WebCom-G: Grid Enabled Metacomputing

John P. Morrison, Brian Clayton, David A. Power, Adarsh Patil
Centre for Unified Computing,
Dept. Computer Science,
University College, Cork,
Ireland

Abstract

Current Grid enabling technologies consist of stand-alone architectures. A typical architecture provides middleware access to various services at different hierarchical levels. Services exposed at these levels may be leveraged by the application programmer. However, the level at which the service appears in the hierarchy determines both its richness and the complexity of its use. Thus, benefits gained by using these services are defined by the manner in which they are accessed.

Generally, choosing to use a particular middleware service inclines the application programmer to use the associated middleware suite as it is difficult to cherry-pick services across middlewares. Interoperability and independent service access are not easily facilitated in current middlewares.

WebCom-G is a fledgling Grid Operating System, designed to provide independent service access through interoperability with existing middlewares.

This paper presents an overview of the WebCom-G Operating System and describes the the WebCom-G Information System and WebCom-G's mechanism for executing Globus tasks.

Keywords - WebCom-G, Grid, Middleware, Interoperability, Co-Existence

1 INTRODUCTION

The computing power available in current desktop machines is equivalent to or supersedes that of past high-performance computers(HPCs). Together with advances in networking, HPCs can be built by harnessing the computing power of commodity computers distributed around the world. These HPCs are typically owned by multiple heterogeneous organisations and Grids are constructed from the amalgamation, sharing and selection of networked services by these organisations. Such services are made available under common sharing and security policies. In a typical Grid middleware, these services are served through the Grid Information Services[5, 10]. Once these services are known, they are aggregated and presented to the application developer

by the Grid resource broker. The application developer must possess a knowledge of both the middleware being used and the underlying computational hardware. Using this information, task dependent libraries and binaries can be produced. These are typically managed by the user, who also has to possess some knowledge of the target architecture. This makes the process of application deployment both time consuming and error prone.

Figure 1: *An overview of the various grid middlewares*

Much research is focused on exploiting the computing resources of geographically distributed volunteers using the Internet and user-level middlewares such as SETI@Home and Distributed.Net. Other approaches, such as Globus[37, 11, 12], Legion[15, 36], JXTA[32], Hydra[3] etc, exploit core level middleware technologies. These provide a fundamental software infrastructure to build Grid technologies that aim to find ways to make computing easier, faster, and more accessible to programmers and users. Some of these core middleware technologies provide a *"bag of services"*[37], toolkits or integrated service architectures. Other projects: Condor-G[7], PBS[31], LSF[30] provide independent job managers that interact with these core middleware services.

Fig. 1 shows an overview of various Grid middlewares, from vertically integrated solutions, such as Sun Grid Engine and Legion, to service oriented architectures, like Globus, to others such as SETI@home and peer to peer systems.

The WebCom metacomputer[26] is an application execution environment operating across the Internet or on intranets. It provides both an execution platform, and a development platform. Applications are specified as Condensed Graphs[21, 24, 27], in a manner which is independent of the execution architecture, and this platform independence facilitates computation in heterogeneous environments.

In addition to its expressive programming model, WebCom automatically handles task synchronisation, load balancing[25], fault tolerance[25], and task allocation without the need for these decisions to be propagated to the application developer. These characteristics, together with the ability of the CG model to mix evaluation strategies to match the characteristics of geographically dispersed facilities and overall problem-solving environment, make WebCom a promising Grid middleware candidate[22].

The remainder of this article is broken up as follows: Section 2 describes the WebCom Metacomputer. Section 3 shows how WebCom Middleware support is added to marshal Globus. Section 4 extends the WebCom view to that of providing a novel Grid Operating system called WebCom-G OS. Section 5 describes WebCom-G's Information System and finally Section 6 outlines some conclusions and future work.

2 WEBCOM METACOMPUTER

Metacomputing systems were developed to harness the power of geographically distributed computing resources. Such resources generally consisted of machines connected to intranets, the Internet and World Wide Web. Different projects in this area of computing range from those harnessing the power of closely coupled networks of workstations, such as Cilk-NOW[33, 17] and Treadmarks[8], to projects such as Charlotte[1, 14], Bayanihan[16, 34, 9], Javelin[4] which utilize the processing power of the Internet.

These systems typically use the server/client model for task distribution. Clients normally consist of stand alone applications, or Java applets with different applets being used for different tasks. The stand-alone application client communicates with the server via proprietary mechanisms, while applet based clients typically communicate using Remote Method Invocation(RMI)[20], Object Request Brokers (ORB's)[29] or Object Serialization.

Distributed applications are typically constructed by using Application Programming Interfaces (API's) provided by the chosen system. Here the onus falls on the programmer to explicitly determine the parallelism of the problem as well as having to implement fault tolerance, load balancing and scheduling algorithms.

WebCom separates the application and execution environments by providing both an execution platform, and a development platform. Applications are specified as Condensed Graphs, in a manner which is independent of the execution architecture. The independence provided by separating these two environments facilitates computation in heterogeneous environments; the same Condensed Graphs programs run without change on a range of implementation platforms from silicon based Field Programmable Gate Arrays[28] to the WebCom metacomputer. Fault tolerance, load balancing, scheduling and exploitation of available parallelism are handled implicitly by WebCom without explicit programmer intervention.

