ISSN No. 0976-5697

Volume 4, No. 6, May 2013 (Special Issue)

International Journal of Advanced Research in Computer Science

REVIEW ARTICAL

Available Online at www.ijarcs.info

Grid Operating System: Making Dynamic Virtual Services in Organizations

Miss.Snehal S. Dhole
B.E-I.T (Final Year), Jawaharlal Darda Institute Of Engineering And Technology,
Yavatmal.(MS)INDIA
snehaldhole14@gmail.com

Prof. M. R. Shahade
Assistant Professor
Jawaharlal Darda Institute Of Engineering And Technology,
Yavatmal.(MS)INDIA

Abstract: The structure of a grid system should be such that even a small personal computer can avail the facility of many supercomputers at a time. The Grid computing has no straightforward way to control and administer grids dynamically. Grid operating systems bear the promise to become the new frontier in management of complex distributed computing systems and services that will offer for a single node: abstraction from hardware, and secure resource sharing with illusion dynamically by integrating grid capabilities into the kernel. It will integrate existing host operating system with a grid through an interoperating interface with expert dynamic OS on different versions of Grid virtual machine implementing grid nodes. Its goal is the creation of parallel processing pervasive grid computing platform that facilitates the rapid deployment and easy maintenance of grids of preferring peer to peer topology.

Keywords—Grid operating systems, distributed computing, host operating system, Expert dynamic OS, Grid virtual machine.

I. INTRODUCTION

Grid is a type of parallel and distributed system [1] that enables the sharing, selection, and aggregation of geographically distributed "autonomous" resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements Today Grid middleware is used to address the complexity of GRID environments and to help users in using GRID resources in an integrated way. This role in conventional computers is played by operating systems. Now it is time to develop a GRID operating system that may offer an integrated support for efficient management of local and remote resources available on a GRID environment to which a machine is connected. Without an operating system, Grids can fail the goal to enter mainstream computing and will not exploit all their functionality.

As a conventional operating system provides an abstraction layer on top of the underlying physical resources of a computer, a GRID operating system must be designed to provide a virtual machine interface layered over the distributed, heterogeneous, autonomous, and dynamically available resources that compose a GRID. Resource sharing is the main objective of Grids and operating systems is the more appropriate environment for providing GRID users access to resource sharing facilities in a secure and transparent way. In a shell environment with recovery facilities enables programs' interactions with the file system to be monitored very closely. Using this close monitoring, some Unix programs can be equipped with more sophisticated features. In this way, a make-like utility has been designed and implemented which provides automatic facilities in performing compilations of programs.

A. A GRID OS should:

a. Provide simple connection to the GRID, Tolerating node failures and allowing application checkpoint
b. Offer access to GRID resources, and Resource distribution transparency: Offering processes transparent access to all resources, and resource sharing between processes whatever the resource and process location.
c. Define policies for providing local resource to a GRID.
d. High performance; High availability.
e. Scalability: Dynamic system reconfiguration, node addition and eviction, transparently to applications.

Grid operating systems support properties and provide functionalities that are usually addressed at middleware level to enable seamless integration and management of distributed resources while providing a uniform interface to applications and services. We believe that the Grid infrastructure must absolutely reduce the burden on the application developer investing on the open source operating systems and extending them towards Grid, simplifying the life of the high-level Grid services implementers because they could rely on the native services of the operating system kernel for tasks such as resource or process management.

A Grid is assumed to be made of an uncountable number of computers that are called Grid nodes (or simply nodes). Grid OS aims to be a first step towards the creation of a true open source operating system for Grid platforms supporting distributed resources, by embedding some important basic services or functionalities directly into the operating system kernel Grid OS aims at making VO management easy for administrators and work within VOs easy for users. The cost of administering and operating a VO (e.g., adding or removing nodes, changing access policy, authenticating and authorizing users) should be minimized to a bounded value rather than simply increase with the number of users and resources participating in the VO. Deployment of Grids with existing Grid middleware [2, 3] involves the installation of multiple layers of software. Mathews et al [7] have highlighted similar issues. Multiple software layers in a Grid do not ensure fault tolerance. For example, with the popular cluster execution service Condor [8], a centralized cluster middleware can be liable to complete failure if a central server crashes. Active research is being pursued into more robust, flexible and fault tolerant Grid architectures, by converging Grid and Peer to Peer (P2P) topologies. However no Grid as of yet, has shown the advantages of such convergence.

