DISTRIBUTED PREEMPTIVE PROCESS MANAGEMENT WITH CHECKPOINTING AND MIGRATION FOR A LINUX-BASED GRID OPERATING SYSTEM

by

HTIN PAW OO @ NUR HUSSEIN

Thesis submitted in fulfilment of the requirements
for the degree of
Master of Science

June 2006

To my parents

ACKNOWLEDGEMENTS

I thank God for giving me the strength to persevere through all the trials of this postgraduate degree. I would like to thank my late father Dr. Mohd. Yunus for inspiring me to get into research, my mother Pn. Bibi Zahida for patiently supporting me throughout this experience, my supervisors Dr. Fazilah Haron and Dr. Chan Huah Yong for their wisdom and guidance, and also the entire faculty of the School of Computer Science in Universiti Sains Malaysia.

I would like to convey my deepest gratitude to Dr. Abdul Rahman Othman, Dr. Jim Basney, Dr. Mor Harchol-Balter, Dr. Low Heng Chin, Dr. Yen Siew Hwa, and Dr. Quah Soon Hoe for taking some of their time to shed some insight into my research problem.

I would also like to thank Josef Sipek for his help and for lending me his computer cluster, Lim Lian Tze for teaching me LaTeX, Dr. Con Kolivas for his scheduler expertise, and for their expert technical (and nontechnical) advice, Seth Arnold, Dr. Daniel Berlin, Jeff Dike, Pat Erley, Henrý Þór Baldursson, Bert Hubert, William Lee Irwin III, Christian Leber, Ahmed Masud, Zwane Mwaikambo, Eduardo Pinheiro, Rik Van Riel, Steinn Sigurðarson and the rest of my friends out there on the internet who have patiently endured my questions over the past three years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PENGURUSAN PROSES PREEMPTIF TERAGIH DENGAN PENITIKSEMAKAN DAN MIGRASI UNTUK SISTEM PENGOPERASIAN GRID BERASASKAN LINUX

ABSTRAK

Kemunculan perkomputeran grid telah membolehkan perkongsian sumber perkomputeran teragih antara peserta-peserta organisasi maya. Walau bagaimanapun, sistem pengoperasian kini tidak memberi sokongan paras rendah secukupnya untuk perlaksanaan perisian grid. Kemunculan suatu kelas sistem pengoperasian yang dipanggil sistem pengoperasian grid memberikan pengabstrakan peringkat sistem untuk sumber-sumber grid. Tesis ini mencadangkan penambahan pengurusan proses preemptif teragih kepada sistem pengoperasian GNU/Linux untuk menjadikannya sistem pengoperasian grid. Dengan menampal inti Linux dengan kemudahan penitiksemakan yang dipanggil EPCKPT, pembuktian konsep perisian tengah yang dipanggil Zinc telah dibina. Perisian Zinc menggunakan kemudahan penitiksemakan dengan cekap untuk membolehkan pengurusan proses teragih yang merangkumi penskedulan, penempatan proses grid dan migrasi proses grid. Dengan menggunakan daya pemprosesan (*throughput*) sebagai metrik pengukuran prestasi, kecekapan kemudahan migrasi proses telah diukur pada pelantar ujian grid yang terdiri daripada kluster PC di Pusat Pengajian Sains Komputer, Universiti Sains Malaysia. Proses-proses grid juga telah berjaya dimigrasikan melalui internet. Eksperimen telah dijalankan yang menunjukkan bahawa migrasi proses preemptif yang dijalankan oleh sistem pengoperasian membantu mengekalkan daya pemprosesan (*throughput*) yang tinggi tidak mengira strategi penempatan proses yang digunakan.

DISTRIBUTED PREEMPTIVE PROCESS MANAGEMENT WITH CHECKPOINTING AND MIGRATION FOR A LINUX-BASED GRID OPERATING SYSTEM

ABSTRACT

The advent of grid computing has enabled distributed computing resources to be shared amongst participants of virtual organisations. However, current operating systems do not adequately provide enough low-level facilities to accommodate grid software. There is an emerging class of operating systems called grid operating systems which provide systems-level abstractions for grid resources. This thesis proposes the addition of preemptive distributed process management to GNU/Linux, thus building a subset of the required functionality to turn GNU/Linux into a grid operating system. By patching the Linux kernel with a popular checkpointing facility called EPCKPT, a proof-of-concept grid middleware called Zinc was constructed which effectively makes use of checkpointing to provide distributed process management which encompasses scheduling, placement and migration of grid processes. By using job throughput as our performance metric, the effectiveness of the process migration facility was measured on a testbed grid which consisted of PC clusters in the School of Computer Science at Universiti Sains Malaysia. Grid processes were also successfully migrated over the internet. An experiment was carried out that showed that preemptive process migration in the operating system helps maintain system throughput that is consistently high, regardless of the process placement strategy used.

