Architecture and Environment for Enabling Interactive Grids

Vanish Talwar, Sujoy Basu, Raj Kumar
HP Laboratories Palo Alto
HPL-2004-217
November 24, 2004*

E-mail: firstname.lastname@hp.com

access control, data management, dynamic accounts, Grid computing, interactive sessions, QoS, resource management

Traditional use of grid computing allows a user to submit batch jobs in a grid environment. We believe, next generation grids will extend the application domain to include interactive graphical sessions. We term such grids Interactive Grids. In this paper, we describe some of the challenges involved in building Interactive Grids. These include fine grain access control, performance QoS guarantees, dynamic account management, scheduling, and user data management. We present the key architectural concepts in building Interactive Grids viz. hierarchical sessions, hierarchical admission control, hierarchical agents, classes of dynamic accounts, application profiling, user data management, scheduling for interactive sessions, persistent environment settings, and exporting remote desktop. We also describe IGENV, an environment for enabling interactive grids. IGENV consists of GISH -'Grid Interactive Shell', Controlled Desktop, SAC -'Session Admission Control' module, GMMA -'Grid Monitoring and Management Agents', and Policy Engine. We also present our testbed implementation for Interactive Grids using and extending Globus Toolkit 2.0 for the Grid middleware infrastructure, and VNC as the remote display technology.

* Internal Accession Date Only
Published in the Journal of Grid Computing, volume 1, issue 3, pp 231-250, 2003
Approved for External Publication

Journal of Grid Computing **1:** 231–250, 2003.
© 2004 *Kluwer Academic Publishers. Printed in the Netherlands.*

Architecture and Environment for Enabling Interactive Grids

Vanish Talwar, Sujoy Basu and Raj Kumar
Hewlett-Packard Labs, 1501 Page Mill Road, MS 1181, Palo Alto, CA 94304, USA
E-mail: {vanish.talwar,sujoy.basu,raj.kumar}@hp.com

Key words: access control, data management, dynamic accounts, Grid computing, interactive sessions, QoS, resource management

Abstract

Traditional use of grid computing allows a user to submit batch jobs in a grid environment. We believe, next generation grids will extend the application domain to include interactive graphical sessions. We term such grids Interactive Grids. In this paper, we describe some of the challenges involved in building Interactive Grids. These include fine grain access control, performance QoS guarantees, dynamic account management, scheduling, and user data management. We present the key architectural concepts in building Interactive Grids viz. hierarchical sessions, hierarchical admission control, hierarchical agents, classes of dynamic accounts, application profiling, user data management, scheduling for interactive sessions, persistent environment settings, and exporting remote desktop. We also describe IGENV, an environment for enabling interactive grids. IGENV consists of GISH – 'Grid Interactive Shell', Controlled Desktop, SAC – 'Session Admission Control' module, GMMA – 'Grid Monitoring and Management Agents', and Policy Engine. We also present our testbed implementation for Interactive Grids using and extending Globus Toolkit 2.0 for the Grid middleware infrastructure, and VNC as the remote display technology.

1. Introduction

Grid Computing [1, 2] envisions a future where heterogeneous resources could be shared by users across geographical and administrative boundaries, and as a utility. Several efforts [3–7] are underway to architect and deploy a middleware infrastructure for Grid Computing. Commercial acceptance of Grid Computing technology is also steadily increasing. Traditional use of Grid Computing has been for the execution of batch jobs in the scientific and academic community. We believe that next generation grids will extend the application domain to include interactive sessions. Such sessions would allow the end-user to interactively submit interactive jobs to remote nodes in a Grid. The end-user will also be able to view the graphical and interactive output of the submitted jobs and applications through such interactive sessions. Example use cases for such interactive sessions could be for, but not limited to, graphics visualization applications, engineering applications like CAD/MCAD, digital content creation, financial applications, office applications, e-mail applications, software development, command line interactions, streaming media, video games. We consider such Grids in an enterprise environment provided either by Application Service Providers (ASPs) [8] or by in-house IT departments. In [8], we extended the ASP model to provide customers access to a remote *computer's desktop* for interactive use as a *service*. Some of the new requirements for the design of such grids are: fine grain access control, performance QoS, dynamic account management, scheduling, user data management.

Most of the early work on Grids has been for batch jobs. Other work like [9] does not focus on providing interactive job submission 'sessions', and [10] does not address the needs for graphical and multimedia sessions. Existing thin client architectures [11, 12] use remote display technologies but they do not have an architecture in the context of Grids. Neither do the other related works provide a comprehensive framework for access control, QoS, scheduling, account management, and user data management for graphical interactive sessions in a Grid Computing environment. In this paper, we introduce Interactive Grids and describe the architecture and runtime environment for enabling such Grids. The user of an Interactive Grid is given an execution node in the Grid for interactive use. Keyboard and mouse events are sent from the users' submission node to the remote execution node in the Grid, and the output of the applications is viewed by the end-user using remote display technologies like VNC [13]. Our key contributions are:

1. Key architectural concepts for Interactive Grids consisting of:
   (a) Hierarchical sessions,
   (b) Hierarchical admission control,
   (c) Hierarchical agents,
   (d) Classes of dynamic accounts,
   (e) Application profiling,
   (f) User data management,
   (g) Scheduling for interactive sessions in the Grid,
   (h) Persistent environment settings,
   (i) Exporting remote desktop.
2. A runtime environment for enabling interactive grids providing for fine grain access control and QoS for graphical interactive sessions in a Grid, consisting of the following components:
   (a) GISH – a 'Grid Interactive Shell'
   (b) Controlled Desktop
   (c) SAC – a 'Session Admission Control' module
   (d) GMMA – 'Grid Monitoring and Management Agents'
   (e) Policy Engine.

The above components are tightly coupled to each other, and are designed with grid-enabled features.

Our architecture is proposed as an extension to the existing Grid middleware infrastructure. Our implementation uses Globus Toolkit 2.0 [3] as this Grid middleware infrastructure. Our design is modular and policy based allowing for easy extensibility and easy manageability.

The rest of the paper is organized as follows. In Section 2, we present the requirements and issues for building Interactive Grids. Section 3 presents the overall system architecture. In Section 4, we describe our runtime environment IGENV in detail. Sections 5 and 6 describe our hierarchical agents and dynamic accounts framework respectively. In Section 7, we describe the data access and data affinity scheduling algorithm. Section 8 presents discussion and analysis. In Section 9, we describe our implementation. Section 10 presents Related Work and we conclude in Section 11.

2. Requirements and Issues

In this section, we present below the requirements and issues in architecting Grids enabled for interactive sessions.

Fine Grain Access Control. Interactive sessions allow a malicious user to submit unauthorized jobs to the remote node and permit end-users unspecified time of access to the remote node. Further, interactive sessions allow a malicious user to probe for vulnerabilities in the Grid system, and launch attacks against other remote nodes in the Grid system. A fine grain access control in the Grid context is needed.

Performance QoS. Interactive Grids permit end-users to request and launch interactive applications. Such applications are sensitive to response time and real time performance requirements. There is a need to guarantee quality of service for such applications.

Dynamic Account Management. Interactive Grids pose a challenge in terms of account management for arbitrary end-users of the grid.

Scheduling. Interactive Grids require a wide area scheduling system that can perform (a) discovery of resources, (b) matching of resources to user requirements for graphical interactive sessions, (c) global admission control before the launching of graphical interactive sessions, (d) reservation of resources for the desired usage time, as well as fine grained reservations like CPU, network bandwidth reservations, (e) resource assignment, (f) job dispatching, (g) global session state management.

User Data Management. There is a need to provide persistent storage for the user's data. The data needs to be made available to the user on being allocated a session with ownership given to the dynamic account assigned to him.

3. Overall System Architecture

Figure 1 shows the conceptual overview of an Interactive Grid computing system that we are considering. The system consists of heterogeneous execution nodes distributed across multiple administrative domains. These nodes are managed by a Grid Distributed

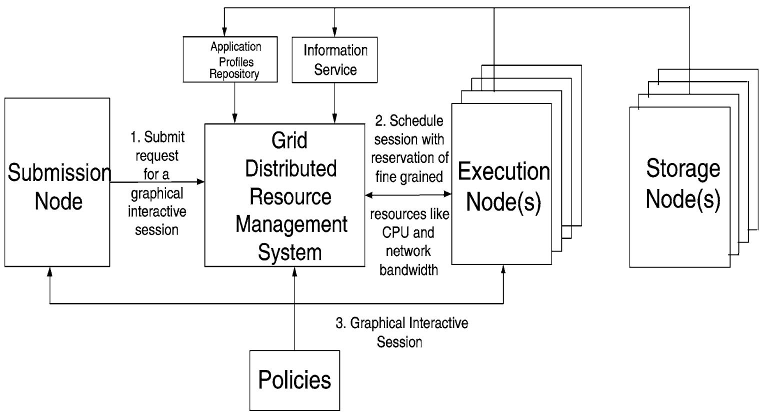

Application Profiles Repository
Information Service
Submission Node
1. Submit request for a graphical interactive session
Grid Distributed Resource Management System
2. Schedule session with reservation of fine grained resources like CPU and network bandwidth
Execution Node(s)
Storage Node(s)
3. Graphical Interactive Session
Policies

Figure 1. Conceptual overview of the Interactive Grid computing system.