A real operating system presents three principal interfaces to its users [6]: the virtual machine or operating system primitives accessible through programming languages; the utility programs such as compilers, linkers, and editors, and the command language or means by which users access system resources from a terminal. Most system services are available through one or more of these interfaces (see Figure 1). The idea of a virtual operating system is to provide standard versions of these interfaces, based on organizational requirements. Possible applications include data management environments, office information environments, real-time process control environments, and program development environments, to name a few.

Once the three interfaces are specified, implementation consists choosing one or more programming languages;

a) developing run-time libraries or extending the selected programming languages to support the chosen virtual machine on each target system;
b) implementing the utilities and command language in one or more of the selected programming languages, relying on the virtual machine to interface to the target operating systems;
c) Writing the necessary documentation.

A virtual operating system becomes a real operating system when the associated virtual machine corresponds to a physical machine. However, the emphasis in building a virtual operating system is on the interface presented to the user. The virtual machine is a highly idealized set of primitive functions geared to organizational programming requirements. It bears almost no functional resemblance to the underlying hardware which actually performs the work. In general, a virtual operating system is restricted to those system layers. Installation consists of interfacing the standardized virtual machine to the vendor supplied system. parts of an ordinary operating system which an organization considers important in completing its work. Obviously, a single real operating system can support many virtual operating systems. To achieve the full benefits of this approach, the virtual machine must be implementable without changing the vendor software. This implies a functional equivalence between the chosen virtual machine and the target systems. A bootstrapping design procedure is therefore required. Every candidate virtual machine function must be tested on each target system before it can be adopted. The virtual operating system approach reduces the problem of moving to a new system to the (nontrivial) problem of implementing a virtual machine.

All utilities and user utility.. Finally, command language procedures are also portable, since the command language program is portable. The availability of the entire virtual operating system (virtual machine, utilities, an command language) makes it easy for users and programs to move from one vendor system to another. We emphasize that this approach reduces the cost of moving both people and software to zero. The overhead is the cost of implementing the virtual machine on the candidate sys-Communications of the ACM programs are completely portable since their interface to any particular operating system is through the virtual machine. Similarly, higher level procedures written for a portable utility are themselves portable. For example, a file containing editor commands will work on any machine supporting the editor.

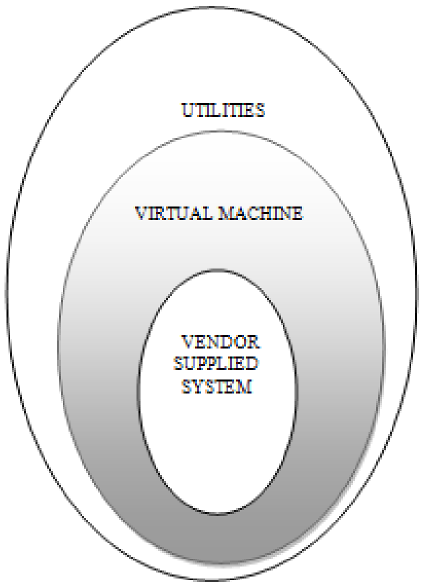

Figure: 1. A virtual operating system provides standardized versions of the three outermost system layers. Installation consists of interfacing the standardized virtual machine to the vendor supplied system

II. LITERATURE REVIEW

A. *Virtual services in organizations:*

A virtual service can be seen as a temporary or permanent coalition of geographically dispersed entities (individuals, groups, organizational units or entire organizations) that pool resources, capabilities and information to achieve common objectives. There usually will be legal or contractual arrangements between the entities. The resources can be physical equipment such as computing or other facilities, or other capabilities such as knowledge, information or data. In an organization, information is stored and services and applications are executed by a set of computers in a Grid. Key components of a virtual services are an administrator of the organizations, who is authorized to manage VO membership and policies, a set of participating users (called Grid users) in different participating domains, a set of roles which users/resources can play in the VO, a set of rules/policies on resource availability and access control, an (renewable) expiry time of the VO. The main responsibilities of node-level management include: translating from grid identities into local identities; granting or denying access to resources, checking limitations of resource usage (CPU wall time, disk quotas, memory, etc.); protecting and separating of resource usage by different users; logging and auditing of resource usage, etc.. A VO and its implementation by an operating system can reside in several stages of VO lifecycle: VO identification, VO formation, VO operation, VO evolution, and VO dissolution. In each stage a set of security threats to the overall system exists.