CHAPTER 1

INTRODUCTION

1.1 Concerning Grid Computing

The growing ubiquity of cheap computing power and high-speed networks have given birth to distributed computing, which combine the resources of networked computers and harness the resulting combined power of its constituent computing elements. Tanenbaum and Van Steen [74] describe distributed systems as "a collection of independent computers that appears to its users as a single coherent system". From this definition, it could be inferred that a distributed system has a generic goal of providing a transparent and coherent service to users of systems comprising more than one physical computing machine. It could be said that grid computing is a special instance of distributed systems. Grid technology allows us to collectively perform complex computational tasks that would not be feasible on a single computer by means of pooling together resources that are shared by various institutions, organisations and individuals. Foster and Kesselman define computational grids as "hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities"[22].

Grid computing grew out of metacomputing, an early effort to consolidate disparate and diverse computing resources to take advantage of the resulting combined computing power. Previously, a user trying to utilise such a wide collection of different resources had to put up with manually configuring and scheduling jobs on different user accounts, machines and programs. Current developments have produced automated tools and advanced job scheduling and monitoring technologies to assist the user in the sharing of this collection of resources. These technologies form the fabric of grid computing.

Various approaches have been taken to designing software that control and facilitate the computational grid. Usually, grid software is implemented as middleware, a layer of abstraction that lies between user programs and the host computational hardware and software. Examples of such software are Globus [21], Condor [46] and Legion [27]. These systems provide a collection of services for both users and user programs to help aggregate and share computing resources.

This chapter is a prelude to the design and implementation of Zinc, a layer of grid software for the GNU/Linux operating system developed with the idea of making grid middleware as transparent and as easy to use as possible, while maximising the job throughput of the grid. The design of Zinc is from a perspective of an operating systems programmer, while its implementation covers kernel modifications and userspace tools to support those modifications. The primary goal of the system is to provide a foundation for which we can experiment with the possibilities opened up by extending the operating system to accommodate grid computing, with regard to process management. These extensions can be seen as a first step towards the creation of a *grid operating system*, which is an operating system that provides an abstraction of grid services to make the technology more transparent and easy to use by both end-users and programmers.

1.2 Research Motivation

1.2.1 Investigating The Factors That Influence Job Throughput

Much literature has been written about the effect of process migration, scheduling algorithms and other aspects of distributed computing on high-performance problems. However, research on high throughput computing have not been as extensive, and warrants further investigation. We believe that the grid's primary function is an enabler of high-throughput computing. Although many or most hardware in the grid is going to provide high-performance computing facilities to its users, the entire system as a whole exists to maximise the amount of work done with the resources available. Hence, looking into what conditions are favourable to increase the throughput in our distributed system is justified, and the results of our observations can be used to build better grids.

1.2.2 Introducing Distributed Process Management Support Into GNU/Linux

GNU/Linux[1] is a Unix-like operating system which is worked on by various programmers over the world, both voluntarily and or for commercial purposes sponsored by various companies. GNU/Linux is a collection of open source programs that make up a free operating system which can be modified and redistributed by anyone. At the heart of GNU/Linux is the Linux kernel, a free operating system kernel licensed under the GNU General Public License (GPL). Linux was initiated by Finnish programmer Linus Torvalds, and at the time of writing, he continues to spearhead its development in collaboration with thousands of developers world wide to further improve and enhance the Linux kernel. The userspace of GNU/Linux consists largely of utilities derived from the GNU project founded by MIT hacker Richard Stallman to create a truly free Unix-like operating system. Since GNU/Linux tries to be a clone of Unix, it is also a centralised network-enabled operating system by design. We have thus chosen GNU/Linux as a platform for our grid operating system research, so that it can be extended to facilitate grid-specific requirements and

[1]In print, the usage of both the terms "GNU/Linux" and "Linux" refer to the operating system based on the Linux kernel. There is a difference of opinion on whether or not the "GNU" part should be included when referring to the OS, but for the purpose of this thesis, the distinction between GNU/Linux and Linux is that GNU/Linux refers to the complete operating system (with userspace, C libraries, compilers and all) whereas Linux refers to just the Linux kernel.

investigate process migration.

Most of the tools available in GNU/Linux distributions are clones of the original Unix tools, or new software developed from scratch to make a functional desktop and server. However, neither the Linux kernel nor the userspace of GNU/Linux is designed with grid extensions in mind. There are a number of kernel-related projects which provide check-pointing and facilities for process migration such as MOSIX [3], but have not been fully developed for the purpose of internet-based migration for grid computing. MOSIX assumes a persistent, reliable and high-bandwidth network connection is available between hosts in the distributed system, an assumption which we can not make for internet migration.