Resource Management system (DRM). An end-user submits a request through a submission node to the Grid DRM to obtain a remote execution node for interactive use. On receiving the request from the user, the Grid DRM selects a remote execution node based on the session requirements, and reserves this node for the requested duration of the session. The Grid DRM also performs a reservation of fine grained resources like CPU, network bandwidth and storage bandwidth, for the users' session. At the requested time, the DRM would use remote display technologies like VNC [13] to establish an interactive desktop session between this remote execution node and the end-user's submission node. The end-user then interacts directly with this remote node,[1] through the established session. During this session, the user can submit requests directly to the remote node, to launch multiple applications. These interactions could either be graphical or text-based. We are more interested in addressing the problem for graphical interactive sessions to remote nodes. However, the solutions being proposed and developed by us are also applicable for text-only interactive sessions, as a special case. The interaction of the end-user with the remote node involves the execution of both installed applications and user specified binaries. It is also assumed that the user does not have a local account with the remote node apriori.

We propose hierarchical sessions in such an Interactive Grid computing system consisting of global interactive sessions and per-application interactive sessions. A *global interactive session* constitutes a remote desktop session using remote display technologies like VNC [13]. A global session consists of many per-application sessions. A *per-application interactive session* constitutes the direct interaction between the end-user and the application executing on the remote execution node. A per-application interactive session occurs in the context of a global interactive session. We are most interested in graphical application sessions. All the per-application sessions share the resources allocated to the global session in which they execute. The display for the application session is sent to the user's submission nodes. The data for the application session is accessed from the remote data nodes. Such a hierarchical classification of global and per-application sessions simplifies the management of remote display interactive applications in the Grid.[2] Corresponding to global and per-application interactive sessions, we introduce the notion of hierarchical admission control in our framework consisting of a global admission control module at the Grid DRM node, and a per-application session admission control module at the execution node. The global and per-application session admission control modules make admission control decisions for global and per-application sessions respectively. For simplicity, we refer to the 'per-application session admission control module' as just 'session admission control module'. The following is the sequence of steps in such a proposed system:

1. The end-user creates a job request template for a new global interactive session, specifying the static resource requirements, session requirements, and the de sired list of applications to be launched during the global session. The user could also

[1] The terms 'remote node' and 'remote execution node' are used interchangeably in this paper.

[2] The terms 'global session' and 'global interactive session', 'per-application session' and 'per-application interactive session', are used interchangeably in the remainder of the paper.

specify the desired QoS requirements for the requested applications. The job request is submitted to the Grid DRM node.

2. The request is received by a Grid Scheduler running on the Grid DRM node. In the first pass, the Grid Scheduler performs a matching of resources in the Grid to satisfy the coarse requirements of the user, for example, matching of the hardware requirements of the user. The Grid middleware provides a distributed repository (like MDS [14]), where various resources can publish their services. The scheduler queries this repository to discover resources that match with the user's job needs.
3. In the next pass, the Grid Scheduler interfaces with the Global Admission Control system, which performs the admission check for the requested global interactive session [15]. Application profiles determined through historical logs is used to guide the admission control process.
4. In the final pass, the Grid Scheduler selects the best execution node for the global interactive session based on a resource assignment policy. In this pass, it only considers the nodes that satisfy the Global Admission Control test in step 3. The resource assignment tables are updated to reflect the current assignment of the users' request to the selected execution node.
5. At the requested time, the selected execution node is allocated to the end-user, and the job dispatcher sends the request for the new global interactive session to the allocated execution node along with the SLA for the session.
6. A dynamic account is created by the Dynamic Account Manager for the new global interactive session. The Dynamic Account Manager maintains pools of dynamic accounts on each resource. Unlike normal user accounts which remain permanently assigned to the same real-world user, a dynamic account is assigned to a user temporarily.
7. Appropriate reservations are made on the execution node for the fine grained resources like CPU, network bandwidth, etc., as specified in the SLA for the global session.
8. A configuration process configures the system for the user before launching the global interactive session. This involves, for example, setting up the applications desired by the user, and customizing the user's desktop environment in accordance with the policies for this user.
9. The users's data is now retrieved from persistent storage and setup for the global session with owner-ship given to the dynamic account assigned to him. We also retrieve the users' environment settings, e.g., shell environment settings from the persistent repos itory and set it up appropriately.
10. We then start a remote display server (a VNC server in our testbed) to export the remote desktop to the end-user. A global interactive session is now initiated between the allocated execution node and the end-users' submission node. Session specific monitoring and management agents are also started.
11. The end-user can now request for new per-application interactive sessions directly through the started global interactive session.
12. The requests for per-application interactive sessions are verified by the runtime components Grid Interactive Shell (GISH) and Controlled Desktop for access control checks, and if successful are passed onto the Session Admission Control system on the execution node.
13. The Session Admission Control system performs an admission control check to determine if the requested per application session can be admitted into the global interactive session. If not, the request for new per-application session is denied. Else, the per-application session is started.
14. The Resource Management Monitoring Agents monitor the global interactive session and per-application session utilization values. The monitored data is aggregated by Aggregator Agents. This aggregation is done, for example, based on application, or based on session. This is explained in more detail in Section 5. Enforcement Agents use this data to enforce the SLA and QoS requirements. An Application Predictor system uses the aggregated data to predict the application behavior which is reflected back in the application profiles used by the Grid DRM.
15. The Enforcement Agents end the global interactive session at the SLA specified time. The users' data and environment settings is copied back to the persistent storage.
16. The execution node is now freed up to execute a new global interactive session if scheduled by the Grid Scheduler.

From the above description, we summarize the key architectural concepts as (1) hierarchical sessions, (2) hierarchical admission control, (3) hierarchical agents, (4) classes of dynamic accounts,

(5) application profiles, (6) user data management, (7) scheduling for interactive sessions, (8) persistent environment settings, (9) exporting remote desktop. In addition, there are runtime components provided to the end-user once the session starts. In this paper, we describe in more detail IGENV – the Interactive Grid runtime environment, hierarchical agents, classes of dynamic accounts, and user data management. More details on global admission control and scheduling for interactive sessions is currently a work in progress.

4. IGENV: Interactive Grid Environment

We propose IGENV: a runtime environment for graphical interactive sessions, in our proposed Interactive Grid Computing system. IGENV consists of the following components:

1. Controlled Shell: GISH – 'Grid Interactive Shell'.
2. Controlled Desktop.
3. SAC – 'Session Admission Control' module.
4. GMMA – 'Grid Monitoring and Management Agents'.
5. Policy Engine.

These components provide at runtime fine grain access control, and QoS in interactive grids. Figure 2 shows the interaction of IGENV components in the context of a dynamic account created by the Dynamic Account Manager. In the next few sections, we describe GISH, Controlled Desktop, GMMA, SAC, and Policy Engine.

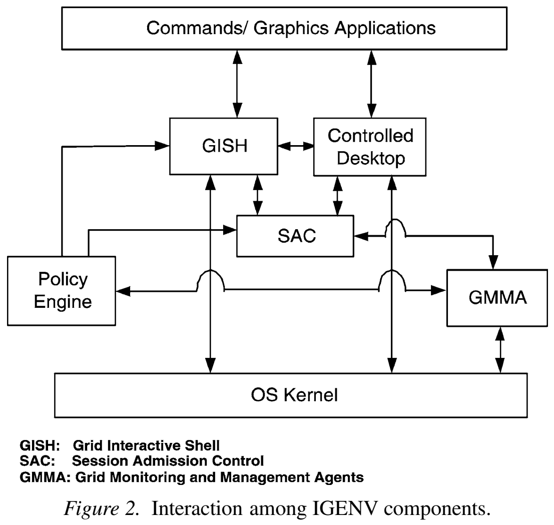

Commands/ Graphics Applications
GISH
Controlled Desktop
SAC
Policy Engine
GMMA
OS Kernel
GISH: Grid Interactive Shell
SAC: Session Admission Control
GMMA: Grid Monitoring and Management Agents

Figure 2. Interaction among IGENV components.

4.1. *Controlled Shell*

A controlled shell provides a restricted interface to the end-user to submit requests for executing applications and commands to the remote node interactively. We have designed a controlled shell called GISH – 'Grid interactive shell'. GISH accepts requests to execute two kinds of commands/applications:

1. Commands and applications that are already installed on the remote node by the system administrator.
2. Commands and applications that are NOT already installed on the remote node, and is a user specified binary file.

