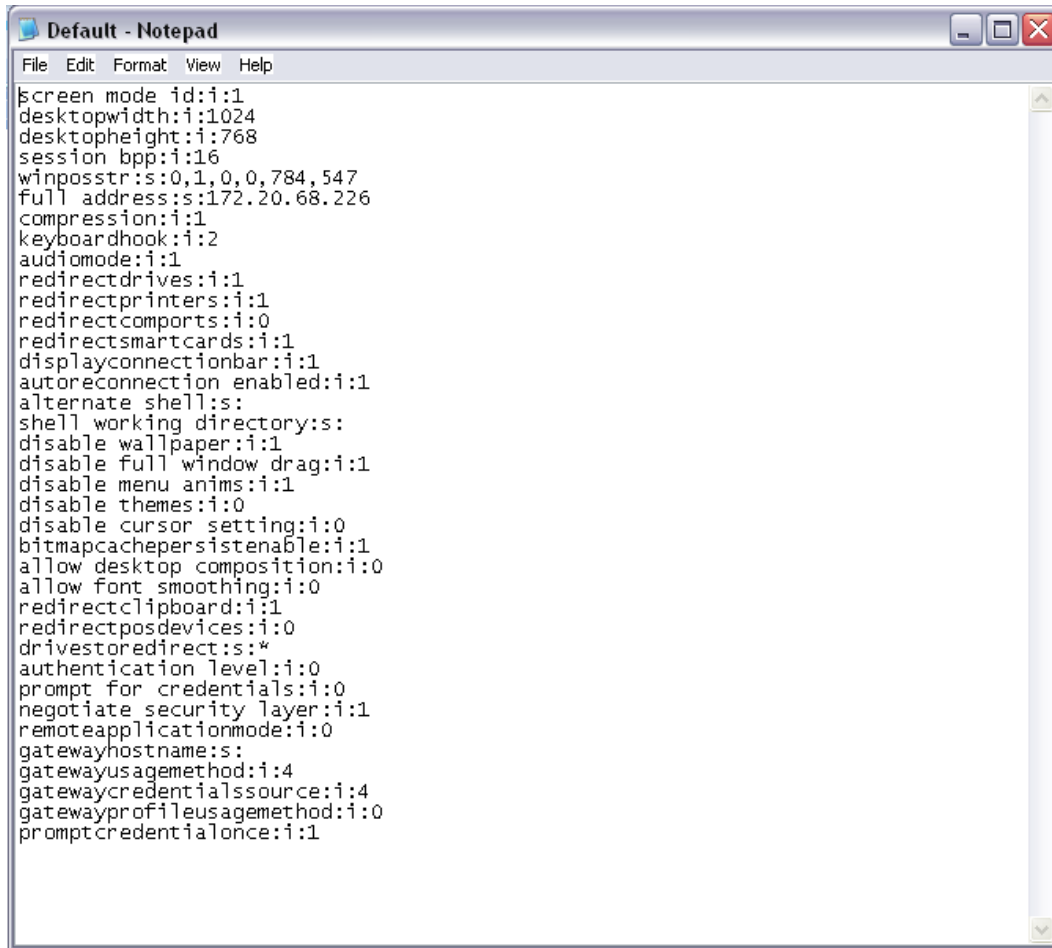
## **E. Customization of a Remote Application Session Using RDP File**

RDP version 5.0 and above deployed in the Windows XP operating system offer a lot more capabilities than a normal remote desktop session. For instance, the remote desktop client allows the user to define the display settings for the remote desktop sessions. This allows a better control over the user experience versus the performance of a remote session.

Besides that, the remote desktop connection also allows the initiation of a program when a session starts. The remote desktop experience could also be optimized by enabling/disabling of advance features. In order to increase the versatility of remote desktop connection with the properties mentioned above, a



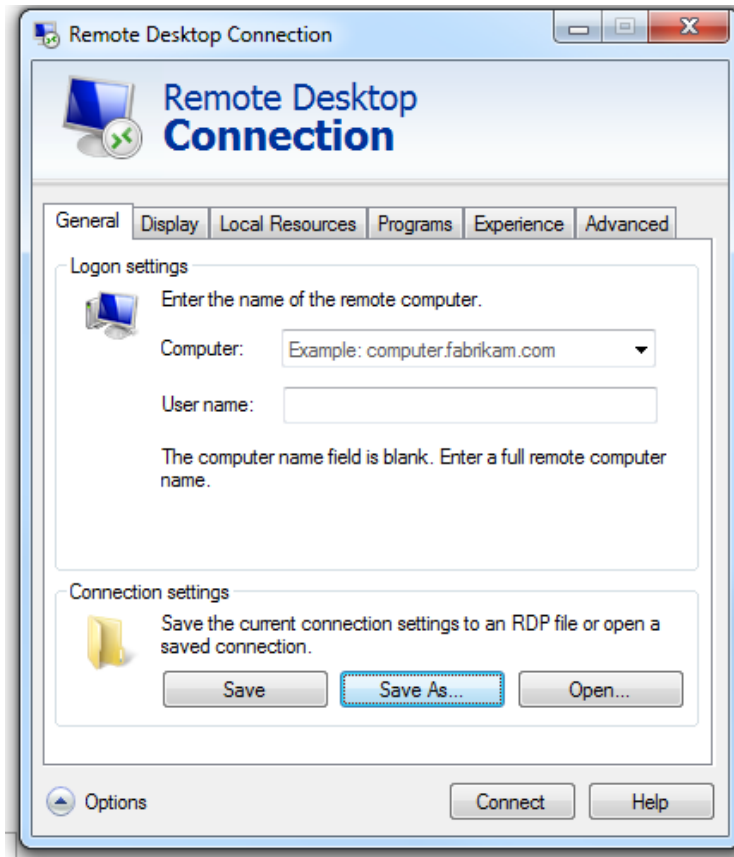
file type with the extension “.rdp” is being created in Windows XP. The RDP files contained parameters that control all the properties mentioned above and they can be modified using a text editor as shown in Figure 72 A RDP file being edited by notepad.