Most grid software such as Globus [21] or Condor [46] have GNU/Linux versions, but few projects attempt to fully integrate the userspace tools with added functionality in the operating system. Moreover, most distributed operating system projects were initiated before grids became popular, thus there is little effort to support grid computing in research distributed operating systems. However, the Plan 9 [53] operating system is quite well-suited to grid computing, because of it's unique resource abstraction mechanism that presents everything on the system as network accessible files, even CPUs and other devices. However, Plan 9 does not support process migration or dynamic load sharing. Also, the problem with using research operating systems is that there is very little hardware and software support for them (most do not implement the full feature set of modern Unix-like systems), and all applications that want to take advantage of the system must be at least recompiled, if not rewritten (plus there is inertia when users need to switch operating systems).

Therefore, our motivation in choosing GNU/Linux as our vessel for investigating the issues surrounding distributed process management, wide-area process migration, and grid operating system design is because of the following:

1. Non-restrictive licensing terms for copying, modifying and redistributing the system.
2. Customisable open source kernel, and userspace software.
3. A wide range of free developer tools plus support for almost all major programming languages.
4. Very popular and supports a wide range of hardware device drivers.
5. Popular computational, scientific and grid software is available for it. These existing tools could benefit from additional grid-specific improvements.
6. It is evolving rapidly. Every few months, a new Linux kernel is released, with more feature added with each release. This rapid development process gives ample opportunity for new functionality to be included into a popular operating system (at least, gradually). This helps overcome the inertia of organisations and users refusing to totally change their operating systems, a problem described by the Legion team in Grimshaw *et al.* [26].

While GNU/Linux has been used extensively in grid computing, it was not designed as a grid operating system from the ground up. Padala [58] has proposed some enhancements to the network stack to the Linux kernel for improving network performance for grid applications. However, there have yet been no attempts to add on grid functionality to to GNU/Linux at a more fundamental operating system design level. Our research explores the idea of what a grid OS should look like, and proposes the design of a system for distributed process management as an enhancement to the GNU/Linux operating system.

1.2.3 Reducing Cruft

The New Hacker's Dictionary [63] defines "cruft" as excess, superfluous junk; used especially of redundant or superseded code. Crufty software is software with a design that's overly (and perhaps unnecessarily) complex. The design philosophy of the Globus toolkit is to work at a middleware layer, using only internet protocols. The justification for this decision as presented in [23] is to enable Globus to work on heterogeneous architectures and operating systems, and that "traditional transparencies are unobtainable" for grids. However the introduction of various new APIs in each of the components of Globus also increases the complexity of utilising a grid.

We disagree with Foster and Kesselman that trading off transparency and simplicity for heterogeneity is a necessary compromise in creating a grid. In Gabriel's essay on the design of Lisp [24], he characterises two software design strategies, one called "The MIT Approach" and another called "Worse-Is-Better". Both approaches stress the simplicity of design, where the "MIT Approach" would try and do the "right thing", where simplicity of the interface is more important than the simplicity of the implementation, whereas the "Worse-Is-Better" philosophy it is the other way around. The Globus toolkit however is both complex in terms of interface and implementation, which is in sharp disagreement with both software design philosophies. The simplicity of the design philosophy of Unix influenced the proposal of the implementation of a grid operating system in this thesis.

We also assert that it is possible to provide extra transparency via operating systems modifications while maintaining the same amount of support for heterogeneous platforms as Globus does now. Since operating system extensions are mostly transparent to userspace, it is possible for toolkits such as Globus to make use of the underlying grid features when available and still achieve its goals. The advantage of operating system support for grid functionality however, is that given a sufficiently large collection of computers of the same architecture, it is possible to create a grid without complex middleware toolkits. Considering that the Intel x86 architecture continues to be the most prevalent computing platform, plus the growing popularity of the GNU/Linux operating system, it is not inconceivable that a computational grid of reasonable size and usefulness can be constructed with relatively homogeneous hardware.

1.2.4 Enabling Internet Computing

There exists a vast pool of computing resources worldwide, and the advent of fast internet technologies have enabled organisations willing to share their computing facilities to do so at an unprecedented level. Currently, projects such as SETI@Home [1] and Folding@Home [68] create a high-throughput computational environment via specialised programs to perform their tasks. We hope that with a grid operating system, it will be possible to easily create generic programs that work like SETI@home and Folding@Home.