Grid OS, that is to say, the operating system is fully Grid-enabled. Once the Grid OS system has been installed on a machine, this machine is ready to participate in a VO with no need to install additional system software. Modifications to Linux to natively support VOs are done with a careful design to keep backward compatibility while providing build-in VO management interfaces [4, 5] that are as secure and simple to use as possible. System services and utilities such as login and shell programs, together with libraries, are extended in a modular approach so as to favor VO-level resource sharing requirements while keeping maximal transparency to users.

Access security in Grid OS will be policy driven. This means that for each resource (which includes VOs, applications, hosts, etc., in fact anything that requires protection) there will be a policy specifying who can access it and what they can do with it. In the case of a resource such as a file, the who could be a list of individuals and/or VOs, and the what could be read, write or execute actions similar to the conventional Linux file permissions (with a VO being considered as a sort of group). However, in a distributed and VO-based environment access will typically involve more than one entity, each with its own policies. The idea to monitor the operating system running on a PC is to execute the backdoor and the monitored OS in different virtual machines on top of a virtual machine monitor. The main issue to be tackled in the implementation is the extraction of OS state from the memory.

a. Application Management:

As all layers will be integrated, the system will be able to offer information about the progress of the job, accurate monitoring of the used resources, error information, etc. In the current Grid world, given that the managers for the different layers are not integrated, a lot of information is lost in the way and the one that survives it is not correlated making it very difficult to use. For instance, in current Grid systems it is difficult to know why an application failed, when and with exactly what resources it run, etc. The integration of all services in a single OS will remove the lack of integration and offer users an execution environment with plenty of monitoring information and a powerful control of execution.

As the computational system are very large number in nature so it is planned in the present work to allocate the type of programming in a particular node hence when a user desires to avail the grid facility; the host local OS should handover the problem to the expert dynamic OS when software is loaded. The other types of a program which is complex in nature and requires the participation of many nodes . The host local OS computer evaluates the problem and transfers the modules to the participated computers. The third types of software used to such that it is divided in modules equal to the number of different grid OS and all the participating computers processing paralleled, then the responses of each computer are integrated in the host local OS node and which transfers the result to the originating PC interface with expert dynamic OS. However, when multiple users launch applications on the same cluster, it may happen that the workload exceeds the cluster capacity. To avoid this situation, a solution is to execute a batch system on top of the grid operating system. When an application is launched with the fork-delay capability enabled, its processes are queued if the cluster is overloaded. When a process terminates its execution, the global scheduler resumes the execution of the delayed processes, if any. At any time, if the cluster load is too high, the global scheduler may decide and use grid stacks that only suspend the execution of a very few or no processes.

b. Data Management:

It should support extended meta-data, hierarchical names (the traditional directory structure), private, shared and collaboration data, and data archives. It should also support named Grid pipes, used by workflows where different processes produce data and some others consume it, the various processes being located on different nodes. Access rights should be managed in a manner such that file access could be granted to Grid users according to VO policies.

III. PROPOSED WORK

Super peer paradigms have recently gained popularity as they enable Grids to integrate some of the advantages of peer to peer systems making a Grid infrastructure more robust, scalable and fault tolerant [14]. The toolkits require a common set of services from the underlying operating system. The key principle in Grid OS is to provide modularity. The modules provide a policy-free API which can be used to develop high level services like Grid FTP. Grid OS provides a basic set of abstract dynamic services that are common to prevalent Grid software infrastructures with minimal Core Operating System Changes. Architectural components of Grid OS are designed to be self configuring and plug-and-play in order to facilitate the rapid deployment of a Grid e.g. adding a node to a site involves a simple sign-in peer, adding a site to a remote region, involves registration process with a remote peer. The OS can be seen from two perspectives: First an integrated Grid Stack allowing rapid deployment of Grids, whilst making administration of Grids in an operating system which provides built in support for Grid computing. There is overlap between both kernel and use modes.

If an organization chooses to use the stack configuration, they can easily unload the kernel space modifications and use Grid computing from a user and middleware level. The lowest layer in Grid OS is the kernel layer and includes modules which facilitate Grid enabling of interactive application and fine resource management in Grids. Grid OS however makes use of process migration which transfers the execution context of processes to nodes where enhanced processing capabilities are available. Process migrations allow the transparent Grid enabling of existing applications without any need to modify them.