A screenshot of a Notepad window titled "Default - Notepad". The window contains a list of RDP configuration parameters. The parameters are: screen mode id:i:1, desktopwidth:i:1024, desktopheight:i:768, session bpp:i:16, winposstr:s:0,1,0,0,784,547, full address:s:172.20.68.226, compression:i:1, keyboardhook:i:2, audiomode:i:1, redirectdrives:i:1, redirectprinters:i:1, redirectcomports:i:0, redirectsmartcards:i:1, displayconnectionbar:i:1, autoreconnection enabled:i:1, alternate shell:s:, shell working directory:s:, disable wallpaper:i:1, disable full window drag:i:1, disable menu anims:i:1, disable themes:i:0, disable cursor setting:i:0, bitmapcachepersistenable:i:1, allow desktop composition:i:0, allow font smoothing:i:0, redirectclipboard:i:1, redirectposdevices:i:0, drivestoredirect:s:\*, authentication level:i:0, prompt for credentials:i:0, negotiate security layer:i:1, remoteapplicationmode:i:0, gatewayhostname:s:, gatewayusagemethod:i:4, gatewaycredentialssource:i:4, gatewayprofileusagemethod:i:0, and promptcredentialonce:i:1.

```
File Edit Format View Help
|screen mode id:i:1
desktopwidth:i:1024
desktopheight:i:768
session bpp:i:16
winposstr:s:0,1,0,0,784,547
full address:s:172.20.68.226
compression:i:1
keyboardhook:i:2
audiomode:i:1
redirectdrives:i:1
redirectprinters:i:1
redirectcomports:i:0
redirectsmartcards:i:1
displayconnectionbar:i:1
autoreconnection enabled:i:1
alternate shell:s:
shell working directory:s:
disable wallpaper:i:1
disable full window drag:i:1
disable menu anims:i:1
disable themes:i:0
disable cursor setting:i:0
bitmapcachepersistenable:i:1
allow desktop composition:i:0
allow font smoothing:i:0
redirectclipboard:i:1
redirectposdevices:i:0
drivestoredirect:s:*
authentication level:i:0
prompt for credentials:i:0
negotiate security layer:i:1
remoteapplicationmode:i:0
gatewayhostname:s:
gatewayusagemethod:i:4
gatewaycredentialssource:i:4
gatewayprofileusagemethod:i:0
promptcredentialonce:i:1
```

*Figure 72 A RDP file being edited by notepad*

Alternatively, the RDP file works similarly as a shortcut button whereby a double click on an RDP file will launch the remote desktop connection and subsequently the remote desktop session. As a result, the customization of a remote desktop session is made easy as all the configuration can be saved in an RDP file which the RDP client could read from. [88]



*Figure 73 Windows remote desktop connection*

## **F. Integrity Test for PFS File System**

This integrity test was intentionally done on the PFS under vigorous operating condition so as to unveil the reliability of the PFS in a certain way. The test script (Figure 74 Integrity test script) was run and this involved 10,000 file creations with varying sizes on the PFS.

```

Test script 2

// This is to create 10K files with variance size for the integrity test.

#!/bin/bash

RANGE=50240000 #maximum size of files to create

for i in {1..100}
do
    number=$RANDOM*10000          #RANDOM      NUMBER
    GENEERATOR not big enough
    number+=$RANDOM
    let "number %= $RANGE"

    CMD="dd  if=/dev/urandom  of=./test/testfile$number  bs=$number
count=1"

    $CMD

done

```

*Figure 74 Integrity test script*

Upon completion of the file creations and the mirroring, the Linux command:

*diff -r -N </Path on Client> <Path on Server>*

was issued to detect any discrepancies between the files in the client and server. Main function is shown in Figure 75 Main function to detect any discrepancies between the files in the client and server. However, no discrepancies were found, and all 10,000 files written on the client were mirrored on the server. This demonstrates that PFS is indeed reliable as it ensures file operations performed on the client are being mirrored accurately onto the server.

```

//Main function
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <ctime>
#include <sys/stat.h>
using namespace std;
int main (int argc, char * argv[]) { //argv[1]=# of kb to write. argv[2]=# of times to
run simulation. argv[3]=file path + name
char Data[1025];
clock_t t1, t2;
t1 = clock();
for(int y=0;y<atoi(argv[2]);y++){
    for(int x=0;x<1024;x++)Data[x]='A'; //Each write is 1kb
    Data[1024]='\0'; //Null Byte the string
    ofstream myfile;
    myfile.open (argv[3]);
    for(int x=0;x<atoi(argv[1]);x++) myfile<< Data;
    myfile.close();
} //End for
t2 = clock();
float diff = ((float)t2 - (float)t1) / (float)atoi(argv[2]); //10000.0F;
cout <<endl<< "Time Taken for " <<argv[1] << "kb=" << diff << "
ms"<<endl<<endl;
    return 0;
} //End Main

```

*Figure 75 Main function to detect any discrepancies between the files in the client and server*

## **G. Latency Test for PFS File System**

A test script (Figure 76 Latency test script) was written to measure the latencies experienced during a read operation under the above two mentioned conditions, and the latencies was benchmarked against the default file system in Linux. For this latency test, files of various sizes ranging from 1KB to 50MB were being

read by the client computer, and this was followed with the writing of these files as well. The graphs in Figure 37 and Figure 38 display the results for both the read and write latency tests.

```
Test script 1

// This is the test script for the latency test.
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <ctime>
#include <sys/stat.h>
using namespace std;
//Run it as such: ./a.out <# of simulation> <file1> <file2> .....
int main(int argc, char * argv[]){ //argv[1]=# of times to run simulation.
    argv[2] onwards = files to work on
    struct stat filestatus;
    clock_t t1, t2;
    for(int y=2;y<argc;y++){
        stat( argv[ y ], &filestatus);
        t1 = clock();
        for(int x=0;x<atoi(argv[1]);x++){
            FILE *file = fopen ( argv[y], "r" );
            char line [102400];//[Default][50000x],1024,
            while ( fgets(line, sizeof(line), file ) != NULL){
                //fputs ( line, stdout );
            }
            fclose ( file );
        }
        t2 = clock();
        float diff = ((float)t2 - (float)t1) / (float)atoi(argv[1]);//10000.0F;
        cout<<endl << "Timetaken for " << filestatus.st_size <<" bytes
        ="<<diff<<"ms"<<endl<<endl;
    }
    return 0;
}
```