1.3 Research Objectives

1.3.1 Integrating Process Checkpointing And Grid Process Scheduling Into The Linux Kernel

Currently, the Linux kernel does not support process checkpointing, a feature necessary for process migration to work. Therefore, we will port and update the EPCKPT [61] checkpointing patch into the Linux kernel 2.4.22. We will also tweak the default kernel process scheduler to better handle CPU-intensive processes by enforcing a policy that favours long-running processes and by allowing userspace to have better control over process priorities. This will allow us the necessary functionality to create a foundation for our process migration and grid process management research.

1.3.2 Creating A Prototype Grid Process Management System

With the necessary modifications to the Linux kernel, we will thus build a proof-of-concept grid process management software in userspace called Zinc. It will incorporate a userspace scheduler, monitoring daemons and command line tools for the user to submit jobs. The design goals for the system are as follows:

1. Transparency – the user must be able to submit regular programs as jobs in the grid without modification
2. High throughput – the system will try to accomplish the most amount of work for as long as it runs
3. Adaptability to dynamic resources – the system will adapt to the variable conditions of the grid
4. Decentralisation – the system must reflect the decentralised nature of the grid
5. Minimising residual dependencies – the system must try to minimise the residual dependencies of a process when migrating it

With this prototype system, we will have a controlled environment that will enable us to perform further experiments on throughput in a grid operating system.

1.3.3 Enabling Wide-Area Process Migration

For the paradigm of "grid processes" to be complete, we must allow processes to migrate around the grid to any node connected to the system. This requires that processes be able to migrate over wide areas, as the grid is a large-scale distributed system that may even span continents. Therefore, we will determine whether grid process migration is feasible over a wide area by conducting an experiment to see the time required to migrate grid processes over the internet across continents.

1.3.4 Investigating The Factors Influencing Throughput In The Grid OS

We are interested in the question of whether or not preemptive process migration will help job throughput in the grid operating system, and also the factors that influence throughput. Since grid operating systems are still in their infancy, we will use the prototype that we develop to conduct our experiments, as we will be able to control external factors while implementing only the features that we need.

1.4 Scope Of Research

In defining the extensions to operating systems (in this case, Linux) for grid computing, substantial changes need to be done to all the subsystems of the OS. However, for the purpose of this thesis, we will restrict the scope of the implementation to distributed process management and process migration on the grid for the purpose of experimentation and implementation of a prototype kernel. Thus, grid filesystems, I/O, distributed device management, and distributed memory management was not implemented. These topics however, are discussed briefly in the last chapter. The implementation of process migration assumes the processes will not be performing inter-process communication. Thus the processes are "atomic" and may move about freely independent of other processes. The last chapter also discusses a scenario where IPC is allowed between processes and how wide-area process migration may take place in such a situation.

In the implementation of the Zinc grid process management framework, our goal is to create a proof-of-concept system to provide for us a controlled experimental test-bed to test the feasibility of wide-area process migration and to investigate the factors that influence throughput in the grid OS. Therefore, no benchmarking will be done to compare Zinc with existing similar systems such as Condor or MOSIX, as the latter projects have different design goals, thus a meaningful benchmark is not possible without compromising our own goals.

1.5 Contributions

This thesis explores the outcomes of adding explicit features to support grid computing into the Linux operating system. The primary contribution of this research is the introduction of the concept of grid process management and global process migration via the internet to GNU/Linux. Grid process management is a subset of the functionality required for a grid operating system, and is the subset that was chosen as a focus for this thesis. The design issues with Linux that need to be addressed when extending the operating system for grid process management were identified. The goal of the grid process management implementation is to provide transparent, wide-area process migration and a means to manage the aforementioned processes. To this end the EPCKPT [61] checkpointing patch available on the internet was applied to the Linux kernel as a foundation for the distributed process management algorithms.

A two-level scheduling system was designed for the grid operating system which consists of a modified kernel scheduler which was produced in collaboration with Linux kernel developers [32], and a userspace distributed process scheduler which was implemented in a program called Zinc. Zinc is a prototype proof-of-concept implementation of resource discovery, process scheduling and execution monitoring software that takes advantage of the checkpointing mechanism and the kernel scheduler in the modified Linux kernel.

Within the Zinc userspace scheduler, an algorithm called Zinctask was introduced for placement of jobs on a distributed system which makes decisions based on state information collected from all nodes in the system. This algorithm improves upon placement algorithms based on run queue length alone [19] by adding information on full distributed state via the Name Dropper resource discovery algorithm. The Zinctask algorithm also takes into account the staleness of state information when making decisions, as well as memory and CPU loads of the system. Zinctask is evaluated against random placement of processes and is found to be superior to it in almost all scenarios in the grid test-bed used for the experiments. Together with Zinctask and process migration, the Zinc-enabled Linux-based grid operating system yields both high throughput and creates an efficient load-distribution system.