GISH allows grid-users to be logged on to the remote node through two kinds of user accounts:

1. Controlled normal user accounts. This corresponds to a normal user account given by the underlying operating system, restricted by the access control policy files.
2. Controlled super user accounts. This corresponds to a super user account given by the underlying operating system, restricted by the access control policy files.

Figure 3 shows the design of GISH. It consists of a command interpreter interfaced to an access control subsystem. The access control subsystem consists of access control modules described in detail below. The user submits a request to start a command or application to GISH. The command is first parsed by the command interpreter, and then passed onto the access control modules. Each of the access control modules performs an access control check. If the access control check fails for any of the modules, a failure message is returned back to the user and the request to start the application/command is denied. If the access control check succeeds for all the modules, then the command or application is started by GISH and the graphical output, if any, can be viewed through the remote graphical display. We describe briefly some of these access control modules below. In order to make the design modular, we choose to interface GISH with a Session Admission Control system (SAC). SAC is responsible for making admission control decisions for session parameters.

4.1.1. *EAM: Executables and Files Access Control Module*

This module is responsible for verifying that the requested command/application (i) belongs to the list of

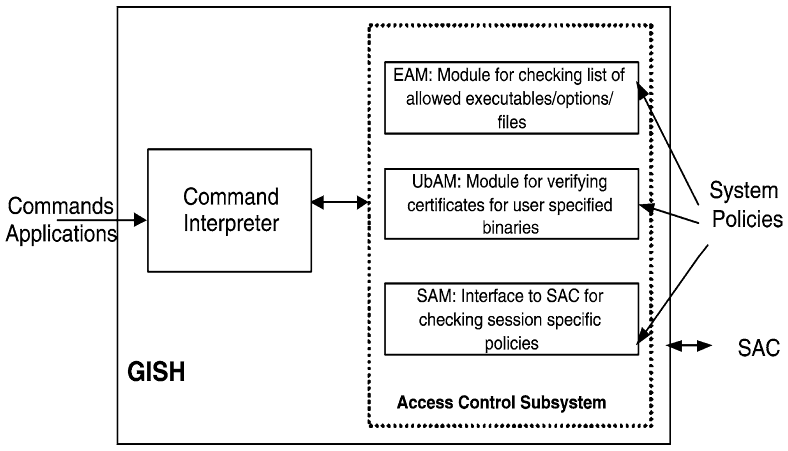

EAM: Module for checking list of allowed executables/options/ files
UbAM: Module for verifying certificates for user specified binaries
SAM: Interface to SAC for checking session specific policies
Commands
Applications
Command Interpreter
System Policies
SAC
GISH
Access Control Subsystem

Figure 3. GISH design.

allowed executables, (ii) is invoked with a list of allowed arguments/options, (iii) only accesses allowed files and directories. This verification is enforced through a policy file which enumerates the list of allowed executables, allowed executable arguments, allowed files and directories for the user. The policy files used by EAM is categorized based on whether the user is logged on as a controlled normal user or as a controlled superuser. For allowed executables, EAM would not be able to determine all of the files and directories that an application would access. In order to restrict the applications from accessing only the allowed files and directories at run-time, we supplement GISH, based on a policy decision, with systems like [16, 17] which compartmentalize the execution of processes, or with virtual machine sandbox environments like [18].

4.1.2. *UbAM: User Binaries Access Control Module*

This module is responsible for verifying the signature for user specified binaries. We assume that there exists trusted services in our grid computing environment that checks user specified binaries and signs non-malicious binaries. UbAM verifies such signatures. If such trusted services are unavailable to the user, we provide based on a policy decision, a virtual machine environment [18] for executing the users's binaries, or supplement the system with runtime system-call monitoring systems [10].

4.1.3. *SAM: Session Access Control Module*

This module interfaces with SAC – 'Session Admission Control' module, for verifying session specific parameters. SAC is explained in detail in Section 4.3. SAM passes the requested command to the SAC. SAC replies back with an 'Allow' or 'Deny' decision for the requested command/application.

The GISH design shown in Figure 3 could be extended with other access control modules as seemed appropriate for a particular implementation. We have presented a few access control modules that we envision to be necessary in a Grid environment for graphical interactive sessions.

4.2. *Controlled Desktop*

The controlled desktop has to be identical to GISH in terms of the policies enforced. The desktop's menus and icons is customized by a desktop configuration file that enforces these policies. At the time of initialization of the session, this file is read in for customizing the desktop. The user is not given permission to modify this file, or to add or modify menu items or icons. Only the allowed executables with allowed arguments, and allowed files for the user is accessible through the controlled desktop.

4.3. *SAC*

SAC stands for Session Admission Control module. This module is responsible for making an admission control decision for a requested application, based on Service Level Agreements (SLAs), and session policies. Figure 4 shows the inputs to a SAC system. These are explained below:

(1) *Requested application*. The graphics application which the user is requesting to be launched. This is provided to SAC by GISH through the SAM module.

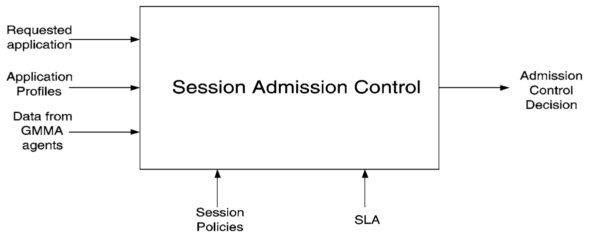


Figure 4. SAC design.

(2) *SLA*. The Service Level Agreement for the session in progress. The SLA is determined prior to the start of the session.
(3) *Application profiles*. The application profiles contain the estimated CPU and bandwidth required for various classes of applications to meet their acceptable performance levels. Example classes of applications are engineering applications, visualization applications, video games, etc. Such application profiles are determined by a system administrator, and refined by an application predictor system.
(4) *Data from GMMA agents*. The resource usage data gathered by GMMA monitoring agents. The GMMA monitoring agents are explained in Section 4.4.
(5) *Policies*. The session policies in place for the session.

Given these inputs, SAC checks the session parameters to verify availability of resources in compliance to SLAs. These session parameters are:
(1) Number of processes launched during a session.
(2) Usage time for a session.
(3) Disk quota usage for a session.
(4) CPU utilization percentage for a session.
(5) Network bandwidth utilization percentage for a session.

SAC compares the current values for these session parameters with the limiting values agreed upon in the SLA. If there is a violation, or if a violation would occur upon executing the application, SAC decides on a 'Deny' decision for executing the application. Otherwise, an 'Allow' decision is made for the application by SAC. Figure 5 shows an algorithm for SAC to make an admission control decision, based on the CPU and network bandwidth utilization parameters for a session.

SAC could be extended to support other session parameters as seemed appropriate for a particular implementation. We have presented a few session parameters that we envision to be necessary in a Grid environment for graphical, interactive sessions.

Input: Application request, reservation agreement from SLA, data from GMMA agents, application profiles

Output: Admission control decision (Allow or Deny)

1. Determine the class of applications that the requested application belongs to.
2 Obtain from the application profiles, the CPU and network bandwidth usage requirement for this application.
3. Use the data gathered by GMMA agents to obtain the current CPU and network bandwidth utilization values for the session.
4. Obtain from the SLA, the CPU and network bandwidth reservation values made for this session.
5. Compare the values from Step 3 and Step 4 to determine the CPU and network bandwidth available for the requested application, to comply with the SLA.
6. Compare the values from Step 2 and 5 to determine if executing the requested application would violate the SLA. If so, return "Deny". Else return "Allow".

Figure 5. An algorithm for SAC with CPU and network bandwidth utilization as the session parameters.

4.4. *GMMA*

GMMA stands for 'Grid Monitoring and Management Agents'. The monitoring agents collect dynamic monitoring data, which is used by the management agents to enforce session SLAs, QoS for applications, intrusion protection, and access control policies. Some of the GMMA agents are associated with a specific session, while some others are system wide agents that monitor all the sessions started through the Interactive Grid environment. The monitoring agents log their information in log files,[3] interface with other peer agents, other monitoring systems, as needed. The management agents use the data gathered by monitoring agents for enforcement purposes. Figure 6 shows the design of GMMA. The figure shows a few categories of these agents based on the functionality to be provided by these agents in the context of Interactive Grids. For example, some of the functionality is to monitor and manage (i) session specific parameters like usage time for the session, number of processes spawned during the session, number of socket connections opened during the session, disk quota usage for the session, etc.; (ii) QoS parameters like CPU utilization, network bandwidth utilization, etc., for

[3] This logged data is used only for session and Grid management purposes. Any privacy issues regarding this information would be agreed upon as an agreement prior to the start of the session.