*Figure 76 Latency test script*

## H. ICSCluster (Imprecise Computation Scheduling Cluster)

### Simulation

a) HOW TO run this simulation:

If octave is properly installed, simply run the shell file 'run' under terminal. Otherwise, 'run.m' is prepared for MATLAB use. However, this code is not tested under MATLAB, so there is no guarantee it can run in MATLAB.

b) File Description:

<b>gentask.m</b>	Randomly generate task set given a work load.
<b>simics_sched.m</b> (mentioned as ICS_sched in Chapter 5)	Carry out the imprecise computation scheduling job given task set and processors for SLTN (single level tree network) and returns solution ( <b>sol</b> ).
<b>simics.m</b>	Workbench for simulation.
<b>toptest.m</b>	Evaluate the performance of each algorithm

```
%===== gentask.m=====
```

```
function t=gentask(totalu)
ntask=3;
maxpi=20;
maxp=10;
u=rand(ntask-1,1)*2*totalu/ntask;
while sum(u)>=totalu
    u=rand(ntask-1,1)*2*totalu/ntask;
end
```

```

u=[u;totalu-sum(u)];
pi=ceil(rand(ntask,1)*maxpi);
m=pi.*u;
o=zeros(ntask,1);
for i=1:ntask
    o(i)=rand(1)*(pi(i)-m(i));
end
tau=m+o;
p=rand(ntask,1)*maxp;
id=(1:ntask)';
t=struct('id',num2cell(id),'m',num2cell(m),'tau',num2cell(tau),'pi',num2cell(pi),'p',num2cell(p));

```

```
%=====run.m=====
```

```

% example: t=gentask(1.5);
t=struct('id', {1, 2, 3}, 'm', {13, 4, 2.5}, 'tau', {15, 10, 5}, 'pi', {18, 10, 6}, 'p', {2, 10, 8});
p=struct('id', {0, 1}, 'w', {1, .5}, 'z', {1, 1});

```

```

fprintf(1, 'Algorithm 1 (EDF):\n');
simics('algo1',t,p);
fprintf(1, 'Algorithm 2 (RMS):\n');
simics('algo2',t,p);
fprintf(1, 'Algorithm 3 (LEF):\n');
simics('algo3',t,p);
fprintf(1, 'Algorithm 4 (MEF):\n');
simics('algo4',t,p);

```

```
%=====run=====
```

```

#!/bin/sh
octave --eval "run;"

```

```
%=====simics.m=====
```

```

function totalE=simics(str,t,p)
%periodic tasks with integer processing time, which can be
divided into mandatory and optional parts.

```

```

threshold=1e-8;
% make this a threshold thing
%body
sol=simics_sched(str, t, p);

```

```

if size(sol,2)==0
    disp('no schedule result. ');
    totalE=Inf;
    return
end

totalT=lcm(t.pi);
t2tid=[];
tinterval=[]; % The last size(sol,2) entries are schedule
times, preceded by moments when new tasks come in
if size(t,2)==1, tlen=size(t,1);, else, tlen=size(t,2);,
end
for i=1:tlen
    temp=0;
    while temp < totalT
        tinterval=[tinterval, temp];
        t2tid=[t2tid, i];
        temp=temp+t(i).pi;
    end
end
tinterval=[tinterval, sol.t1];
[tsorted, index] = sort(tinterval);
tsorted=[tsorted, totalT];

%config processor queue and todo list for every processor
pq(size(p,2))=struct('id', [], 't', []);
todo(size(p,2))=struct('id', [], 't', []);

%time loop
curtime=0;
totalerror=0;
for i=1:size(tsorted,2) % i is the seq # of time intervals
    if curtime < tsorted(i)
        % time flies
        for j=1:size(p,2) % j is the seq # of processors
            if size(todo(j).id, 2)>0 % if processor j has
anything to do
                for k=1:size(pq(j).id, 2) % then do it
                    if pq(j).id(k)==todo(j).id
                        pq(j).t(k)=pq(j).t(k)-(tsorted(i)-
curtime)/p(j).w;
                        todo(j).t=todo(j).t-(tsorted(i)-
curtime)/p(j).w;
                        if pq(j).t(k)<threshold, pq(j).id(k)=[];,
pq(j).t(k)=[];, end;
                        if todo(j).t<threshold, todo(j).id=[];,
todo(j).t=[];, end;
                        break;
                    end
                end
            end
        end
    end
end

```



```

        end
    end
end
end
curtime=tsorted(i);
if curtime==totalT
    left=0;
    for tid=1:tlen
        for j=1:size(p,2) % j is the seq # of processors
            for k=1:size(pq(j).id, 2)
                if pq(j).id(k)==tid, left=left+pq(j).t(k);,
break; end;
            end
        end
        fprintf(1, 'At %d, task %d committed with %f not
finished.\n', curtime, tid, left);
        totalerror=totalerror+left*t(tid).p;
    end
    fprintf(1, 'total error is %f.\n', totalerror);
    break;
end
end
if index(i) <= size(tinterval, 2) - size(sol, 2) % new
task comes
    tid = t2tid(index(i)); % tid is the seq # of the task
whose time has come.
    % commit computation result for task tid in processor
queue and todo list
    left=0;
    for j=1:size(p,2) % j is the seq # of processors
        if size(todo(j).id, 2)~=0 && todo(j).id(1)==tid,
todo(j).id=[];, todo(j).t=[];, end;
        for k=1:size(pq(j).id, 2)
            if pq(j).id(k)==tid, left=left+pq(j).t(k);,
pq(j).id(k)=[];, pq(j).t(k)=[];, break; end;
        end
    end
    if curtime~=0
        fprintf('At %d, task %d committed with %f not
finished.\n', curtime, tid, left);
        totalerror=totalerror+left*t(tid).p;
    end
    % add the new task to the processor queue for the root
processor
    pq(1).id=[pq(1).id, tid];
    pq(1).t=[pq(1).t, t(tid).tau];
else
    % schedule time
    solid=index(i)+size(sol,2)-size(tinterval,2);