Finally, with the implemented prototype systems and grid test-bed, the factors influencing the throughput of grid computing jobs were studied. The factors of interest are:

1. The availability of preemptive processes migration.
2. The placement strategy of processes.
3. The configuration of the machines in the grid.
4. The length of the majority of jobs that are submitted to the grid.

It was discovered that the different interactions between these factors influence throughput on the grid test-bed, and certain combination of factors produce different levels of throughput.

1.6 Summary And Organisation Of Thesis

This research aims to bridge the gap between existing grid middleware and operating systems development both of which are currently not integrating in a way to provide transparency to the user. Our goal of unifying these domains is presented in figure 1.1.

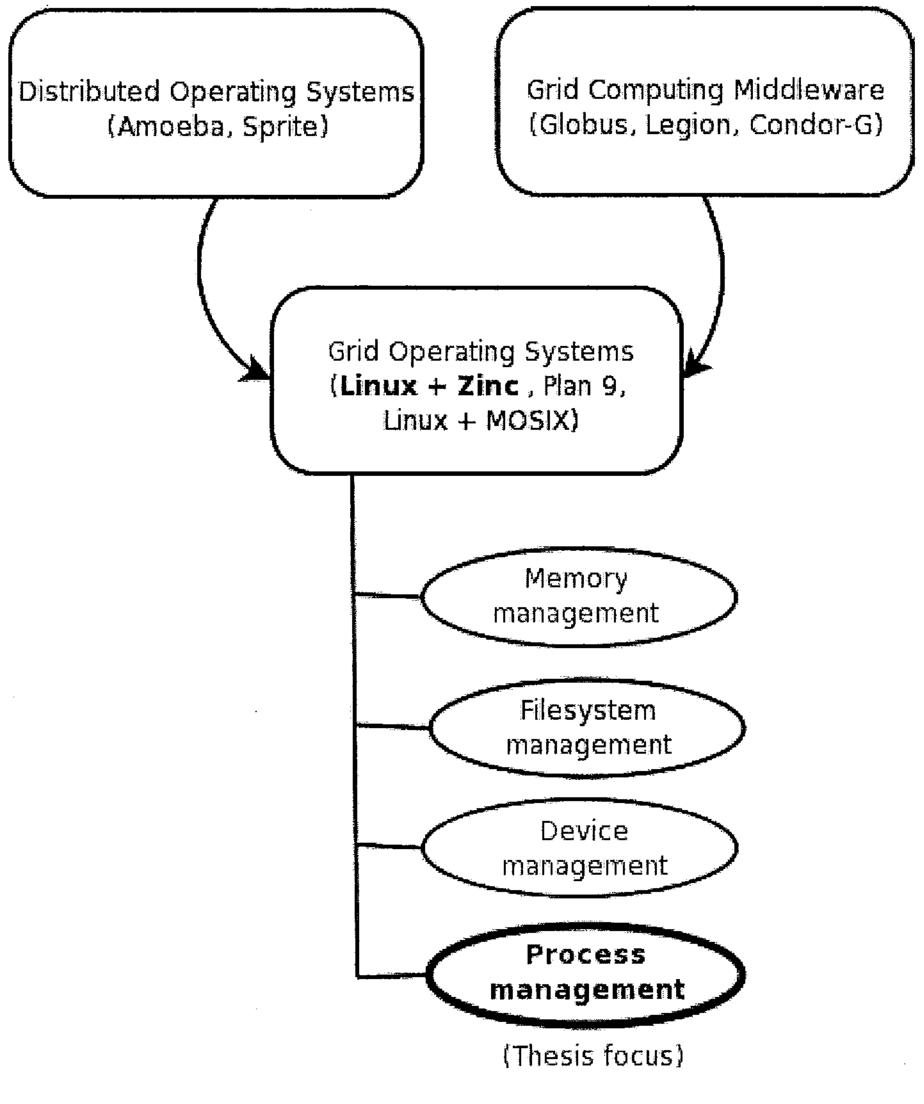
Distributed Operating Systems (Amoeba, Sprite)
Grid Computing Middleware (Globus, Legion, Condor-G)
Grid Operating Systems (Linux + Zinc , Plan 9, Linux + MOSIX)
Memory management
Filesystem management
Device management
Process management
(Thesis focus)

Figure 1.1: The Unification Of Operating Systems And Grid Computing

The rest of this thesis is organised as follows. Chapter 2 presents a survey and discussion of existing research related to our own. Chapter 3 gives a brief look at the design goals we have with our Zinc system, whereas the details of the design is found in Chapter 4. Chapter 5 describes the implementation of all the components of Zinc in depth. The experiments we carried out with our system and grid test-bed is presented in Chapter 6, together with the results and discussion. Chapter 7 provides a conclusion and summary of the research plus suggestions for future work.