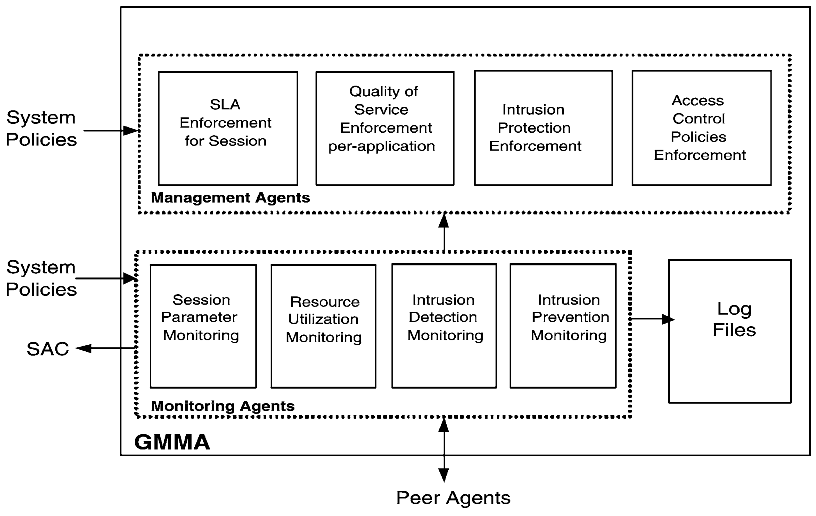

System Policies
SLA Enforcement for Session
Quality of Service Enforcement per-application
Intrusion Protection Enforcement
Access Control Policies Enforcement
Management Agents
System Policies
SAC
Session Parameter Monitoring
Resource Utilization Monitoring
Intrusion Detection Monitoring
Intrusion Prevention Monitoring
Monitoring Agents
Log Files
GMMA
Peer Agents

Figure 6. GMMA design.

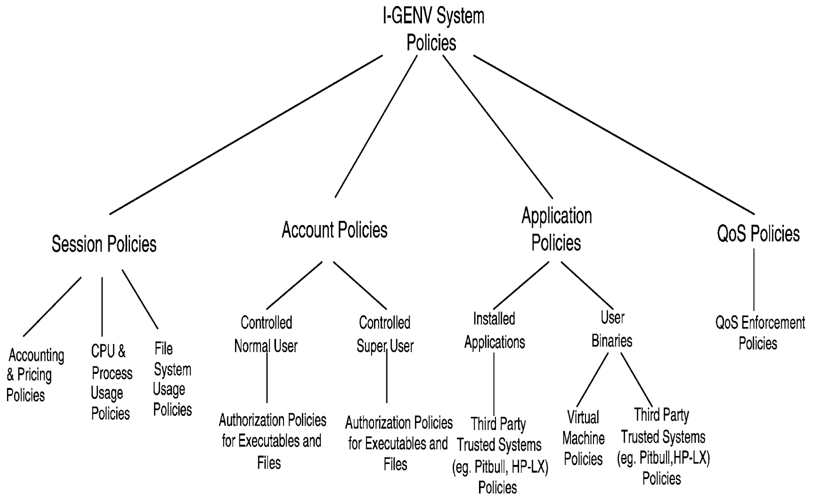

I-GENV System Policies
Session Policies
Account Policies
Application Policies
QoS Policies
Accounting & Pricing Policies
CPU & Process Usage Policies
File System Usage Policies
Controlled Normal User
Controlled Super User
Installed Applications
User Binaries
QoS Enforcement Policies
Authorization Policies for Executables and Files
Authorization Policies for Executables and Files
Third Party Trusted Systems (eg. Pitbull, HP-LX) Policies
Virtual Machine Policies
Third Party Trusted Systems (eg. Pitbull,HP-LX) Policies

Figure 7. Taxonomy of IGENV system policies.

the applications; (iii) security parameters for intrusion detection and access control like IP addresses of incoming and outgoing connections, system calls made by applications, etc.

The GMMA design presented in Figure 6 could be extended with other monitoring and management agents as seemed appropriate for a particular implementation. As for GISH and SAC, we have presented a few categories of monitoring and management agents that we envision to be necessary in a Grid environment for graphical interactive sessions. However, the design can be extended with other categories of monitoring and management agents as well. In Section 5, we describe the architectural aspects for the monitoring and management agents in more detail.

4.5. *Policy Engine*

Policy files are needed for SLA enforcement and security issues. We also need rules covering different scenarios. Thus, a process violating its SLA can be run with a lower priority or be killed under different circumstances, as dictated by policy. Instead of allowing policy-based decisions to be internal to other modules such as GISH, SAC and GMMA, we follow a modular design. The policy engine is driven by rules and can take decision based on configuration parameters in the policy files and information collected by monitoring agents. These decisions are taken when requested, for example, by SAC or management agents. The system policies can be classified into the following categories also shown in Figure 7:

```
 % Example of Account Policies

 % Authorization policy for
 % executables
account pool: MMGRID
action: allow executables
 ls
 mkdir
 cd
 df
 gcc
 vi
```

```
 % Example of Account Policies

 % Authorization policy for
 % files
account pool: MMGRID
action: allow files
 /home/mgrid001/*
 /usr/local/graphics/examples/*
 /usr/local/engg/examples/*
 /usr/include/*
 /usr/local/include/*
```

```
 % Example of Session Policies


event: NUM_PROCESSES > 10
action: KILL

event: USAGE_TIME > 60
action: KILL
```

Figure 8. Example system policy files.

(1) *Session policies.* These specify policy information for each session. Examples of such policies are accounting and pricing policies, CPU and process usage policies, file system and disk quota usage policies. The policies specify the default action to be taken on a violation of the system parameters.

(2) *Account policies.* These specify policy information associated with account pools. There are separate policies for controlled normal users and controlled superusers. Examples of such policies are the authorization polices for executables and files for a user of the account pool.

(3) *Application policies.* These specify policy information for applications that are started by IGENV. There are two kinds of applications: installed applications, and user specified binaries. The execution of these applications could take place in a secure environment using systems like Pit-Bull [16], HP-LX [17], or virtual machine environments [18]. Such systems have their own policies.

(4) *QoS policies.* These specify policy information for QoS metrics. Example policies are the QoS enforcement policies on violation of QoS metrics specified in the application profiles.

Each of the above policies are customized for a given grid-user of the system. Figure 8 shows examples of some system policy files.

4.6. *Discussion for IGENV*

Figure 2 shows the interaction among the components of IGENV. As shown in the figure, there is a tight coupling among the components. These components exist in the context of a dynamic account created by a Dynamic Account Manager. Together, the IGENV components achieve access control, QoS, and Manageability. We explain this below.

4.6.1. *Access Control through IGENV*

Access control for a grid-user is achieved through a combination of GISH, Controlled Desktop, GMMA, SAC, and system policies. Using these components, we can control the access of the user to (i) executables, (ii) files, (iii) network interfaces, (iv) network connections, (v) resource usages decided in SLA.

4.6.2. *QoS through IGENV*

QoS in IGENV is achieved through a combination of SAC, GMMA, and system policy files. We assume that the the Grid middleware would have made CPU and network bandwidth reservation, before the session is launched on the remote node. We also assume the existence of application profiles which contain the estimated CPU and network bandwidth required for various classes of applications to meet acceptable performance levels. GMMA monitoring agents monitor the actual usage values of the CPU and network bandwidth utilization of applications. This data is used to enforce QoS for each application as specified in the application profiles. The data gathered by GMMA monitoring agents is also used by the GMMA management agents for enforcing the reservation limits for sessions stated in the SLA (policing). Further, before a new application is launched, the Session Admission Control System (SAC) verifies that the requested application would consume resources within the reservation limits agreed upon for the session. This check ensures that the SLA and QoS guarantees for currently

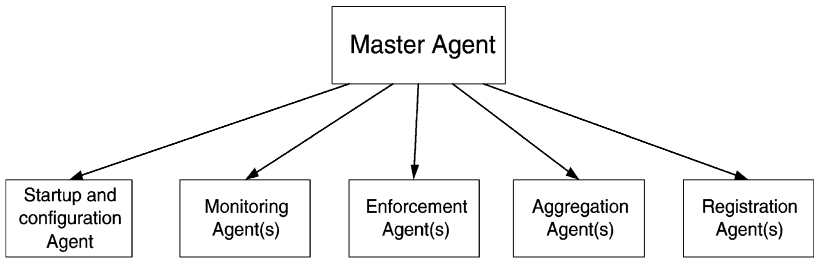


Figure 9. Agents on the execution node.