```

```

pid=sol(solid).pid+1;
tid=sol(solid).tid;
if sol(solid).type==0 % communication
    k=find(pq(1).id==tid);
    if size(k, 2)==0 || pq(1).t(k(1)) <
sol(solid).amount-threshold
        disp('algorithm error!!!');
        return;
    elseif size(todo(1),2) ~= 0 && todo(1).id == tid &&
pq(1).t(k(1)) < todo(1).t + sol(solid).amount - threshold
        disp('algorithm error!!!');
        return;
    end
    pq(1).t(k(1))=pq(1).t(k(1))-sol(solid).amount;
    k=find(pq(pid).id==tid);
    if size(k,1)==0 || size(k, 2)==0
        pq(pid).id=[pq(pid).id, tid];
        pq(pid).t=[pq(pid).t, sol(solid).amount];
    else
        pq(pid).t(k(1))=pq(pid).t(k(1))+sol(solid).amount;
    end
else % execution
    k=find(pq(pid).id==tid);
    if size(k, 2)==0 || pq(pid).t(k(1)) <
sol(solid).amount-threshold
        disp('algorithm error!!!!a');
        return;
    elseif size(todo(pid).id, 2) ~= 0
        disp('algorithm error!!!!b');
        return;
    end
    todo(pid).id=tid;
    todo(pid).t=sol(solid).amount;
end
end
end
end

```

```
totalE=totalerror;
```

```
%===== toptest.m=====
```

```
%randomly generate 100 task sets each time increase
workload from 0.01 to 1.20 with increment step of 0.01 to
test the schedulable rate for each algorithm.
```

```
%choose a scheduling algorithm
```

```

%algo1=EDF, algo2=RMS, algo3=LEF(least execution time
first), algo4=MEF(most execution time first)
str='algo1';
ntask=100;%number of task sets
startu=0.01;%start utility of the system
stepu=0.01;%increment step
endu=1.2;%end utility of the system
%processors
p=struct('id', {0}, 'w', {1}, 'z', {1});

%Precise computation
fprintf(1, 'Algo 1 precise computation');
i=0;
for totalu=[startu:stepu:end]
    count1=0;% record the successful scheduled task number
    count2=0;% record scheduled task number
    fprintf(1, 'total utility of the system is %f.\n',
totalu);
% task sets
    for loops=0:1:ntask
        t= gentask(totalu);
        fprintf(1, 'task set %d.\n', count2);
        count2=count2+1;
        totalE=simics(str, t, p);
        if isinf(totalE)
            else
                count1=count1+1;
            end
        end
    end
    fprintf(1, 'total pass rate is %f.\n', count1/count2);
    probability(1,i+1)=count1/count2;
    i=i+1;
end

%Imprecise computation: task is divided into to tasks,
mandatory takes up certain percentage
percent=0.9;
fprintf(1, 'Algo 1 imprecise computation');
j=0;
for totalu=[startu:stepu:end]
    count1=0;% record the successful scheduled task number
    count2=0;% record scheduled task number
    fprintf(1, 'total utility of the system is %f.\n',
totalu);
% task sets
    for loops=0:1:ntask
        t= gentask(totalu*percent);
        fprintf(1, 'task set %d.\n', count2);
        count2=count2+1;
    end
end

```

```

        totalE=simics(str, t, p);
        if isinf(totalE)
        else
            count1=count1+1;
        end
    end
    fprintf(1, 'total pass rate is %f.\n', count1/count2);
    probability(2,j+1)=count1/count2;
    j=j+1;
end

totalu=[startu:stepu:end];
plot(totalu, probability);

str='algo2';
%Precise computation
fprintf(1, 'Algo 2 precise computation');
i=0;
for totalu=[startu:stepu:end]
    count1=0;% record the successful scheduled task number
    count2=0;% record scheduled task number
    fprintf(1, 'total utility of the system is %f.\n',
totalu);
% task sets
    for loops=0:1:ntask
        t= gentask(totalu);
        fprintf(1, 'task set %d.\n', count2);
        count2=count2+1;
        totalE=simics(str, t, p);
        if isinf(totalE)
        else
            count1=count1+1;
        end
    end
    fprintf(1, 'total pass rate is %f.\n', count1/count2);
    probability2(1,i+1)=count1/count2;
    i=i+1;
end

%Imprecise computation: task is divided into to tasks,
mandatory takes up certain percentage
percent=0.9;
fprintf(1, 'Algo 2 imprecise computation');
j=0;
for totalu=[startu:stepu:end]
    count1=0;% record the successful scheduled task number
    count2=0;% record scheduled task number