CHAPTER 2

RELATED WORK

2.1 Introduction

In this chapter, we discuss the literature on existing work that is relevant to our research. Firstly, the properties of the grid environment is discussed in section 2.2. Then, in section 2.3 we try and define what a grid operating system is based on previous definitions of existing operating systems. Section 2.4 discusses an efficient resource discovery algorithm that we use for Zinc. Section 2.5 presents definition of grid scheduling while section 2.6 discusses distributed scheduling algorithms design choices. Next, section 2.7 surveys process migration techniques while section 2.8 discusses some existing systems which implement process migration in different ways. The emerging field of grid operating systems are discussed in 2.9 while a summary of the chapter is provided in 2.10.

2.2 The Grid Environment

In general, the computational grid comprises the following:

1. A set of resources which are shared to the users of the grid. Resources can mean any computational infrastructure, such as hardware like CPU, RAM, disk space, network bandwidth and software like databases, shared libraries and compilers.

2. Middleware to facilitate the coordinated sharing of all these resources to all the users. Grid software will automate the authentication of users, allocation of resources,

execution and monitoring of jobs, maintain the quality-of-service, throughput, and security of the entire system.

Grid computing shares some similarities with cluster computing. They both take advantage of the abundance of cheap hardware, and to some extent perform complex computational tasks in a collaborative manner between the different processing elements. However, there are a few differences between the two technologies:

1. Clusters are tightly coupled, with processing elements consisting of individual computers connected to each other via a high-speed networking interconnect such as fast Ethernet, Gigabit Ethernet or a specialised interconnect technology such as Myrinet. Grids are usually built on a bigger scale, encompassing distributed systems networked over wide distances such as LANs, WANs and the internet. The individual processing elements of a grid can be individual computers, mainframes or entire clusters.

2. Clusters are centrally administered, and its processing elements are physically located close to each other, usually in the same room. For grids, each processing element may be independently administered by different parties, and each component that comprises the grid may be located at geographically distant locations.

3. Clusters usually consist of homogeneous processing elements. Each node in a cluster (with perhaps the exception of the master node) have identical architecture, the same hardware and software configurations and usually cluster administrators try to set up a single system image with their clusters. Grids are usually comprised of different types of computers, storage devices, instruments and other networked gadgets, creating a heterogeneous computing environment for each of these machines and devices will have its own architecture, operating system, system libraries, and other features unique to each machine or device.

With these differences, there come certain implications that make software and algorithms suited for cluster-type operation unusable or inefficient on grid systems. The following factors have to be taken into account when constructing grid software:

1. The bandwidth and low latency readily available for cluster communication is not guaranteed on a grid. The more widely distributed the components on the grid, the more prone it is to suffer from bandwidth congestion, high latency and lag times, and other undesirable effects of wide area networking.

2. The possibility that processes and jobs may be shared by different computing facilities that are separately administered creates a problem of security. How will systems administrators authenticate and set permissions for tasks that do not originate from within their administrative control? How is trust established between different administrative domains? Different administrative domains also mean there is no guarantee of the immediate availability of resources, since one administrator has no control over the equipment administered by another party. Furthermore, if there is a hardware or software failure at a different administrative domain, there is nothing the local administrator or grid software can do to correct it. Unlike centrally administered cluster software, a designer of grid software must take all these issues into consideration.

3. The issue of heterogeneous architectures is the most problematic when designing grid software. Usually, programs compiled for a specific architecture cannot be run under normal circumstances on a different architecture. Even if the architectures are identical, it is seldom possible to run programs which are compiled for different families of operating systems such as Microsoft Windows and GNU/Linux. To get around this, users of heterogeneous systems standardise on a single portable bytecode-based programming language such as Java, Perl or Python.

These three points are an indication of a need for different approach to distributed computing when thinking about grids. The software and algorithms used for clusters cannot be totally reapplied without consideration for the preceding issues.

Most grid computing middleware is implemented as userspace daemons, libraries and programs. For example, Globus [21] is a collection of services that comprises a resource manager called GRAM (Globus Resource Allocation Manager), a communication library called Nexus, a directory service for state information called MDS (Metacomputing Directory Service), a remote data access service called GASS (Global Access To Secondary Storage), a monitoring tool called HBM (Heartbeat Monitor), a security and authentication framework called GSI (Globus Security Infrastructure) and an executable programs management service called GEM (Globus Executable Management).