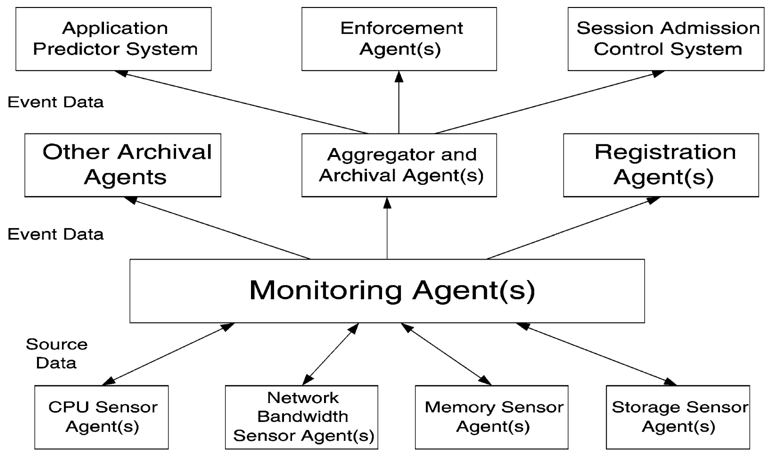


Figure 10. High level overview of the coordination model for components on the execution node.

executing applications, would not be violated upon launching the application.[4]

4.6.3. *Manageability through IGENV*

The system policy files are associated with a pool of dynamic user accounts. A grid user is mapped to one of these pools of dynamic accounts based on a VO-wide policy. The user is then dynamically allocated one of the accounts from the mapped account pool. The user is subject to the system policies associated with that account pool. After the session for the user expires, the dynamic account is returned back to the pool. Such a design coupled with the access control and monitoring agents system provides for easy manageability through the proposed Grid Environment, IGENV.

[4] Even if the SAC were not to perform this admission control check, the monitoring agents would detect the violation and an appropriate enforcement action would be taken. However, there would be a time delay before such an action can take place. Performing a check at the SAC itself ensures no violation of SLA even for this time delay period.

5. Hierarchical Agents

We introduced the GMMA agents as part of the runtime environment, IGENV, in Section 4.4. In this section, we describe the architectural concepts for the agents. Figure 9 shows the agents on the execution node. We assume a master agent that is responsible for all of the agents on the execution node. Figure 10 shows how some of the agents coordinate and interact with each other. This coordination model is based on the producer-consumer paradigm [19]. In Figure 10, some of the components act as both producers and consumers. The source data is provided by Sensor Agents like CPU sensors, memory sensors, network bandwidth sensors, and storage sensors. Monitoring Agents interface to these Sensor Agents, and act as a ‘Producer’ to consumers – Aggregator Agents, Registration Agents, and other archival agents. The Aggregator Agents themselves serve as producers to an Application Predictor system, Enforcement Agents, and Session Admission Control System. The agent implementations could follow open standards like FIPA [20, 21]. The system can be extended to support a registry service, which would aid in supporting information publication about components, and discovery of components. Figure 10 shows these components residing on a single node. We describe the agents below.

5.1. *Startup and Configuration Agent*

This agent is responsible for launching a new global interactive session on the execution node. This agent is also responsible for configuring the system appropriately for the launched global interactive session. For example, in our implementation, this agent configures the KDE desktop environment based on the system policy files corresponding to the allocated dynamic account. Our implemented Startup Agent also starts up a VNC server and connects to the end users' VNC client thus establishing a graphical, global interactive session.

5.2. *Sensor Agents*

The Sensor Agents collect resource information in real time on a continuous basis. These Sensor Agents are off-the-shelf sensors like CPU sensors, memory sensors, network bandwidth sensors, and storage sensors. The Monitoring Agent interfaces with these sensors to obtain this resource information.

5.3. *Monitoring Agent*

The Monitoring Agent acts as a 'Producer' and makes resource usage data available to other components. It itself obtains the resource usage data from Sensor Agents. The Monitoring Agent uses the producer interface as being defined in the Grid Monitoring Architecture [19] to send events to a consumer. The event data is the overall and per-application resource usage data (CPU, network, memory, storage) obtained from the Sensor Agents. The Monitoring Agent could also apply a prediction model on the gathered data and supply the forecasted resource load values to the consumers, for example, the predicted CPU load assuming current set of processes. Based on implementation choice, separate Producer interfaces and interaction channels may be required for each resource type like CPU, memory, network bandwidth, etc. The consumers for the Monitoring Agent in our framework are Aggregator Agents, and Registration Agents. These consumers subscribe to the event data made available by the Monitoring Agent using publish/subscribe model. The Monitoring Agent sends the event data to these consumers at periodic intervals agreed upon in the subscription. Other interaction models may also be considered based on implementation choice. Other consumers of the Monitoring Agent data could be archival agents for storing the history of resource usage information, fault detectors to detect resource aliveness. Based on implementation choice, the event data could be sent as messages to the consumers, or could be communicated via shared memory paradigm. The exact protocols and data formats to be used are implementation dependent.

5.4. *Aggregator Agent*

The Monitoring Agent provides raw resource data from the Sensor Agents. However, we need a framework to support the aggregation of this data. Aggregation is the process of consolidating the resource data obtained from lower level Monitoring Agents. Aggregation allows compaction of data to minimize storage, application of filtering for interpreting data at various granularities, and re-organization of data for computing statistics and inferring events. The Aggregator Agent behaves as a compound Producer/Consumer. It acts as a Consumer to subscribe to per-application resource usage data from the Monitoring Agent. The interval for receiving the event data is determined through policies, and is agreed upon in the subscription. The Aggregator Agent aggregates this received data based on an aggregation function and policies. Some examples of this aggregation are: (i) aggregation by time: all the processes running at a particular point in time are aggregated together. This helps in inferring the load on the system at a particular time; (ii) aggregation by application: for a given application, we aggregate the statistics of the application at multiple points in time. This helps in inferring the behavior of applications; (iii) aggregation by session: for a given session, we aggregate the statistics of the applications belonging to the session, for total session resource usage values like total number of processes launched during the session; the total session wall-clock usage time; total session CPU, network bandwidth, storage utilization. We also obtain information about the average, minimum, and peak resource utilization values for a session. Aggregating data based on session helps in inferring the session behavior, and the conformance to SLAs.

The Aggregator Agents can also apply sampling to the data obtained from the Monitoring Agents at a coarser time interval, thus filtering out some data

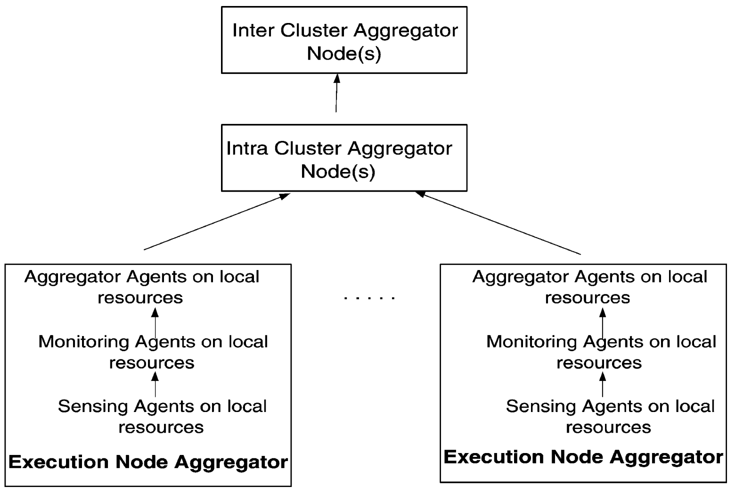

Inter Cluster Aggregator Node(s)
Intra Cluster Aggregator Node(s)
Aggregator Agents on local resources
Monitoring Agents on local resources
Sensing Agents on local resources
Execution Node Aggregator
. . . . .
Aggregator Agents on local resources
Monitoring Agents on local resources
Sensing Agents on local resources
Execution Node Aggregator

Figure 11. Hierarchical Aggregator nodes.

and reducing the data set. The aggregated data is also archived into persistent storage. Prediction models could also be run on the aggregated data to obtain the forecasted resource utilization per application, or per global session, assuming current set of processes. The Aggregator Agent acts as a 'Producer' for the aggregated data to consumers. Some of the consumers we have identified are Application Predictor system, Enforcement Agents, and Session Admission Control System. The interaction between the Aggregator Agent and consumers is based on publish/subscribe or query-response model. Similar to Monitoring Agent, the Aggregator Agent uses the producer and consumer interface as defined in the Grid Monitoring Architecture [19]. The event data format, and communication paradigm is implementation dependent. The Aggregator nodes host the Aggregator Agents. Figure 11 shows a hierarchy of Aggregator nodes. The lowest level corresponds to execution nodes. In addition to sending the aggregated data to the consumers on the execution node, the Aggregator Agents on the execution nodes also send their aggregated data to the intra-cluster Aggregator node. The intra-cluster Aggregator nodes in turn send their aggregated data to inter-cluster Aggregator node, thus forming a hierarchy. The Aggregator Agents on each of these hierarchical Aggregator nodes execute different aggregation functions.

5.5. *Enforcement Agent*

The Enforcement Agent is responsible for enforcing Service Level Agreements (SLAs) for global sessions, and providing guaranteed QoS for graphics applications. The Enforcement Agent behaves as a 'Consumer' to receive aggregated resource usage data from Aggregator Agent. The interval for receiving the data is determined through policies, and is agreed upon in the subscription. These agents take as input the data from the Aggregator Agents, the SLAs for the global sessions, the application profiles, and policies. Using these inputs, it checks for violation of the SLAs or QoS guarantees. Once a violation is detected, an enforcement action is taken. For example, this enforcement action could be one or combination of the following: (i) decrease the priority of applications that exceed their resource utilization levels; (ii) increase the priority of applications falling below their desired resource utilization levels; (iii) kill applications that have violated their resource utilization levels by a large amount. The enforcement process is controlled by policies.