WebCom uses a server/client model for task distribution. Clients consist of Abstract Machines(AM's) that can be either pre-installed or downloaded dynamically form a WebCom server. AM's are uniquely comprised of both volunteers and conscripts. Volunteers donate compute cycles by instantiating a web based connection to a WebCom server and dynamically downloading the client abstract machine. These clients, constrained to run in the browsers sandbox, will execute tasks on behalf of the server. Task communication is carried out over dedicated sockets. Pre-installed clients, also communicate over dedicated sockets. Upon receipt of a task representing a Condensed Graph (the task can be partitioned for further distributed execution), such clients are promoted to act as other WebCom servers. The returning of a result causes a promoted AM to be demoted, and act as a simple client once more.

The execution platform consists of a network of dynamically managed machines, each running WebCom. WebCom can assume a traditional client server connection model or the more contemporary peer to peer model, Fig. 2.

WebCom sees each abstract machine as a "unit". Each unit contains a number of modules. The modules include an execution engine module and others for communication, load balancing, fault tolerance, scheduling and security, Fig. 3. These modules are plugins to a backplane. Communications between WebCom units use a messaging system. Plugins can send messages between themselves on the local unit
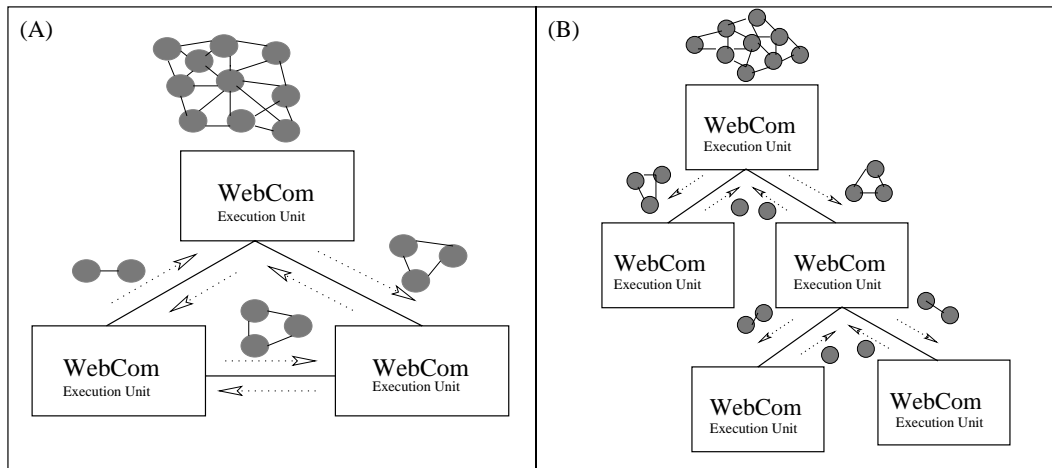
Figure 2: *WebCom can be configured as both a peer to peer hierarchy (A) and a traditional server client hierarchy (B). Each unit can receive instructions that comprise of either Condensed Graphs or "atomic" instructions. Atomic instructions are generally value transforming instructions and can be of arbitrary grainsize.*

or to any plugin on other connected units. The default engine module is capable of executing Condensed Graphs applications.
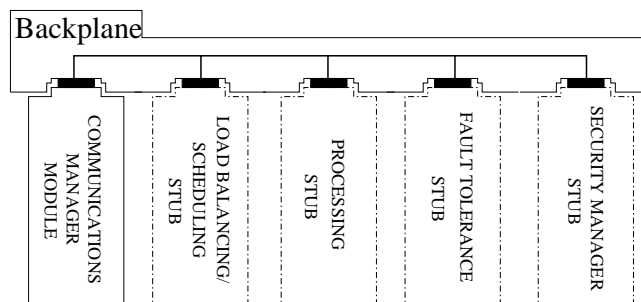

Figure 3: *A WebCom Unit consists of a collection of plugins. One each for the fault tolerance, load balancing, compute engine, connection manager, security and scheduling. The backplane is used to bootstrap the whole system and handles communication between the plugins.*

A WebCom execution is initiated by a user on a single unit launching a Condensed Graph application. As nodes become available for execution they are formed into messages and passed to the scheduler. The scheduler, in conjunction with the load balancing, fault tolerance, communications and security plugins decide where the node is to be executed. If it is to be executed locally it is passed back to the engine. For remote execution, information about the unit selected to execute the node is incorporated into the message, and the message is then passed to the communications manager plugin for distribution to the selected unit.

Once a node has completed its operation, a result message is created and returned to the unit where the corresponding instruction originated. This unit then incorporates the returned result into its graph and the execution proceeds. If at any time an executing node fails to complete its task, the fault tolerance plugin will cause the

node to be rescheduled to an alternative compatible unit. If no such unit is available, the node is retained for subsequent assignment.

WebCom possesses a node targeting mechanism. For example, if a particular node represents a COM instruction, that instruction is sent to a WebCom unit with a COM engine plugin capable of executing the instruction. This mechanism is the same for other technologies such as DCOM, EJB and Corba. One implementation of this system (Anyware) was used to create enterprise wide applications using middleware integration. This philosophy can be applied at higher levels of abstraction.

3   WEBCOM MIDDLEWARE

The WebCom metacomputer described in Section 2 provided the basis for the unification of existing midlewares such as Corba, EJB, COM, DCOM. Interaction with each middleware is handled through an appropriate compute engine plugin. Web-Com provides a general plugin interface. By adhering to this interface plugins can be easily constructed for each of the modules outlined previously. Specific compute engine plugins can be created to leverage existing Grid technologies, such as Globus. The remainder of this section briefly describes task execution within the Globus environment and proceeds on to detail WebCom's marshaling of Globus tasks.