```

```

    fprintf(1, 'total utility of the system is %f.\n',
totalu);
% task sets
for loops=0:1:ntask
    t= gentask(totalu*percent);
    fprintf(1, 'task set %d.\n', count2);
    count2=count2+1;
    totalE=simics(str, t, p);
    if isinf(totalE)
    else
        count1=count1+1;
    end
end
fprintf(1, 'total pass rate is %f.\n', count1/count2);
probability2(2,j+1)=count1/count2;
j=j+1;
end

totalu=[startu:stepu:endu];
figure;
plot(totalu, probability2);

str='algo3';

%Precise computation
fprintf(1, 'Algo 3 precise computation');
i=0;
for totalu=[startu:stepu:endu]
    count1=0;% record the successful scheduled task number
    count2=0;% record scheduled task number
    fprintf(1, 'total utility of the system is %f.\n',
totalu);
% task sets
for loops=0:1:ntask
    t= gentask(totalu);
    fprintf(1, 'task set %d.\n', count2);
    count2=count2+1;
    totalE=simics(str, t, p);
    if isinf(totalE)
    else
        count1=count1+1;
    end
end
fprintf(1, 'total pass rate is %f.\n', count1/count2);
probability3(1,i+1)=count1/count2;
i=i+1;
end

```

```

%Imprecise computation: task is divided into to tasks,
mandatory takes up certain percentage
percent=0.9;
fprintf(1, 'Algo 3 imprecise computation');
j=0;
for totalu=[startu:stepu:end]
    count1=0;% record the successful scheduled task number
    count2=0;% record scheduled task number
    fprintf(1, 'total utility of the system is %f.\n',
totalu);
% task sets
for loops=0:1:ntask
    t= gentask(totalu*percent);
    fprintf(1, 'task set %d.\n', count2);
    count2=count2+1;
    totalE=simics(str, t, p);
    if isinf(totalE)
    else
        count1=count1+1;
    end
end
fprintf(1, 'total pass rate is %f.\n', count1/count2);
probability3(2,j+1)=count1/count2;
j=j+1;
end

totalu=[startu:stepu:end];
figure;
plot(totalu, probability3);

str='algo4';
%Precise computation
fprintf(1, 'Algo 4 precise computation');
i=0;
for totalu=[startu:stepu:end]
    count1=0;% record the successful scheduled task number
    count2=0;% record scheduled task number
    fprintf(1, 'total utility of the system is %f.\n',
totalu);
% task sets
for loops=0:1:ntask
    t= gentask(totalu);
    fprintf(1, 'task set %d.\n', count2);
    count2=count2+1;
    totalE=simics(str, t, p);
    if isinf(totalE)
    else
        count1=count1+1;
    end
end

```

```

end
fprintf(1, 'total pass rate is %f.\n', count1/count2);
probability4(1,i+1)=count1/count2;
i=i+1;
end

%Imprecise computation: task is divided into to tasks,
mandatory takes up certain percentage
percent=0.9;
fprintf(1, 'Algo 4 imprecise computation');
j=0;
for totalu=[startu:stepu:endu]
    count1=0;% record the successful scheduled task number
    count2=0;% record scheduled task number
    fprintf(1, 'total utility of the system is %f.\n',
totalu);
% task sets
    for loops=0:1:ntask
        t= gentask(totalu*percent);
        fprintf(1, 'task set %d.\n', count2);
        count2=count2+1;
        totalE=simics(str, t, p);
        if isinf(totalE)
            else
                count1=count1+1;
            end
        end
    end
    fprintf(1, 'total pass rate is %f.\n', count1/count2);
    probability4(2,j+1)=count1/count2;
    j=j+1;
end

totalu=[startu:stepu:endu];
figure;
plot(totalu, probability4);

probability5=[probability(1,:);probability2(1,:);probabilit
y3(1,:);probability4(1,:)];
probability6=[probability(2,:);probability2(2,:);probabilit
y3(2,:);probability4(2,:)];

figure;
plot(totalu, probability5);
figure;
plot(totalu, probability6);

%=====simics_sched.m (ICS_sched)=====

```

```

function sol=simics_sched(str, t, p)
%sol=struct('t1',{0, 1, 2, 3, 4, 5},'t2',{1, 2, 3, 4, 5,
6}, 'type', {1, 1, 1, 1, 1, 1}, 'pid', {0, 0, 0, 0, 0, 0},
'tid', {1, 2, 1, 2, 1, 2}, 'amount', {1, 1, 1, 1, 1, 1});
threshold=1e-8;

%EDF algorithm
if strcmp(str,'algol')
    %calculate equivalent W
    denominator=1;
    if size(p,2)==1
        W=p(1).w;
    else
        f=( [p(2:end).w]+[p(2:end).z] )./[p(1:end-1).w];
        for i=1:size(p,2)-1
            denominator=denominator+prod(f(i:end));
        end
        W=(prod(f))/(denominator)*p(1).w;
    end
end

    % construct time domain
    totalT=lcm(t.pi);
    t2tid=[];
    tinterval=[];
    if size(t,2)==1, tlen=size(t,1);, else, tlen=size(t,2);,
end
for i=1:tlen
    temp=0;
    while temp < totalT
        tinterval=[tinterval, temp];
        t2tid=[t2tid, i];
        temp=temp+t(i).pi;
    end
end
[tsorted, index] = sort(tinterval);
tsorted=[tsorted, totalT];

% start
Q(tlen)=struct('m', [], 'o', [], 'd', []);
pointer=1;
curtime=0;
i=1;
while i<=size(tsorted,2)
    if i==size(tsorted,2) && curtime==tsorted(i)
        % terminates
        return;
    elseif curtime==tsorted(i)
        % queue task up
        tid=t2tid(index(i));