6. Dynamic Accounts

We propose dynamic or template accounts in Interactive Grids to make the resource virtualization more appropriate for grids. The scalability and manageability of the system are enhanced if we do not require grid users to have their personal user accounts on all the machines that are part of the grid. Instead the system administrator has to add the user once to a directory maintained by the virtual organization in which the user has obtained membership. Any site that participates in that virtual organization (VO) will check the user's membership with the directory during authentication, and authorize the user as a dynamic account if he does not have a static account. The dynamic account is chosen from the pool of dynamic

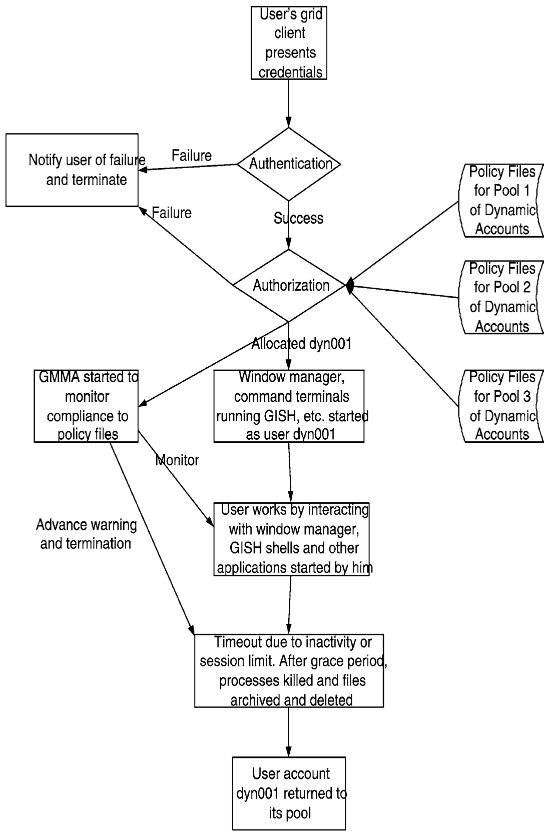

User's grid client presents credentials
Notify user of failure and terminate
Failure
Authentication
Failure
Success
Policy Files for Pool 1 of Dynamic Accounts
Authorization
Policy Files for Pool 2 of Dynamic Accounts
Allocated dyn001
GMMA started to monitor compliance to policy files
Window manager, command terminals running GISH, etc. started as user dyn001
Policy Files for Pool 3 of Dynamic Accounts
Monitor
Advance warning and termination
User works by interacting with window manager, GISH shells and other applications started by him
Timeout due to inactivity or session limit. After grace period, processes killed and files archived and deleted
User account dyn001 returned to its pool

Figure 12. Flowchart describing the process of account allocation, access control and session management.

accounts maintained for that VO. Each dynamic account is a full-fledged Unix account created on the execution node, but without a permanent real-world user associated with it.

Each pool is associated with a set of policy files, customized to the target users of that pool. Unlike normal user accounts that belong permanently to their real-world owners, a dynamic account is bound to a user temporarily. The selection of a pool and the binding of the user to an available dynamic account from that pool are based on the Grid credentials presented. After the successful selection and binding of user to a dynamic account, the graphical interactive session is started. The window manager, terminal windows running the GISH shell, and other programs specified in the window manager's startup files are all started as processes owned by the allocated dynamic account. The entire process can be described by a flowchart shown in Figure 12. The dynamic account is freed at the termination time agreed upon for the session that is using the dynamic account. At the termination time, the management agents kill the processes still running with this account as owner, and delete all files owned by the account. The account is then returned to the pool. We could also choose to archive the files created by the user as against deleting it, on a server maintained by the VO. Subsequent sessions for this user retrieve the files from the archive. Section 7 describes more about the data management aspects.

We organize the dynamic accounts into classes to simplify the work of the system administrator. The system administrator may have to add a new class or modify the attributes of an existing class occasionally, however these activities will not be frequent and the time required will be much less than managing the policy files separately for each user or group or users. Each class can correspond to a role in the real world. Thus, members of the finance department can be assigned to one class, while members of the engineering department can have another class. In our scheme, each class of dynamic accounts is associated with its set of policy files. This set includes, among others, a policy file for resource usage. This can specify the minimum and maximum CPU and network bandwidth utilization by the user's session. Another policy file lists the commands the user is allowed to execute. Yet another policy file lists the directories and files the user is allowed to access.

The different classes of dynamic accounts are arranged in a hierarchy for ease of administration. We illustrate this hierarchy by considering the functions of system administrators. Common functions such as testing and releasing OS patches, restoring files from backup and monitoring and enforcing resource usage limits can be placed in a base class so that all system administrators can perform these functions when needed. Beyond that, there can be groups of system administrators who are responsible for installing and maintaining certain groups of applications. We keep a class of dynamic accounts for each of these application groups. The list of commands and applications allowed for these classes and the directories and files they are allowed to modify are specific to their application group. When a system administrator connects to a grid computing resource and is allocated a dynamic account, the set of policy files governing that account is created by the combination of policies appropriate to this administrator's roles. Thus, if he is responsible for application groups A and B, his policy files will be obtained by merging the basic privileges of system administrators and the privileges of application groups A and B obtained from their respective policy files. By organizing the classes in a hierarchy, we ensure that each class inherits the policy files of its parent classes. Conflict resolution rules are used to determine the value of an attribute inherited from multiple parents. A class can add commands and applications to those its parent classes allow to be executed. It can also override inherited attributes.

7. Data Management

In this section, we describe the architectural features for data management in Interactive Grids. Once an execution node has been assigned for the user, the Grid middleware needs to ensure that the user's files are present on a filesystem accessible from the execution node assigned to him. Furthermore, access rights for the user's files have to be given to the dynamic account assigned to him. We assume that the users' files are archived in a Grid Storage Service (GSS). For interactive sessions on an execution node allocated by the Grid DRM, the user's files containing his session state will have to be restored at the beginning of every interactive session and saved at the end of the session since the dynamic account is associated with the user only for the duration of the session. The Grid Storage Service is used to store the users' files at the end of the session, and the files are then transferred to the appropriate execution node at the beginning of a new interactive session. To minimize the time taken to transfer files at the beginning of an interactive session, it makes sense (i) to cache the users' data on the execution node at the end of the interactive session instead of deleting it; (ii) for the DRM to schedule interactive sessions with *affinity* to the execution node used during the previous interactive sessions by the same user. Assuming that the execution node selected was used in a previous session by the user, there is still no guarantee that the files cached locally will be the latest version since the user might have been assigned a session at another site by the grid middleware between two consecutive sessions on the resource selected. During the session at the other site, he might have modified the files. Hence the middleware should verify whether any of the files cached locally are stale, and if so, invalidate them or get their updated versions, in the background, from GSS.

7.1. *Data Affinity Scheduling Algorithm*

The steps involved in the data affinity scheduling algorithm for allocation of an execution node with dynamic account, in response to a user's request for allocation sent to the grid middleware, are illustrated in a flowchart in Figure 13. Affinity scheduling is possible only for frequent users for whom an execution node has been allocated at this site previously and is available. This is shown in step F. All other users are assigned execution nodes in step E. This involves checking the resource, application and session requirements specified by the user in his job template when requesting an interactive session. For frequent users, this also involves checking history of previous sessions. If the user frequently requested interactive sessions during certain time periods, it is best to assign execution nodes that are expected to be available during those time periods. After selection of the execution node, a check is done on the execution node for the existence of a home directory named by mapping the user's grid credentials. If a directory exists and is owned by a reserved dynamic account, the dynamic account is assigned to the user in step K. If a non-reserved dynamic account is assigned, ownership of the user's home directory tree, if it exists, is given to the dynamic account. The user is logged in with his user ID being the dynamic account assigned. He finds his shell and desktop customized according to the policies being enforced. Files that are stale will be temporarily unavailable while they are updated in the background from the grid storage service. At the end of the interactive session, all files that have been modified during the session are updated in the grid storage system using the user's grid credentials. Also, the user's dynamic account is put in a reserved pool if he is a frequent user. If he returns for another interactive session before the account is reused, the ownership of the files do not have to change.

8. Analysis

We now provide an analysis of our solutions as described in the previous sections. While designing the system, we came up with a list of requirements for our solutions to satisfy. These were: (1) be applicable across heterogeneous platforms, (2) extend existing general purpose tools, (3) require minimal changes to the existing system software, (4) be extensible and modular, (5) address needs of graphics and multimedia applications, (6) support self-managing capabilities, (7) work for all application types, (8) be flexible to interface and interoperate with other complementary and grid solutions, (9) be driven by policies, (10) scalability.