Of each of these services, almost all of them introduce a set of APIs for the programmer to use when designing grid programs. The disadvantage of this approach is the introduction of complexity and cruft, especially for the user who needs to create new programs designed specifically for the grid, as well as users who wish to run their existing applications on the new grid environment. Furthermore, grid software currently available runs on top of existing operating systems that are completely unaware of the existence of grid users and grid processes that are executing on top of it. Therefore, the process management of the OS cannot schedule or handle these grid tasks in a way that would benefit the grid application. The emergence of these grid applications has created unique new requirements for process management, which most mainstream operating systems have no support for.

2.3 Operating Systems Support For Grid Computing

To argue the case for specific support for grid computing in operating systems, we shall briefly consider the different kinds of traditional operating systems and their different levels of support for networking and distributed resource sharing. Then we will identify another subclass of operating system which complements existing types of operating systems; the grid operating system.

2.3.1 The Centralised/Local Operating System

The centralised/local operating system represent a class of operating systems without network support and were common in the earlier days of computers before networking became popular. They can be single user or multiuser, but lack a network stack to communicate with other computers. The early versions of Unix were entirely centralised and local operating systems. A local operating system generally implements the four basic components of operating systems : file management, device management, memory management and process management.

2.3.2 The Network Operating System

The network operating system is an OS which implements a network stack, and implements several network services such as remote file or print servers. All modern operating systems such as Windows NT and Unix have networking functionality, and thus qualify as network operating systems. Unix and Unix-like operating systems like GNU/Linux usually come with many networking protocols, but the most popular of protocols in the Internet era is TCP/IP. Unix-like systems implement TCP sockets as an extension of the file and device management components; every network connection socket can be treated as a file to which we can read and write from.

The network operating system grew out of local operating systems which were given networking support. Therefore, they are generally not designed for running on a collection of computing elements which act as a cohesive whole. A distributed operating system is an OS for multiprocessing and multicomputing environments and is run as a single system image.Tanenbaum and van Renesse [75] outline several fundamental characteristics of the network operating system as opposed to the distributed operating system:

1. Each computer is running its own private operating system.

2. Each user logs in on each computer individually without a single sign-on nor a single system image which dynamically allocates CPU usage to the user from a pool of available CPU's.

3. File placement on different computers need to be managed manually; copying files from one machine to another requires manual network copy.

4. Very little fault tolerance; if some computers are out of commission, the users on that computer cannot continue working.

2.3.3 The Distributed Operating System

The key distinguishing characteristic of a distributed operating system from the network operating system is the transparency of its operation on multiple computers. The user should be able to see the distributed operating system as a single system image, where every computing resource is represented as part of a whole. The user should authenticate and log in only once, be able to access files on a local or remote machine anywhere in the system, run a process on any CPU, and the failure of a single component should not cripple the system. All resources, whether files or CPUs, must be able to be accessed with the same usage semantics regardless of the physical machine they reside on.

There have been several experimental distributed operating systems, such as Amoeba [73], Sprite [57] and Plan 9 [60]. Sadly, they have not gained widespread acceptance beyond the OS research community. Even so, modern network operating systems have come a long way since early network operating systems were introduced, and many of the distributed operating systems' features have been incorporated into them. Modern Unix-like systems can be equipped with NIS [34] or LDAP-based ([84], [80]) single system sign-on and authentication, and various distributed filesystems have been introduced such as NFS [65, 59] and Coda [66, 38]. Symmetric multiprocessor (SMP) and NUMA machines also incorporate dynamic CPU allocation across multiple CPUs. The Linux operating system supports all these technologies, and therefore is used on clustering and parallel processing platforms. However, the support for automatic distributed process management and CPU allocation outside of proprietary NUMA machines or SMP machines remains missing in mainstream GNU/Linux distributions.

2.3.4 The Grid Operating System

The current definitions of "local operating systems" and "network operating systems" are inadequate to describe an operating system with grid support. It might be convenient to group grid-enabled operating systems together in the "distributed operating system" category, but there are several aspects of grid computing which are not addressed by the definition of a distributed operating system.

A distributed operating system implies a single administrative domain (where a single party is responsible for controlling access, maintaining and granting permissions to users), whereas a grid can encompass many different administrative domains that want to pool their selected resources together. This has two consequences:

1. A grid can be a very decentralised entity with different authentication systems, different administrators and different geographical locations. A decentralised system defined by rules on the sharing of distributed computing resources is referred to as a *virtual organisation* [23]. Thus, though there is a mutual agreement of the sharing of a resource pool, not all resources belonging to each party are shared, and perhaps not all the time as well.

2. A distributed operating system assumes that all resources on the system can be allocated and scheduled with full authority. This is not the case in a grid resource pool which may encompass different administrative domains have their own allocation schemes and access control mechanisms which can not be overridden by another administrative domain. Hence, there is no central authority in grid systems.