```



```

        if size(Q(tid).m,2)==1 && Q(tid).m>threshold
            disp("can't schedule");
            sol=struct('t1', {}, 't2', {}, 'type', {}, 'pid', {},
'tid', {}, 'amount', {});
            return;
        end
        Q(tid)=struct('m', t(tid).m, 'o', t(tid).tau-
t(tid).m, 'd', curtime+t(tid).pi);
        i=i+1;
    elseif curtime < tsorted(i)
        % do scheduling
        nexttask=0;
        earliest=inf;
        for j=1:tlen
            if size(Q(j).m,2)==1 && Q(j).m>threshold &&
Q(j).d<earliest
                nexttask=j;
                earliest=Q(j).d;
            end
        end
        if earliest~=inf
            % execute task nexttask's mandatory part
            t1=curtime;
            exectime=min(tsorted(i)-curtime, W*Q(nexttask).m);
            curtime=curtime+exectime;
            Q(nexttask).m=Q(nexttask).m-exectime/W;
            if Q(nexttask).m<threshold &&
Q(nexttask).o<threshold, Q(nexttask).m=[];,
Q(nexttask).o=[];, Q(nexttask).d=[];, end;
            t2=curtime; tid=nexttask; amount=exectime/W;
        else
            % execute any task's optional part or idle
            t1=0;
            for j=1:tlen
                if size(Q(j).o,2)==1 && Q(j).o>threshold &&
t(j).p>t1, nexttask=j;, end;
            end
            if nexttask==0
                exectime=0;
            else
                t1=curtime;
                exectime=min(tsorted(i)-curtime, W*Q(nexttask).o);
                curtime=curtime+exectime;
                Q(nexttask).o=Q(nexttask).o-exectime/W;
                if Q(nexttask).o<threshold, Q(nexttask).m=[];,
Q(nexttask).o=[];, Q(nexttask).d=[];, end;
                t2=curtime; tid=nexttask; amount=exectime/W;
            end
        end
    end
end

```

```

    % schedule
    if exectime~=0
        alpha=zeros(1,size(p,2));
        for j=1:size(p,2)-1
            alpha(j)=prod(f(j:end))/denominator;
        end
        alpha(size(p,2))=1/denominator;
        sol(pointer)=struct('t1', t1, 't2', t2, 'type', 1,
'pid', 0, 'tid', tid, 'amount', alpha(1)*amount);
        pointer=pointer+1;
        for j=2:size(p,2)
            sol(pointer)=struct('t1', t1, 't2',
t1+alpha(j)*p(j).z*amount, 'type', 0, 'pid', j-1, 'tid',
tid, 'amount', alpha(j)*amount);
            pointer=pointer+1;
            sol(pointer)=struct('t1', t1+alpha(j)*p(j).z*amount,
't2', t2, 'type', 1, 'pid', j-1, 'tid', tid, 'amount',
alpha(j)*amount);
            pointer=pointer+1;
        end
    else
        curtime=tsorted(i);
    end
else
    disp('error');
    break;
end
end % while i<=size(tsorted,2)

%RMS algoritihm

elseif strcmp(str,'algo2')

    %calculate equivalent W
    denominator=1;
    if size(p,2)==1
        W=p(1).w;
    else
        f=( [p(2:end).w]+[p(2:end).z] )./[p(1:end-1).w];
        for i=1:size(p,2)-1
            denominator=denominator+prod(f(i:end));
        end
        W=(prod(f))/(denominator)*p(1).w;
    end

    % construct time domain
    totalT=lcm(t.pi);
    % Compute the least common multiple for all the periods of
    tasks.

```

```

t2tid=[];
tinterval=[];
if size(t,2)==1, tlen=size(t,1);, else, tlen=size(t,2);,
end
for i=1:tlen
    temp=0;
    while temp < totalT
        tinterval=[tinterval, temp];
        t2tid=[t2tid, i];
        temp=temp+t(i).pi;
    end
end
[tsorted, index] = sort(tinterval);
tsorted=[tsorted, totalT];
%"tsorted" is the deadlines of all the tasks

% start
Q(tlen)=struct('m', [], 'o', [], 'd', []);
pointer=1;
curtime=0;
i=1;
while i<=size(tsorted,2)
    if i==size(tsorted,2) && curtime==tsorted(i)
        % terminates
        return;
    elseif curtime==tsorted(i)
        % queue task up
        tid=t2tid(index(i));

        if size(Q(tid).m,2)==1 && Q(tid).m>threshold
            disp("can't schedule");
            sol=struct('t1', {}, 't2', {}, 'type', {}, 'pid', {},
'tid', {}, 'amount', {});
            return;
        end
        %Q(tid)=struct('m', t(tid).m, 'o', t(tid).tau-
t(tid).m, 'd', curtime+t(tid).pi);
        Q(tid)=struct('m', t(tid).m, 'o', t(tid).tau-
t(tid).m, 'd', t(tid).pi);
        i=i+1;
    elseif curtime < tsorted(i)
        % do scheduling
        nexttask=0;
        shortestperiod=inf;

        for j=1:tlen
            if size(Q(j).m,2)==1 && Q(j).m>threshold &&
Q(j).d<shortestperiod
                nexttask=j;
            end
        end
    end
end

```



```

        sol(pointer)=struct('t1', t1+alpha(j)*p(j).z*amount,
't2', t2, 'type', 1, 'pid', j-1, 'tid', tid, 'amount',
alpha(j)*amount);
        pointer=pointer+1;
    end
    else
        curtime=tsorted(i);
    end
    else
        disp('error');
        break;
    end
end
end

%LEF: least execution time first algorithm
elseif strcmp(str,'algo3')

    %calculate equivalent W
    denominator=1;
    if size(p,2)==1
        W=p(1).w;
    else
        f=( [p(2:end).w]+[p(2:end).z] )./[p(1:end-1).w];
        for i=1:size(p,2)-1
            denominator=denominator*prod(f(i:end));
        end
        W=(prod(f))/(denominator)*p(1).w;
    end

    % construct time domain
    totalT=lcm(t.pi);
    % Compute the least common multiple for all the periods of
    tasks.
    t2tid=[];
    tinterval=[];
    if size(t,2)==1, tlen=size(t,1);, else, tlen=size(t,2);,
end
    for i=1:tlen
        temp=0;
        while temp < totalT
            tinterval=[tinterval, temp];
            t2tid=[t2tid, i];
            temp=temp+t(i).pi;
        end
    end
    [tsorted, index] = sort(tinterval);
    tsorted=[tsorted, totalT];
    %"tsorted" is the deadlines of all the tasks