3.1   Globus Execution

Globus provides the basic software infrastructure to build and maintain the Grid. As dynamic networked resources are widely spread across the world, information services play a vital role in providing Grid software infrastructure, discovering and monitoring resources for planning, developing and adopting applications. A crucial part of every Grid is the information services. The onus is on the information service to support the initial discovery and subsequent use of resources and services.

An organisation running Globus will host it resources in the Grid Information Service (GIS) which is running on its gatekeeper machine constituting the organisations entry point to the grid.

The GIS will typically contain both static and dynamic information reflecting the transient and heterogeneous nature of the grid platform. Static informaton includes datails of numbers and configurations of compute nodes. Dynamic features of the grid are captured in machine loads, storage and network information. Machines running Globus may use a default scheduler or more advanced schedulers such as those provided by Condor, LSF, PBS and Sun Grid Engine.

Typically, users authenticated via the Grid Security Infrastructure (GSI) create a Resource Specification Language (RSL) script representing the job they wish to execute on Globus. This script is then executed on the Gatekeeper machine, specifying the application to run and the physical node(s) the application should be executed on and any other relevant information. The gatekeeper contacts a job manager service which in turn decides where the application is executed. For distributed services, the job manager negotiates with the Dynamically Updated Request Online Co-allocator (DUROC) to find the location of the requested service. DUROC makes the decision as to where the application is executed by communicating with each machine's lower level Grid Resource Allocation Manager (GRAM). This information is communicated back
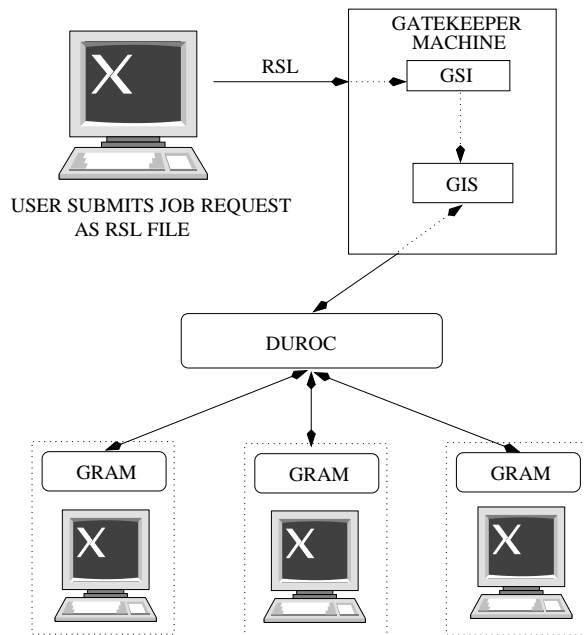
Figure 4: *Globus Execution model. A user generates an RSL script and submits it to the gatekeeper. The gatekeeper, in conjunction with DUROC and GRAM facilitate the distributed execution of requested services on the underlying nodes.*

to the job manager. The job manager will schedule the execution of the application according to its own policies. If no job manager is specified, then a default service is used. This is usually the "fork" command, which returns immediately. A typical grid configuration is shown in Fig. 4.

There are some disadvantages to using RSL scripts. Most notably, in a distributed execution if any node fails, the whole job fails and will have to be re-submitted by the user at a later time. Furthermore, there is no diagnostic information available to determine the cause of failure. Only resources known at execution time may be employed. There is no mechanism to facilitate job-resource dependencies. The resource must be available before the job is run, otherwise it fails. Within Globus, there is no checkpointing support, although some can be included programatically or by specifying an appropriate job manager. This may not be feasible due to the particular grid configuration used. Also, RSL is only suited to tightly coupled nodes, with permanent availability. If any of the required nodes are off-line, the job will fail.

3.2   Marshaling Globus

Due to the highly dynamic nature of resource availability of Globus, it is possible that a job is scheduled with the required resources, but at execution time some resources may no longer be available. This may be caused by hardware failure for example. This unavailability of resources will cause the job to fail. Thus, the scheduling of a large number of jobs to a remote site becomes hard to achieve successfully.

By including jobs as nodes in a Condensed Graph application, WebCom's node targeting mechanism can be used to send them to any existing job manager. Furthermore, the actual job configuration becomes dynamic: the complete configuration may be generated at runtime as a result of the execution of other nodes in the associated
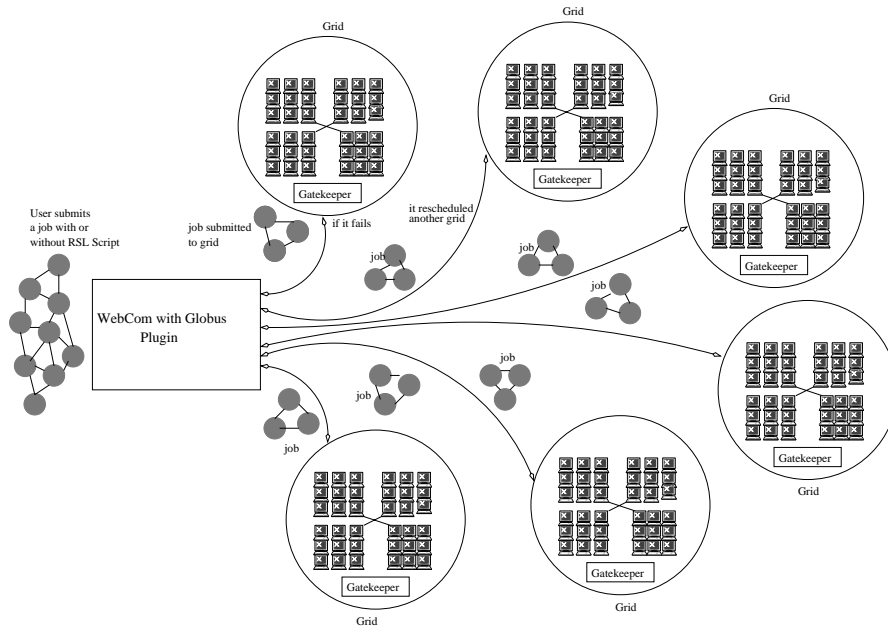
Figure 5: *WebCom-G: WebCom with a Globus/Grid plugin facilitates the execution of grid applications from within a condensed graph. Failures are detected and rescheduled using WebCom's fault tolerance and load balancing strategies.*