According to Mirtchovski et. al [53], current operating systems such as Windows and the Unix variants were designed predating the advent of networking and the internet. Therefore, they are poorly suited for grid computing. Thus, a need for a grid operating system is a real one. Just as traditional operating systems simplified the usage of complicated assorted hardware resources by creating abstractions for them which a user can use transparently and easily, it is hoped grid operating systems will do the same for the eclectic mix of distributed resources on the grid.

2.4 Resource Discovery

The first step in the utilisation of the grid is resource discovery. In a fully decentralised distributed network, the task of querying a global state is done by first determining the existence of other nodes in the network. The process of each node on the network discovering other nodes that want to cooperate with it for distributed processing is the solution to what is known as the resource discovery problem. The system can be represented as a directed graph, where each vertex represents a node. A directed edge from node A to node B represents that node A knows about node B and B is said to be A's neighbour.

The resource discovery problem was described by Harchol-Balter *et al.* [29] in 1999. The model for resource discovery proposed had the nodes in the distributed system partaking in a series scheduling events at synchronous intervals of unspecified length called "rounds". In each round, each node would send a list of its neighbours in to other nodes, and a node receiving a list of neighbours would be able to create connections to new nodes previously unknown to it and add the new nodes as neighbours. After a certain number of rounds, the graph would be fully connected (all nodes know about every other node).

It is important that a robust resource discovery algorithm is used to give each node a picture of the global state of a distributed system. This information must be kept as current and as accurate as possible for each node. However, this must not be done at the expense of flooding the network, sending too many messages, or taking too long to complete. Harchol-Balter *et al.* outlined three metrics for evaluating the performance of resource discovery algorithms:

1. The number of rounds taken for the graph to reach full connectivity.
2. Pointer communication complexity – the number of "pointers" that is communicated during the course of the algorithm.
3. Connection communication complexity – the number of connections that are made during the course of the algorithm.

The Name Dropper algorithm was proposed by Harchol-Balter *et al.* in the same paper. It works as follows; consider a node v, where $\Gamma(v)$ is a set of all nodes which v knows about (the set of neighbours). In each round, each node v transmits $\Gamma(v)$ to *one* randomly chosen node u where $u \subset \Gamma(v)$. Upon receiving $\Gamma(v)$, node u will update its own neighbour list $\Gamma(u)$ with new information from v, i.e. $\Gamma(u) \leftarrow \Gamma(u) \cup \Gamma(v)$. The graph will achieve complete connectivity very quickly in $O(log^2 n))$ rounds with high probability, whereas pointer communication complexity is $O(n^2 log^2 n)$ and connection communication complexity is $O(nlog^2 n)$.

Name Dropper is similar to gossiping [30] algorithms which is used to broadcast information to a set of nodes. However, unlike gossiping, Name Dropper does not require each node to know about every other node in advance, nor does it require a fixed communications network.

Zinc uses Name Dropper to propagate resource discovery and state information updates across administrative domains. Due to its efficiency and simple implementation, Name Dropper performs very efficiently for fast information propagation.

2.5 Grid Scheduling Stages

According to Schopf, the scheduling of a job will go through the following stages in the grid[67]:

1. Resource discovery

   - Authorisation filtering – Restricting the search for resources that are only authorised to be used by the user.

   - Application definition – Defining the requirements of the user's job to select the appropriate resources

- Minimum requirements filtering – Eliminating the resources that do not meet the minimum requirements criteria from the set of resources to choose from.

2. System selection

   - Information gathering – Collecting state information from the grid, which is mainly derived from a Grid Information Service (GIS) and the local scheduler.
   - System selection – Deciding which system matches up with the requirements for the application. Examples of approaches for this are Condor Classads[62], multi-criteria [42, 41] and meta-heuristics [50].

3. Job execution

   - Advanced reservation – An optional step, users may opt to reserve resources in advance.
   - Job Submission – The user submits the job to the system. Currently there is no standardised way of doing this in the different grid middleware implementations.
   - Preparation tasks – Ensuring the files needed are in place, claiming a reservation, or whatever "preparation" steps needed to run the job.
   - Monitoring progress– Enabling the user to track the status of his or her job
   - Job completion – When the job is completed, the user is notified.
   - Clean-up tasks – Removing temporary files, retrieving data files, resetting configurations or other miscellaneous "clean-up" procedures.

Note that though Schopf calls the first stage "resource discovery", the sub-stages listed are have more to do with user authentication and user requirements collection than the actual "discovery" of distributed resources as described by Harchol-Balter in [29], which is categorised by Schopf into a subset of "system selection".