```

```

% start
Q(tlen)=struct('m', [], 'o', [], 'd', []);
pointer=1;
curtime=0;
i=1;
while i<=size(tsorted,2)
    if i==size(tsorted,2) && curtime==tsorted(i)
        % terminates
        return;
    elseif curtime==tsorted(i)
        % queue task up
        tid=t2tid(index(i));
        if size(Q(tid).m,2)==1 && Q(tid).m>threshold
            disp("can't schedule");
            sol=struct('t1', {}, 't2', {}, 'type', {}, 'pid', {},
'tid', {}, 'amount', {});
            return;
        end
        %Q(tid)=struct('m', t(tid).m, 'o', t(tid).tau-
t(tid).m, 'd', curtime+t(tid).pi);
        Q(tid)=struct('m', t(tid).m, 'o', t(tid).tau-
t(tid).m, 'd', t(tid).m);
        i=i+1;
    elseif curtime < tsorted(i)
        % do scheduling
        nexttask=0;
        leastexect=inf;
        for j=1:tlen
            if size(Q(j).m,2)==1 && Q(j).m>threshold &&
Q(j).d<leastexect
                nexttask=j;
                leastexect=Q(j).d;
            end
        end
        if leastexect~=inf
            % execute task nexttask's mandatory part
            t1=curtime;
            exectime=min(tsorted(i)-curtime, W*Q(nexttask).m);
            curtime=curtime+exectime;
            Q(nexttask).m=Q(nexttask).m-exectime/W;
            if Q(nexttask).m<threshold &&
Q(nexttask).o<threshold, Q(nexttask).m=[];,
Q(nexttask).o=[];, Q(nexttask).d=[];, end;
            t2=curtime; tid=nexttask; amount=exectime/W;
        else
            % execute any task's optional part or idle
            t1=0;
            for j=1:tlen

```

```

        if size(Q(j).o,2)==1 && Q(j).o>threshhold &&
t(j).p>t1, nexttask=j;; end;
    end
    if nexttask==0
        exectime=0;
    else
        t1=curtime;
        exectime=min(tsorted(i)-curtime, W*Q(nexttask).o);
        curtime=curtime+exectime;
        Q(nexttask).o=Q(nexttask).o-exectime/W;
        if Q(nexttask).o<threshhold, Q(nexttask).m=[];,
Q(nexttask).o=[];, Q(nexttask).d=[];, end;
        t2=curtime; tid=nexttask; amount=exectime/W;
    end
    end
    % schedule
    if exectime~=0
        alpha=zeros(1,size(p,2));
        for j=1:size(p,2)-1
            alpha(j)=prod(f(j:end))/denominator;
        end
        alpha(size(p,2))=1/denominator;
        sol(pointer)=struct('t1', t1, 't2', t2, 'type', 1,
'pid', 0, 'tid', tid, 'amount', alpha(1)*amount);
        pointer=pointer+1;
        for j=2:size(p,2)
            sol(pointer)=struct('t1', t1, 't2',
t1+alpha(j)*p(j).z*amount, 'type', 0, 'pid', j-1, 'tid',
tid, 'amount', alpha(j)*amount);
            pointer=pointer+1;
            sol(pointer)=struct('t1', t1+alpha(j)*p(j).z*amount,
't2', t2, 'type', 1, 'pid', j-1, 'tid', tid, 'amount',
alpha(j)*amount);
            pointer=pointer+1;
        end
    else
        curtime=tsorted(i);
    end
    else
        disp('error');
        break;
    end
end
end

%MEF: most execution time first algorithm
elseif strcmp(str,'algo4')

    %calculate equivalent W
    denominator=1;

```

```

if size(p,2)==1
    W=p(1).w;
else
    f=( [p(2:end).w]+[p(2:end).z] )./[p(1:end-1).w];
    for i=1:size(p,2)-1
        denominator=denominator+prod(f(i:end));
    end
    W=(prod(f))/(denominator)*p(1).w;
end

% construct time domain
totalT=lcm(t.pi);
% Compute the least common multiple for all the periods of
tasks.
t2tid=[];
tinterval=[];
if size(t,2)==1, tlen=size(t,1);, else, tlen=size(t,2);,
end
for i=1:tlen
    temp=0;
    while temp < totalT
        tinterval=[tinterval, temp];
        t2tid=[t2tid, i];
        temp=temp+t(i).pi;
    end
end
[tsorted, index] = sort(tinterval);
tsorted=[tsorted, totalT];
%"tsorted" is the deadlines of all the tasks

% start
Q(tlen)=struct('m', [], 'o', [], 'd', []);
pointer=1;
curtime=0;
i=1;
while i<=size(tsorted,2)
    if i==size(tsorted,2) && curtime==tsorted(i)
        % terminates
        return;
    elseif curtime==tsorted(i)
        % queue task up
        tid=t2tid(index(i));
        if size(Q(tid).m,2)==1 && Q(tid).m>threshold
            disp("can't schedule");
            sol=struct('t1', {}, 't2', {}, 'type', {}, 'pid', {},
'tid', {}, 'amount', {});
            return;
        end
    end
end