Condensed Graph, Fig. 5. Even the name of the job could be dynamically uncovered. WebCom can interrogate the targeted grid to ensure the required resources are available before the node is sent for execution, or it may request the targeted grid to notify it when the required resources are ready. Alternatively, the node can be sent and put in a wait state until all resources are available. Using WebCom to schedule grid jobs in this manner also allows the utilisation of it's fault tolerance mechanisms[23]. This is used to reschedule any applications targeted to the grid that may have failed. Therefore, once WebCom initiates a job it will complete provided the resources eventually become available.

By using WebCom, a whole grid can be viewed as a single WebCom unit with a specific computation engine plugin. This evolution of WebCom is the first step of producing a grid-enabled middleware: WebCom-G.

Multiple grids are themselves viewed as independent WebCom-G units. When an instruction is sent to WebCom-G all the information is supplied to either dynamically create and invoke an RSL script or to execute the job directly.

When a WebCom-G unit receives an instruction it is passed to the grid engine module. This module unwraps the instruction, creates the RSL script and directs the gatekeeper to execute it. Once the Gatekeeper has completed execution, the result is passed back to the unit that generated the instruction, Fig. 6. As WebCom-G uses the underlying grid architectures, failures are detected only at the higher level. In this case WebCom-G's fault tolerance will cause the complete job to be re scheduled.

A WebCom compute engine plugin is implemented as a module that interacts with WebCom via a well defined interface. This gives a platform independent grid compute engine. Although a Condensed Graph may be developed on a platform that is not grid enabled, the execution of grid operations will be targeted to grid enabled platforms.
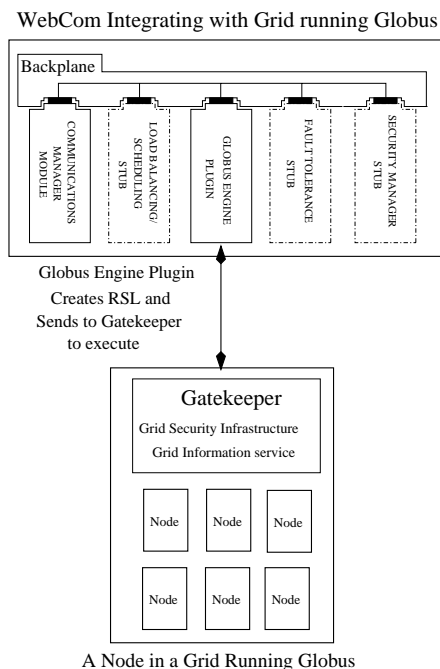
Figure 6: *WebCom-G: The Condensed Graphs compute engine plugin is replaced by the grid compute engine plugin. A targeted grid instruction is received by the compute engine. This dynamically creates the RSL file. The compute engine then causes the RSL script to be executed on the gatekeeper.*

4   WEBCOM-G OS

The WebCom-G Operating System is proposed as a Grid Operating System. It is modular and constructed around a WebCom kernel, offering a rich suite of features to enable the Grid. It will utilise the tested benefits of the WebCom metacomputer and will leverage existing grid technologies such as Globus and MPI. The aim of the WebCom-G OS is to hide the low level details from the programmer while providing the benefits of distributed computing.

Resources in a heterogeneous environment like the Grid join and leave at times of their choosing. Gathering information and analyzing resources is important for resource management, scheduling, load balancing and fault tolerance. To build strict models (process management models, programming models, service oriented models) for quality of service, a proper Grid Operating System is needed. As with most operating systems there will be various components to provide the required functionality.

This section describes how WebCom-G retrieves resource information from a heterogeneous environment and how it will use this information in conjunction with its resource management techniques to provide a home for different models which promise Quality of Service. Different methods for gathering and analyzing resource information to improve the Quality of Service requirement are discussed. A statistics analyser is introduced, which forms the basis for evaluating the costs of executing jobs on the grid. Finally, a comparison with Globus approach to static and dynamic information gathering is made.

The WebCom-G Operating System (Fig. 7) is designed to operate between the

system hardware and any installed grid middleware. It will interoperate with Globus (Versions 2.4 & 3) and with other middlewares such as PVM[13, 35], MPI[18, 19] and will work with standalone products such as Distributed.Net[6] and Seti@Home [2] clients.