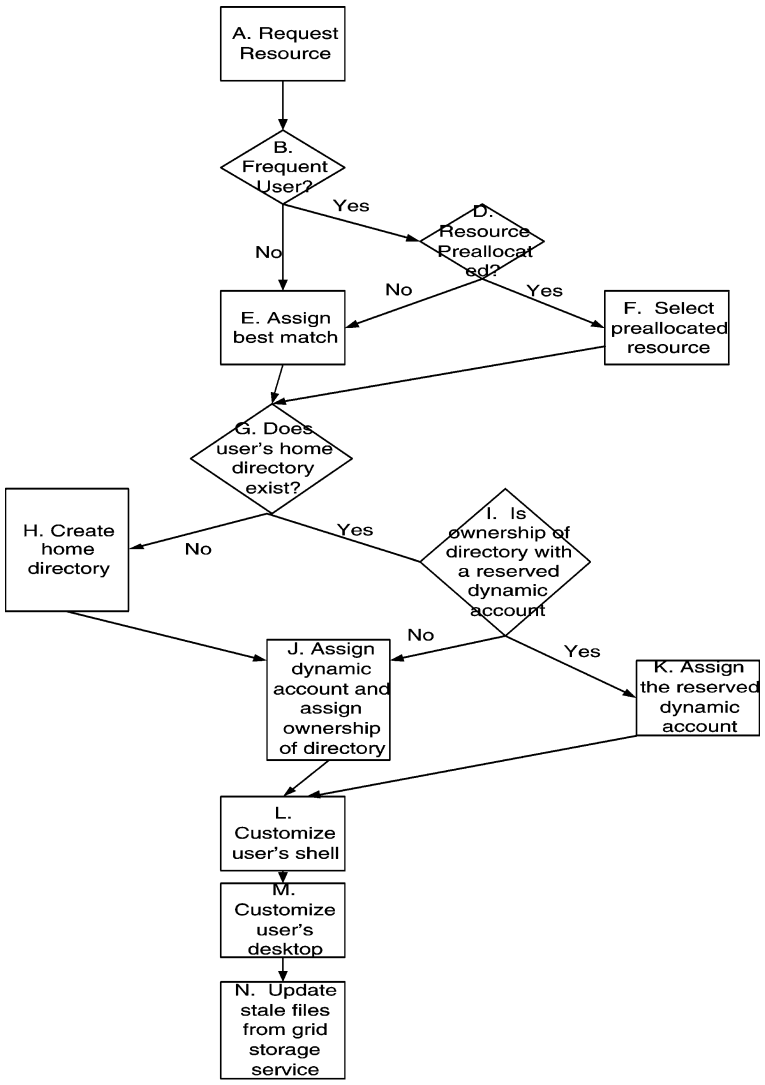

A. Request Resource
B. Frequent User?
Yes
No
D. Resource Preallocated?
No
Yes
E. Assign best match
F. Select preallocated resource
G. Does user's home directory exist?
H. Create home directory
No
Yes
I. Is ownership of directory with a reserved dynamic account
No
Yes
J. Assign dynamic account and assign ownership of directory
K. Assign the reserved dynamic account
L. Customize user's shell
M. Customize user's desktop
N. Update stale files from grid storage service

Figure 13. Flowchart describing data affinity scheduling algorithm.

Towards this end, we avoided as far as possible, designing solutions specific to an OS, or in-kernel solutions, or solutions requiring drastically new tools. This is reflected in our design, for example, through GISH, which was implemented by extending existing popular bash shell. In order to make the design self-managing, we propose monitoring and management agents, which gather run-time information, and enforce appropriate enforcement actions to honor SLAs and access control policies. We also took a two-level approach of (1) filtering commands before execution to verify for access and admission control, and then (2) monitoring and managing the behavior of the system at run-time. In such a two-level approach, we introduce tight feedback loops from the monitoring agents to policy engine and session admission control module, so that dynamic run-time information gathered can be used for subsequent admission and access control decisions. This helps in making the system self-learning and self-managing. The two-level approach is also expected to provide performance benefits by eliminating some of the non-compliant application requests at the shell itself. Since we desired to design the system to work for any application, we avoided QoS solutions like QuO [22] from the Quorum project [23], that specifically address needs of distributed object systems built over CORBA-like middleware infrastructures. Rather, in order to provide QoS management we provide (1) functionality in the Interactive Grid DRM to allocate the most appropriate available resource(s) that would satisfy the QoS requirement of the application, (2) monitoring and enforcement framework to adjust fine-grain resource allocations of applications at run-time to enforce QoS guarantees.

Our solution for the runtime components in IGENV is believed to be efficient as it proposes an integrated set of components to together solve the problems of fine grain access control, QoS, and manageability, compared to providing separate solutions to each of these problems.

In terms of runtime performance analysis, we are interested in reducing the overhead caused during an interactive session in an interactive grid computing system. The overhead caused by Dynamic Account Manager is only at the beginning of a session, and is expected to be minimal since it would primarily involve accessing the *gridmapfile*, and executing a simple logic to map the user to the appropriate account. The overheads caused by the Controlled Shell – GISH, and Session Admission Control module – SAC, is also expected to be insignificant compared to the high human response time expected while the user is interactively submitting commands. The overhead caused by monitoring and management agents would incur in terms of filesystem read access, agent-agent communication, enforcement algorithms. These can be addressed through appropriate implementation techniques and as future work, we are designing the monitoring and management agents taking these factors into consideration, so as to not limit the usefulness of the system due to these overheads.

9. Implementation

We have a working prototype implementation for Interactive Grids. Our implementation environment consists of Intel x86 machines running Red Hat Linux 7.3, as the remote nodes. The end-user can request for one of these remote nodes for interactive use from any machine supporting a web browser. We use Globus Toolkit 2.0 [3] as the Grid middleware platform, and VNC [13] as the remote display technology for remote graphical sessions. GPDK [24] is used to provide a web portal to the end-user for submitting job requests. We have extended the functionality of Globus, so that it can also be used for submitting requests for starting a graphical interactive session to the remote nodes. Globus GSI certificates are used for authentication. We have an implementation of a Scheduler that determines the appropriate node to be allocated for a users' request. Figure 14 in the Appendix shows the job submission process. Once a node is determined for the users' request, the appropriate account pool for the user is determined. A set of policy files is associated with this account pool. A dynamic account from this pool is then allocated for this user. We then start a VNC server and GMMA agents for this session. On a successful VNC authentication, the user is presented with a controlled KDE Desktop environment containing only the applications and menus the user is allowed to access. The KDE desktop environment is pre-configured by the system administrator for each pool of accounts.

The session starts with default startup applications, including a GISH shell. The GISH shell has been implemented as an extension to the popular GNU bash shell for Linux and Windows. The shell source code was modified so as to include the access control modules. GISH currently checks for list of allowed executables from a file, before executing commands. The GMMA Agents started at the beginning of the session, run with super–user privileges. They record the session and system information and store them in predetermined files. Currently, the GMMA agents check for the usage time for the session, number of spawned processes. The system policy files contain information about the session usage time, number of allowed processes, etc., along with their maximum allowed values, and actions to be taken on violation of these policies. The current default action is to *kill* all the processes and end the session, on violation of the session policies. Implementation for other modules and agents for GISH and GMMA is a work in progress. Figure 15 shows some interaction examples within GISH during a session in progress, and Figure 16 shows the session screen on termination.

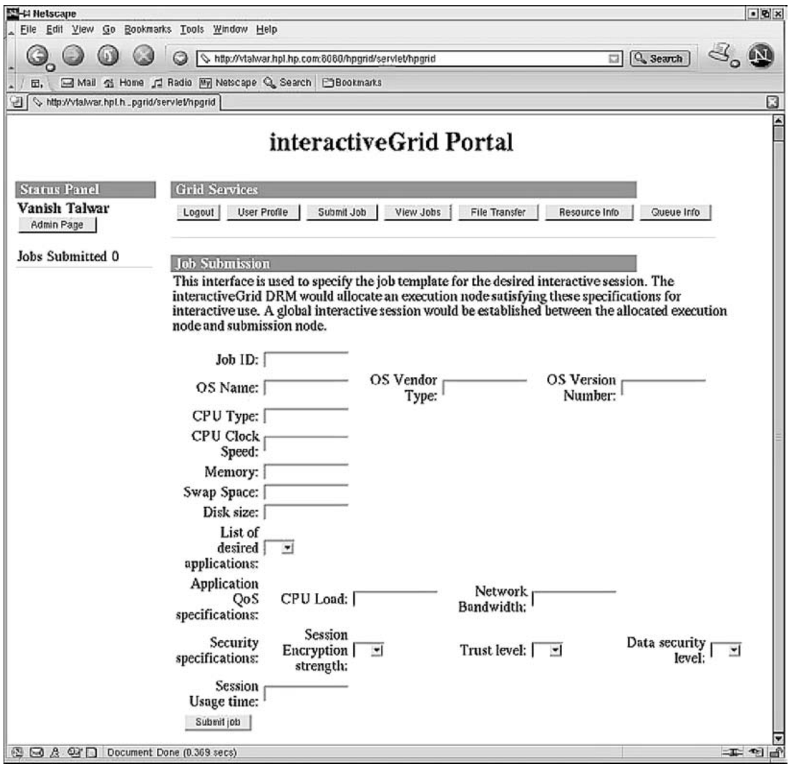
To support dynamic accounts, we modified *globus_gatekeeper* and GSI-SSH daemon from Globus Toolkit 2.0 by linking them with a modified library for reading the *gridmapfile*. Normally the *gridmapfile* contains entries mapping the distinguished name (DN) of the user to the local Unix account. As a first step, we modified the *gridmapfile* to replace the local Unix account name with a predefined string for the user's VO, indicating that a dynamic account from the VO's pool should be used for this user. To get our modified library for reading the *gridmapfile*, we started with a patch to the *globus_gss_assist* package, distributed by the Grid for UK Particle Physics, obtained from [25]. As a second step, we extend this library further to remove the requirement of having an entry in the *gridmap-file*. When the user authenticates himself, the DN obtained from his certificate will be queried against the directory maintained by the VO for membership. If the user is a member, he will be assigned a dynamic account from the pool customized for that VO.