Support for dynamic virtualization [15, 16] is another central feature of the kernel layer with using expert dynamic OS. Grid OS aims to investigate hardware based virtualization in order to use a virtualization engine using fuzzy logic algorithm which enables the rapid creation and destruction of on-demand virtual machines. Both the Quos Management and kernel level process check pointing modules allow users to regulate resource usage of applications and to autonomously migrate them to different nodes within a site.

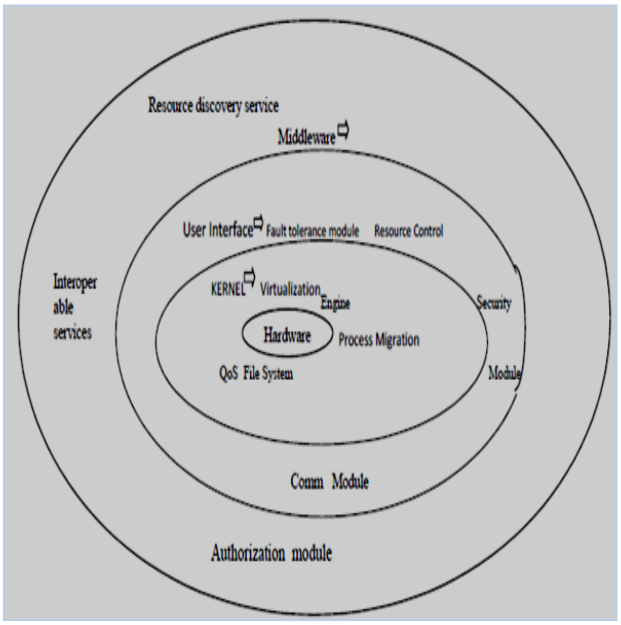

Figure: 2 Grid OS Architecture

A. *What do you mean by the very term shell?*

A shell is a program which provides a user interface to an operating system. It may be a text-based command line interpreter as in Unix, or it may be a graphics-based and process manager. Either way, the ability to repair damage to permanent resources such as les is an important one. The facilities which are typically provided at present are rather primitive, consisting of a waste bin" directory where old versions of les are stored when explicitly deleted, together with various ad hoc backup mechanisms provided by individual applications. It is ironic that one can always undo the deletion of a single character in an editor, but not the deletion of a permanent file in a shell.

The aim of this thesis is to describe a way of designing and implementing .The aim of this thesis is to describe a way of designing and implementing a more intelligent shell which keeps track of versions of les on behalf of the user, together with information about how they were created or manipulated. This enables it to provide a more uniform and consistent mechanism for undoing the effects of commands, recovering old versions of files, repairing accidental damage, and otherwise managing a user's most permanent and valuable resources in a safe and convenient way. Besides, such a shell creates a sophisticated working environment in operating systems. A user can run programs under ultimate control, monitoring their interactions with the file system. The control of file system activities of user programs allows many new applications to be developed and even existing ones to be upgraded with new facilities. For example, the Unix make program can be enhanced easily, whose design and implementation is the additional scope of the thesis.

A. *Shell Features:*

As mentioned in the previous section, shells offer features geared specifically for interactive use rather than to augment the programming language. A shell can provide users with one or more of the following features. Users can

a. create an environment that meets their needs,
b. write shell scripts,
c. define command aliases,
d. edit the command line.
e. manipulate the command history,
f. automatically complete the command line,
g. run lengthy tasks in the background,
h. store data in user-defined or shell-defined variables,
i. link any number of commands together (piping),
j. redirect program input and output.

These features are the primary advantages of interfacing to the system through a shell. Unfortunately, shells don't have a standard way of providing them. A single feature (or capability) can be provided in different ways by different shells, which destroys compatibility. A script file written in a shell's programming language, for instance, may not run in another shell as a result of the syntax variation.

B. *Shell Implementation:*

The Shell is the module of brush that interacts with the user directly and initiates the execution of user-oriented tasks. It is possible to use one of the existing shells to carry out the function of the module. This section discusses the usability of an existing shell that replaces the Shell and presents an implementation of such a use.