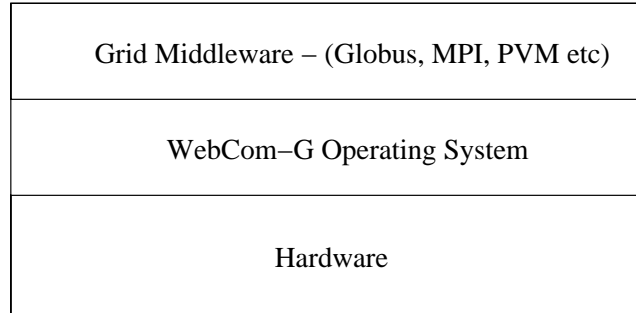| Grid Middleware – (Globus, MPI, PVM etc) |
| WebCom–G Operating System |
| Hardware |

Figure 7: *WebCom-G extends the operating system functionality of the native hardware, interfacing with traditional Grid middleware.*

The WebCom-G Operating System, illustrated in Fig. 8, is designed to be modular. This design allows WebCom-G to be used in a number of different contexts – e.g., where WebCom-G is the only grid middleware installed, or to co-exist with say Globus, or MPI or both. If the system is configured to consist of multiple middlewares, each with its own information provider service, WebCom-G will automatically choose between them based on specific requirements of the application. However, the decision can be overridden by the programmer. WebCom-G will be able to treat different middlewares and users as Vitrual Organisations, giving it control over task priorities.

4.1 Components of the WebCom-G OS

Economy Status Analyser
The Economy Status Analyzer is a plug-in which gets the overall resource status of the machines comprising the grid from either the stats daemon or the information manager in a WebCom-G system or from the WIS if other information gathering middlewares (such as GRIS & GIIS & Ganglia in case of Globus) are present. This module will use various algorithms to evaluate independent cost and total cost of utilization. Scheduling compute intensive, time critical and data intensive jobs depends on accurate and timely updates of resources. Based on this inforamtion, the ESA calculates the cost of executing the job. The quality of this service depends critically on the reported status information.

  Stats Daemon
The Stats Daemon is a Linux daemon or Windows service, and uses standard system calls to retrieve system information - the Linux `sysinfo` kernel command or standard MFC function calls on Windows. It logs system usage to the hard drive, using one of several strategies e.g., fifo fixed filesize logging.

  WebCom-G Information Module
The WebCom-G Information System (WIS) is the information gathering module within the WebCom-G OS. It consists of three parts, a low-level stats daemon to run
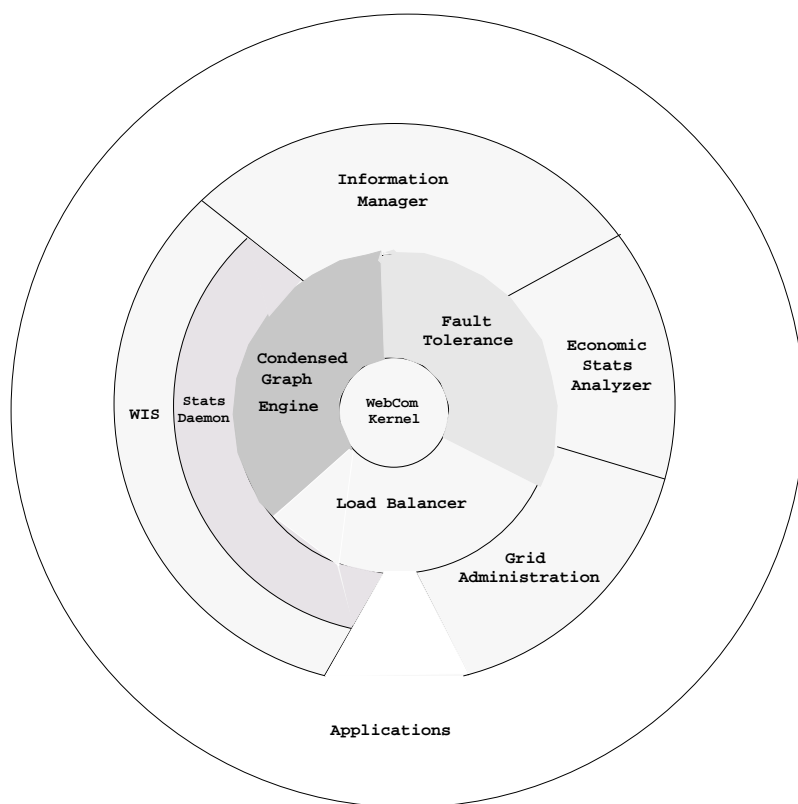
Figure 8: *WebCom-G OS architecture*

directly on the hardware, a higher-level Information Manager, which can be aware of multiple hosts, and the WIS proper, which is capable of communicating with a single stats daemon, with an Information Manager or with Globus via the GIIS.

    Grid Administration

The Grid Administration tool will allow administration by user or middleware. It will be possible to dynamically renice processes when unfair CPU allocation occurs, or to give priorities to particular jobs. Processes launched across the grid will be monitored and recorded, allowing the cluster manager to charge accordingly, or to prove quality of service or even to renice the processes on demand (allowing the user to purchase more CPU time or a higher priority).