```



```

        %Q(tid)=struct('m', t(tid).m, 'o', t(tid).tau-
t(tid).m, 'd', curtime+t(tid).pi);
        Q(tid)=struct('m', t(tid).m, 'o', t(tid).tau-
t(tid).m, 'd', t(tid).m);
        i=i+1;
    elseif curtime < tsorted(i)
        % do scheduling
        nexttask=0;
        largestexect=0;
        for j=1:tlen
            if size(Q(j).m,2)==1 && Q(j).m>threshhold &&
Q(j).d>largestexect
                nexttask=j;
                largestexect=Q(j).d;
            end
        end
        if largestexect~=0
            % execute task nexttask's mandatory part
            t1=curtime;
            exectime=min(tsorted(i)-curtime, W*Q(nexttask).m);
            curtime=curtime+exectime;
            Q(nexttask).m=Q(nexttask).m-exectime/W;
            if Q(nexttask).m<threshhold &&
Q(nexttask).o<threshhold, Q(nexttask).m=[];,
Q(nexttask).o=[];, Q(nexttask).d=[];, end;
            t2=curtime; tid=nexttask; amount=exectime/W;
        else
            % execute any task's optional part or idle
            t1=0;
            for j=1:tlen
                if size(Q(j).o,2)==1 && Q(j).o>threshhold &&
t(j).p>t1, nexttask=j;, end;
            end
            if nexttask==0
                exectime=0;
            else
                t1=curtime;
                exectime=min(tsorted(i)-curtime, W*Q(nexttask).o);
                curtime=curtime+exectime;
                Q(nexttask).o=Q(nexttask).o-exectime/W;
                if Q(nexttask).o<threshhold, Q(nexttask).m=[];,
Q(nexttask).o=[];, Q(nexttask).d=[];, end;
                t2=curtime; tid=nexttask; amount=exectime/W;
            end
        end
        % schedule
        if exectime~=0
            alpha=zeros(1, size(p,2));
            for j=1:size(p,2)-1

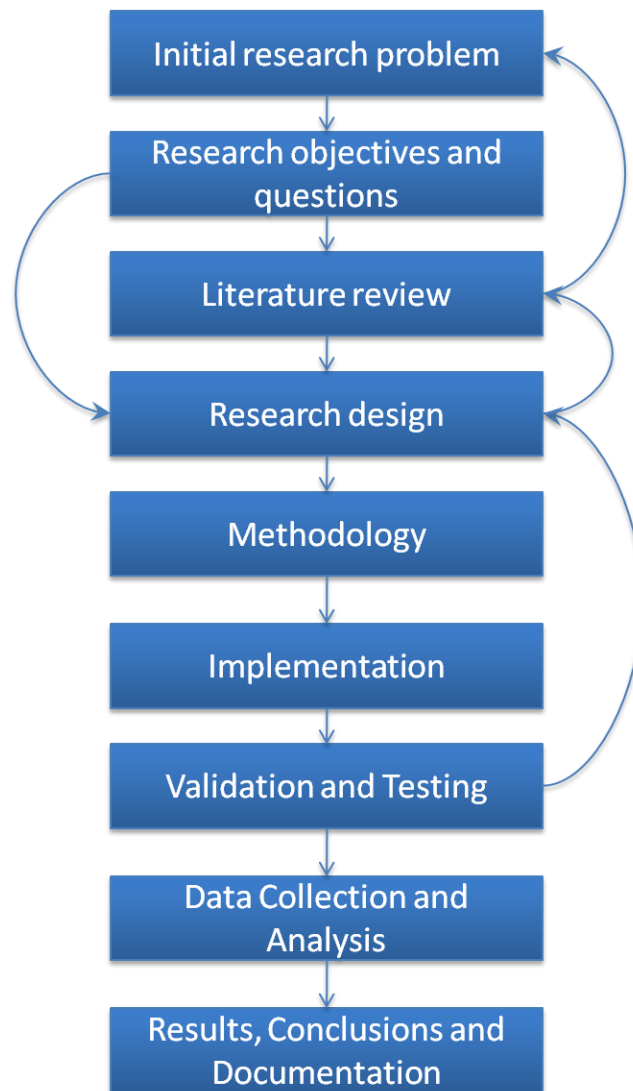
```

```

        alpha(j)=prod(f(j:end))/denominator;
    end
    alpha(size(p,2))=1/denominator;
    sol(pointer)=struct('t1', t1, 't2', t2, 'type', 1,
'pid', 0, 'tid', tid, 'amount', alpha(1)*amount);
    pointer=pointer+1;
    for j=2:size(p,2)
        sol(pointer)=struct('t1', t1, 't2',
t1+alpha(j)*p(j).z*amount, 'type', 0, 'pid', j-1, 'tid',
tid, 'amount', alpha(j)*amount);
        pointer=pointer+1;
        sol(pointer)=struct('t1', t1+alpha(j)*p(j).z*amount,
't2', t2, 'type', 1, 'pid', j-1, 'tid', tid, 'amount',
alpha(j)*amount);
        pointer=pointer+1;
    end
    else
        curtime=tsorted(i);
    end
else
    disp('error');
    break;
end
end
end
end

```

## I. Research Process



*Figure 77 Flowchart of research process*

## **PUBLICATIONS**

---

---

1. Chen Guo, Cenzhe Zhu, Teng Tiow Tay “ShAppliT: A Novel Broker-mediated Solution to Generic Application Sharing in a Cluster of Closed Operating Systems”, International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 2, No. 6, pp. 16-32, June 2012. Doi: 10.7321/jscse.v2.n6.2
2. Chen Guo, Cenzhe Zhu, Teng Tiow Tay, “Design and Simulation of a Green Broker with Imprecise Computation Scheduling for Energy-efficient Large Scale Computing in Clusters”, Journal of Emerging Trends in Computing and Information Sciences, Vol. 4, No. 12, December 2013
3. Chen Guo, Cenzhe Zhu and Teng Tiow Tay “Sharing of Generic Single-user Application without Interference in a Cluster”, 2012 International Conference on Software and Computer Applications, IPCSIT vol. 41, Page(s):101-105 (2012) IACSIT Press, Singapore
4. Chen Guo, Teng Tiow Tay “Implementing a Peer-to-peer Application Sharing Tool on Windows Clustering System”, International work shop on Cloud Computing- Architecture, Algorithms and Applications (Cloudcomp 2011), India
5. Chen Guo, Teng Tiow Tay “On Scalability Migratability and Cost-effectiveness of Next-Generation WDM Passive Optical Network Architectures”, IEEE Sixth International Conference on Signal Processing and Communication Systems, ICSPCS 2012, Gold Coast, Australia