C. *Developing a Shell:*

In shell-based operating systems, shells are commonly used as command interpreters. A shell is a user interface that allows users to access operating system resources by running various commands. Likewise, brush is required to provide a new shell which cooperates with the Recovery Manager during both execution and undoing/redoing of commands. With the new shell, user commands, except undo and redo commands, are supposed to be interpreted in similar way to those of an ordinary shell.

IV. CONCLUSION AND FUTURE WORK

Grid operating system which provides extensive, flexible services for Grid architectures and it also has planned to port Globus libraries to Grid OS thus providing a complete software infrastructure for Grid architectures. Grid OS is not only aimed at adapting dynamic grid computing for frequently related to the set up and administration of Grids but also it is based on dynamic virtualization engine for Grid OS to provide security and resource management to resource owners and privacy to resource users. Using the virtual operating system approach, uniformity can be achieved at the three principal levels of user interface-- the virtual machine, the system utilities, and the command language. For at least one realization of the virtual machine interface, the functional equivalence of vendor operating systems has been established.. Although the effort to install a virtual operating system is large when compared to the effort required when moving a single program, it is small when compared to the cost of moving an organization's software. Moreover, when personnel retraining costs are considered, installation costs are insignificant. The creation of Grid applications and the lack of general fault tolerance within the Grid infrastructure are also issues of concern. Grid OS is a step towards a “Plug and Play” pervasive Grid dynamic computing environment. It is designed to support all types of modern computations, including batch and interactive and dynamic support the creation of Grids of any architecture. The main contribution of this paper is that it presents a dynamic structure for the development of adaptive Grid OS to extend the discovery service to enable self-healing and self organizing behavior. Furthermore we propose that the system should embed the capability for interoperability with existing and emerging Grid infrastructures interface with expert dynamic OS interact with lower host layer local OS by making the system compliant to evolving standards in Grid computing.

V. REFERENCE

[1]. A. Iamnitchi and D. Talia, "P2P Computing and Interaction with Grids", Future Generation Computer Systems, North-Holland, vol. 21, no.3, pp. 331-332, 2005.

[2]. I. Foster, C. Kesselman., “Globus: A Metacomputing Infrastructure Toolkit”, Intl J. Supercomputer Applications, 11(2):115-128, 1997

[3]. E. Laure et al., Middleware for the Next Generation grid Infrastructure, Proceedings of the Computing in High Energy Physics Conference, pages 826, 2004.

[4]. American National Standard FORTRAN. ANS X3.9-1966, Amer. Nat. Standards Inst., N.Y., 1966. Contains the official description of the programming language Fortran 66.

[5]. Snow, C.R. The software tools project. Software Practice and Experience 8, 5 Sept.-Oct. (1978), 585-599. Describes an implementation project on a Burroughs B 1700 computer using an automatic code translation technique 12. Wulf, W.A., Russell, D.B., and Habermann,

[6]. A.N. BLISS: A language for systems programming. Comm. ACM 14, 12 (Dec. 1971), 780-790. Describes BLISS, a language designed to be especially suitable for use in writing production software systems for DEC machine.

[7]. Brinch Hansen, P. Operating System Principles. Prentice-Hall, Englewood Cliffs, N.J., 1973. Designed for readers with a background in programming and a knowledge of elementary calculus and probability theory, focuses on general concepts illustrated with algorithms, techniques, and performance figures.

[8]. B. Mathews, “Towards a Knowledge Grid: Requirements for a Grid OS to support Next Generation Grids”, Core Grid Workshop on NGN, Belgium, 2005

[9]. Litzkow, M. Livny, & M. Mutka, Condor – A Hunter of Idle Workstations, Proceedings of the 8th Int. Conference of Distributed Computing Systems, June 1988, pages 104-111.

[10]. A. Iamnitchi, I. Foster, J. Weglarz, J. Nabrzyski, J. Schopf, M. Stroinski, in: Grid Resource Management (ed.), A Peer-to-Peer Approach to Resource Location in Grid Environments, Kluwer Publishing, 2003.

[11]. Barham, P. et al. “Xen and the art of virtualization”. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (Bolton Landing, NY, USA, October 19 - 22, 2003) SOSP '03. ACM Press, New York, NY, 164-177

[12]. Kernel Virtual Machine,http://kvm.qumranet.com/kvmwik

[13]. A book on A Sophisticated Shell Enviroment by Hauseyin Pehlivan submitted to the university of Bristol in the faculty of Engineering, Computer Science Department.