5   WEBCOM-G IS

The WebCom-G Information Gathering Module (Fig. 9 consists of three components: (1) The Stats Daemon (2) The Information Manager and (3) the WIS

The Stats Daemon sits on the hardware and reports to the Information Manager. The WIS sits above the Information Manager and so the WIS can choose to communicate with either the Stats Daemon directly (if it's only talking to a single machine) or it can talk to an Information Manager (which is a promoted stats daemon and so would have information about more than one machine) or to a third party information service provider such as a Globus GIIS. (see Fig. 10)
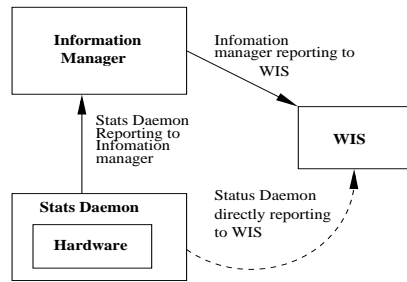
Figure 9: *Architecture of the WebCom-G Information Gathering Module*

The Stats Daemon The Stats Daemon is an independent entity residing on the hardware, which gathers system information. It records information on the CPU load, the RAM (total+used), Pagefiles (total+used), Hard Drives (total+used), Architecture, Processes running, Users, Networking and Kernel version. The Stats Daemon records system information to log files stored on the localhost and provides them over TCP/IP in XML or raw binary format.

The Information Manager The Information Manager module provides detailed runtime and system information about a number of machines. It communicates with one or more Stats Daemons, gathering stats on each machine. It then sorts and partitions the data as required - e.g., giving an average machine load over a cluster for the Grid Administration module. It is possible to partition the information returned by middleware, person or uid and by machines or sub-clusters. This information can then be used by other WebCom-G modules, such as the Load Balancer or by the ESA. Retrieval of information at a given time period (e.g., given a time scale) is essential to determine the nature and health of connected resources and to execute jobs. The system of collection of data at fixed times may not be suitable for heterogeneous environments, which consist of dynamically changing resources. However, this method can be suitable for homogeneous nodes. It is also advantageous where the network load is high, since this model does not use any of the Globus services. Moreover, this information enables the WebCom Scheduler to execute a (less critical) job request on a more suitable resource.

WIS The WIS is the highest-level component of the WebCom-G Information Gathering Module. It will be able to query Globus to retrieve information (see Fig. 10) supplied by Globus through the GIIS and Ganglia (i.e. CPU loads, memory usage etc). It will do this through one of two methods - either by building a new information provider to work with the GRIS or through the use of Commodity Grid kits (e.g., Java Cog Kit), which allow access to the information via the Globus Frame work.

The WIS provides a Java based GUI for Unix and an MFC based one for Windows.

Each WebCom-G enabled machine will have the WebCom-G stats daemon running. The daemon will provide similar functionality to Globus, but will not rely on a single centralised stats server (GRIS, GIIS). Instead, it will use the client promotion feature of WebCom. This allows any client to be promoted to a master. Any Stats Daemon can be promoted to become an Information Manager. This Information Manager can then retrieve information about the cluster in which it is situated. This will eliminate bottlenecks and provide multiple entry points for clusters. Also the user can specify a "sub-cluster" within in a cluster and only retrieve information from and interact with those listed machines.
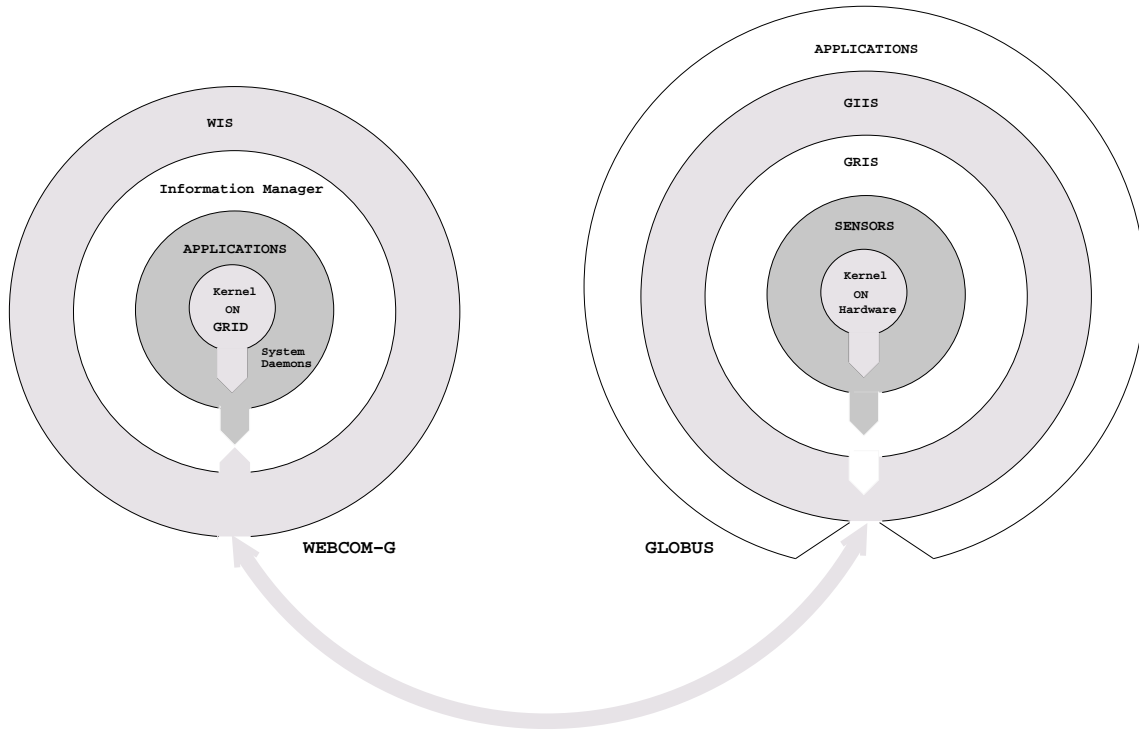
Figure 10: *WIS interopability with Globus*

5.1 Globus Information Gathering

The Grid is a large distributed computing system formed by tightly or loosely coupled computers following a common set of rules for sharing, authentication and job execution thus enabling the creation of virtual organizations within the Grid environment. The Grid is build up of many virtual organizations; resource information about individual systems building the Grid plays an important role in load balancing, resource allocation & re-allocation and execution of jobs within them.

In the case of tightly coupled computers such as Clusters of Workstations there is central information such as the Network Information Service, which stores information about all the individual nodes forming the cluster. In the case of the Grid, information is hosted through information services. Information is the most crucial part of any Grid or meta-computing system. Information about the Grid system can be acquired through information services hosted at different levels in the Grid systems.

Globus is a collection of services toolkit (reference1) used to build computational Grids. The Grid built using Globus toolkit is formed of three supporting layers (reference 2) (1) Information service (2) Data Management services (3) Management services all under one security service. Information services provide information about Grid resources using utilities such as MDS, GRIS, and GIIS.

MDS, at the top level pinnacle provides information directory services for a Grid built using the Globus toolkit. It can be queried to discover the properties of the machines, (architectures, networks, processor availability, bandwidth and disk space). GRIS is a standard information service, which runs on all resources; it interfaces with LDAP and provides information about the resources. GRIS, the core information provider for MDS provides the resource information of the local system which

includes platform type and instruction set architecture, OS version and type, CPU information, Memory, Network interface information and file system summary. Each compute node on the Globus grid runs a GRIS acting as white pages. The GIIS acts as a caching service for searching. Resources register with a GIIS, which in turn publishes the information when requested by the client. MDS uses an LDAP server and (via LDAP protocol and schema) interface for querying, generating, publishing, storing, searching and displaying of such middleware information. Thus the MDS provides tools necessary to build an LDAP based information tree for computational grids. GIIS provides a level of combining individual GRIS services to provide a single system image forming aggregate index service. Thus the GIIS forms yellow pages providing collective indexing and searching function of all the computational resources available in a Grid environment and across multiple virtual organizations forming the Grid.

5.2   Comparing WIS and Globus

Figure 11: *Flow of control and lookup initiation in WebCom-G and Globus*

The application programmer must explicitly use the services provided by the Globus Toolkit to schedule his job. This must be done this at compile time - where the application is coded. Within the WebCom-G system, control is handed over to the WebCom-G kernel, which will automatically seek out necessary information from available services at runtime and will schedule the job accordingly. The application programmer does not need to specify how this is to be done, nor does he need to know how to do it.

Globus and WIS differ in 'lookup initiation' and flow of control (see Fig. 11). In Globus the application programmer specifies the services to be invoked whereas in WIS the kernel will initiate lookups and choose the service to be invoked.

Globus relies on a single server - the GIIS to coordinate information gathering. This can lead to bottlenecks and provides a single point-of-failure. The WIS approach of promoting Stats Daemons to Information Managers avoids the problem of single point-of-failures.

6   FUTURE WORK

Solutions must be developed to free application programmers from the low level complexity of parallel programming in Grid environments. In effect, Grid programming environments must evolve to a point where Grid (and, in general, parallel) programmers are freed from architecture details such as data locality, machine availability, inter-task synchronisation, communication topologies, task load-balancing, and fault tolerance - in the same manner as present day sequential programmers are freed from explicit memory management, disk access protocols and process scheduling. At that point, the grid middleware will adopt the character of a grid operating system and many, if not all, of the issues that make grid programming difficult will no longer reside in the domain of the application developer.

This paper presents the evolution of WebCom, from a metacomputer to middleware, to a Grid middleware, to a Grid Operating System, and details our initial investigations into marshalling Globus tasks. It outlines the WebCom-G Information System and it's interoperability with Globus.

The goal of the WebCom-G Operating System is to hide the Grid, by providing a vertically integrated solution from application to hardware while maintaining interoperability with existing Grid technologies. In addition to maintaining a vertically integrated solution, the available services will be exploited to increase functionality and effect interoperability.

The provision of such a Grid Operating System will remove much of the complexity from the task of the application developer.

ACKNOWLEDGEMENTS

References

[1] Arash Baratloo, Mehmet Karul, Zvi Kedem, and Peter Wyckoff. Charlotte: Metacomputing on the Web. 9th International Conference on Parallel and Distributed Computing Systems, 1996.

[2] S. Bower, J. Cobb, D. Gedye, D. Anderson, W.T. Sullivan(III), and D. Werthimer. A new major SETI project based on Project Serendip data and

100,000 personal computers. In Proceedings of the Fifth International Conference on Bioastronomy, 1997.

[3] Marian Bubak and Pawel Paszczak. HYDRA - Decentralised and Adaptive Approach to Distributed Computing. Workshop on Applied Parallel Computing, Bergen, Norway, June 18-21, 2000.

[4] P. Cappello, B. O. Christiansen, M. F. Ionescu, M. O. Neary, K. E. Schauser, and D. Wu. Javelin: Internet-Based Parallel Computing Using Java. In Geoffrey C. Fox and Wei Li, editors, *ACM Workshop on Java for Science and Engineering Computation*, June 1997.

[5] Karl Czajkowski, Steven Fitzgerald, Ian Foster, and Carl Kesselman. Grid Information Services for Distributed Resource Sharing. Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing.

[6] Distributed.Net.
http://www.distributed.net.

[7] D. H. J Epema, Miron Livny, R. van Dantzig ans X. Evers, and Jim Pruyne. A Worldwide Flock of Condors: Load Sharing among Workstation Clusters. Journal on Future Generations of Computer Systems, Volume 12, 1996.

[8] Cristiana Amza et al. TreadMarks: Shared Memory Computing on Networks of Workstations. IEEE Computer, Pages 18-28, Vol. 29, No. 2, February 1996.

[9] Luis F. G. Sarmenta et al. Bayanihan Computing .NET: Grid Computing with XML Web Services. Workshop on Global and Peer-to-Peer Computing at the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 02), Berlin, Germany, May 2002.

[10] Steven Fitzgerald, Ian Foster, Carl Kesselman, Gregor von Laszewski, Warren Smith, and Steven Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing.

[11] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. International Journal of Supercomputer Applications, 11(2):115-128, 1997.

[12] I. Foster and C. Kesselman. The Grid: Blueprint for a New Computing Infrastructure. Published by Morgan Kaufmann Publishers inc. ISBN:1-55860-475-8.

[13] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidyalingam S. Sunderam. *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing.* MIT Press, Cambridge, MA, USA, 1994.

[14] Mehmet Karul. Metacomputing and Resource Allocation on the World Wide Web. Phd thesis, New York University, May 1998.

[15] Michael J. Lewis and Andrew Grimshaw. The Core Legion Object Model. Proceedings of the fifth IEEE International Symposium on High Performance Distributed Computing.

[16] Satoshi Hirano Luis F. G. Sarmenta and Stephen A. Ward. Towards Bayanihan: Building an Extensible Framework for Volunteer Computing Using Java. ACM 1998 Workshop on Java for High-Performance Network Computing, Palo Alto, California, Feb. 28 - Mar. 1, 1998.

[17] M. O. Rabin M. A. Bender. Online Scheduling of Parallel Programs on Heterogeneous Systems with Applications to Cilk. Theory of Computing Systems Special Issue on SPAA '00, 35: 289-304, 2002.

[18] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. *The International Journal of Supercomputer Applications and High-Performance Computing*, 8, 1994.

[19] Message Passing Interface Forum. MPI-2: Extensions to the Message-Passing Interface. Draft proposal, March 1996.

[20] Sun Microsystems. Remote Method Invocation. http://www.javasoft.com/products/jdk/1.1/docs/guide/rmi/.

[21] John P. Morrison. *Condensed Graphs: Unifying Availability-Driven, Coercion-Driven and Control-Driven Computing.* PhD thesis, Eindhoven, 1996.

[22] John P. Morrison, Brian Clayton, and Adarsh Patil. Comparsion of WebCom in the context of Job Management Systems. International Symposium of Parallel and Distributed Computing (ISPDC 2002), Iasi, Romania, July 17-20, 2002.

[23] John P. Morrison and James J. Kennedy. Fault Tolerance Mechanism Exploiting Referential Transparency in the Condensed Graphs Model. *Proceedings of the international conference on parallel and distributed processing techniques and applications (PDPTA 2001)*, Las Vagas, Nevada, June 25 - 28 2001.

[24] John P. Morrison, James J. Kennedy, and David A. Power. A Condensed Graphs Engine to Drive Metacomputing. Proceedings of the international conference on parallel and distributed processing techniques and applications (PDPTA '99), Las Vagas, Nevada, June 28 - July1, 1999.

[25] John P. Morrison, James J. Kennedy, and David A. Power. Load Balancing and Fault Tolerance on a Condensed Graphs based MetaComputer. The Journal of Internet Technology. Jan 2003.

[26] John P. Morrison, James J. Kennedy, and David A. Power. WebCom: A Volunteer-Based Metacomputer. to be published in the Journal of Supercomputing.

[27] John P. Morrison and Martin Rem. Speculative Computing in the Condensed Graphs Machine. proceedings of IWPC'99: University of Aizu, Japan, 21-24 Sept 1999.

[28] J.P. Morrison, Padraig J. O'Dowd, and Philip D. Healy. Searching RC5 Keyspaces with Distributed Reconfigurable Hardware. ERSA 2003, Las Vegas, June 23-26, 2003.

[29] omg. Common Object Request Broker Architecture, July 1995. http://www.omg.org.

[30] Platform Computing: Load Sharing Facility. http://www.platform.com.

[31] Portable Batch System. http://www.openpbs.org.

[32] Project JXTA. http://www.jxta.org.

[33] Philip A. Lisiecki Robert D. Blumofe. Adaptive and Reliable Parallel Computing on Networks of Workstations. *Proceedings of the USENIX 1997 Annual Technical Symposium*, January 1997.

[34] Luis F. G. Sarmenta. Bayanihan: Web-Based Volunteer Computing Using Java. 2nd International Conference on World-Wide Computing and its Applications (WWCA'98), Tsukuba, Japan, March 3-4, 1998.

[35] Richard Sevenich. Parallel Processing Using PVM. *Linux Journal*, 45, January 1998.

[36] Geoff Stoker, Brian S. White, Ellen Stackpole, T. J. Highley, and Marty Humphrey. Toward Realizable Restrected Delegation in Computational Grids. European High Performance Computing and Networking. Amsterdam, June 25-27, 2001.

[37] The Globus Project. http://www.globus.org.