10. Related Work

Majority of the work in the area of grid computing has been for batch jobs and hence do not address the problems as outlined in the paper. The same holds with recent projects on interactive applications like CrossGrid [9]. Solutions developed in Punch project [10] do not address graphical and multimedia sessions, gsissh [26] provides for encryption of interactive sessions and can be used with our solution. QoS solutions provided through Quorum project [23, 22] address the needs of distributed object systems and provide QoS support in underlying middleware infrastructure like CORBA. We propose QoS support for applications through appropriate initial resource allocation and subsequent monitoring and adjustment of resource allocations by the Grid resource management framework. We do not assume any available support for application adaptation through multiple application behaviors. Several thin client system architectures have been proposed including Citrix Metaframe [11] and SunRay systems [12]. However, to our knowledge, none of these systems have been considered in the context of Grids. Our solution extends the scope of thin client architectures to become part of Grids. Most of the other related work for access control and QoS are in the non-grid context do not completely satisfy the requirements for Grid. For example, traditional OS access control mechanisms do not allow to easily enforce fine grain access control for arbitrary end-users. Sudo [27] is not a replacement for shell and does not provide a complete solution for all our needs in Grid context. Our solution provides for an integrated and comprehensive solution for problems of access control, QoS, and account management in the context of graphical interactive sessions in Grids. Our solution is not specific to any remote display technology and can be used with systems like [13, 28]. Our architecture for hierarchical agents leverages the architectural framework being defined in the Grid Monitoring Architecture [19]. Unlike the Grid Monitoring systems being developed [29–31], our monitoring infrastructure addresses the goal of providing QoS guarantees for graphical applications. We plan to leverage the existing low level hardware and software sensors to collect CPU, memory, network bandwidth measurement data. Dynamic account management has been described in [32, 33]. However, we differ from the prior work in using the dynamic account as a component of our customizable grid environment, by associating each pool of dynamic accounts with its set of policy files that IGENV enforces.

11. Conclusions

In this paper, we introduced Interactive Grids which allow end-users access to remote execution nodes belonging to a Grid, for graphical interactive use. Interactive Grids extend the application domain for Grid computing systems from traditional batch jobs to graphical, interactive sessions. We described some of the problems posed for the design of interactive grids, namely that of fine grain access control, performance QoS, dynamic account management, scheduling, and user data management. In this paper, we have presented key architectural concepts in designing Interactive Grids. They are hierarchical sessions, hierarchical admission control, hierarchical agents, classes of dynamic accounts, application profiling, user data management, wide area grid scheduling for interactive sessions, persistent environment settings, and exporting remote desktop. We also presented IGENV: a runtime environment for enabling interactive grids. Our approach has been in building a set of components addressing the issues of fine grain access control, QoS, and manageability, in an integrated but modular manner. We believe this leads to more efficiency. The components are GISH – ‘Grid Interactive Shell’, Controlled Desktop, SAC – ‘Session Admission Control’ system, GMMA – ‘Grid Monitoring and Management Agents’, Policy Engine. While designing the system, we realized the need to satisfy important requirements of heterogeneous platforms, easy extensibility, modularity, and self-managing capability. We identified the areas of overheads that would occur with a deployment of our solution, which would be considered for optimization in the future work. We also described our implementation of the system on Linux x86 machines, using and extending Globus Toolkit 2.0 for the base grid middleware infrastructure and VNC as the remote display technology.


Netscape
File Edit View Go Bookmarks Tools Window Help
http://vtalwar.hpl.hp.com:8080/hpgrid/servlet/hpgrid
Search
Mail Home Radio Netscape Search Bookmarks
http://vtalwar.hpl.h...pgrid/servlet/hpgrid
Document: Done (0.369 secs)

interactiveGrid Portal

Status Panel

Vanish Talwar

Admin Page

Jobs Submitted 0

Grid Services

Logout | User Profile | Submit Job | View Jobs | File Transfer | Resource Info | Queue Info

Job Submission

This interface is used to specify the job template for the desired interactive session. The interactiveGrid DRM would allocate an execution node satisfying these specifications for interactive use. A global interactive session would be established between the allocated execution node and submission node.

Job ID:

OS Name: OS Vendor Type: OS Version Number:

CPU Type:

CPU Clock Speed:

Memory:

Swap Space:

Disk size:

List of desired applications:

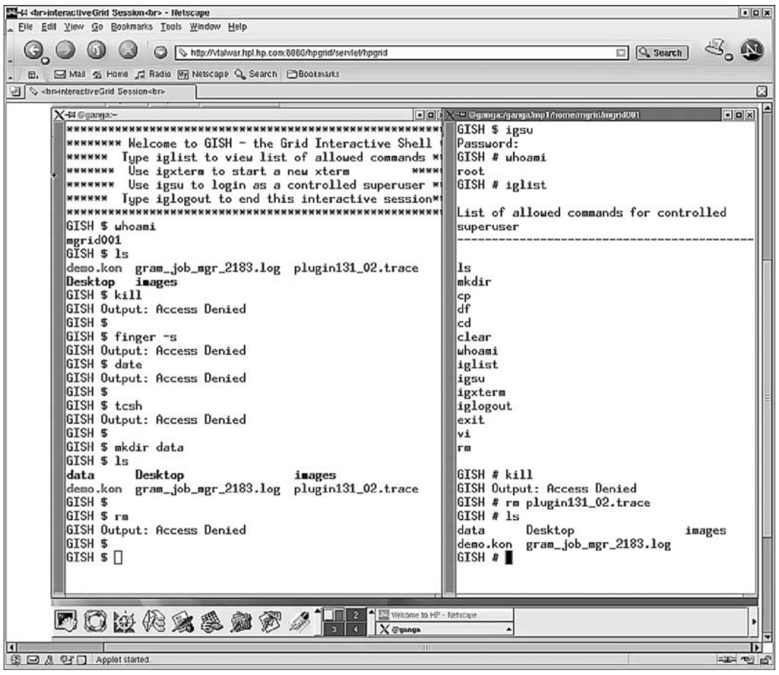
Application QoS specifications: CPU Load: Network Bandwidth:

Security specifications: Session Encryption strength: Trust level: Data security level:

Session Usage time:

Submit job

Figure 14. Job submission screen for a graphical interactive session in an Interactive Grid computing system.



interactiveGrid Session
 - Netscape
File Edit View Go Bookmarks Tools Window Help
http://vtalwar.hpl.hp.com:8080/hpgrid/servlet/hpgrid
Search
Mail Home Radio Netscape Search Bookmarks

interactiveGrid Session
@ganga:~
@ganga:/ganga/tmp1/home/mgrid/mgrid001
Welcome to HP - Netscape
@ganga
Applet started.

```
************************************************************
******** Welcome to GISH - the Grid Interactive Shell
******  Type iglist to view list of allowed commands *
*******  Use igxterm to start a new xterm         *****
*******  Use igsu to login as a controlled superuser *
******  Type iglogout to end this interactive session*
************************************************************
GISH $ whoami
mgrid001
GISH $ ls
demo.kon  gram_job_mgr_2183.log  plugin131_02.trace
Desktop   images
GISH $ kill
GISH Output: Access Denied
GISH $
GISH $ finger -s
GISH Output: Access Denied
GISH $ date
GISH Output: Access Denied
GISH $
GISH $ tcsh
GISH Output: Access Denied
GISH $
GISH $ mkdir data
GISH $ ls
data      Desktop                images
demo.kon  gram_job_mgr_2183.log  plugin131_02.trace
GISH $
GISH $ rm
GISH Output: Access Denied
GISH $
GISH $ 
```

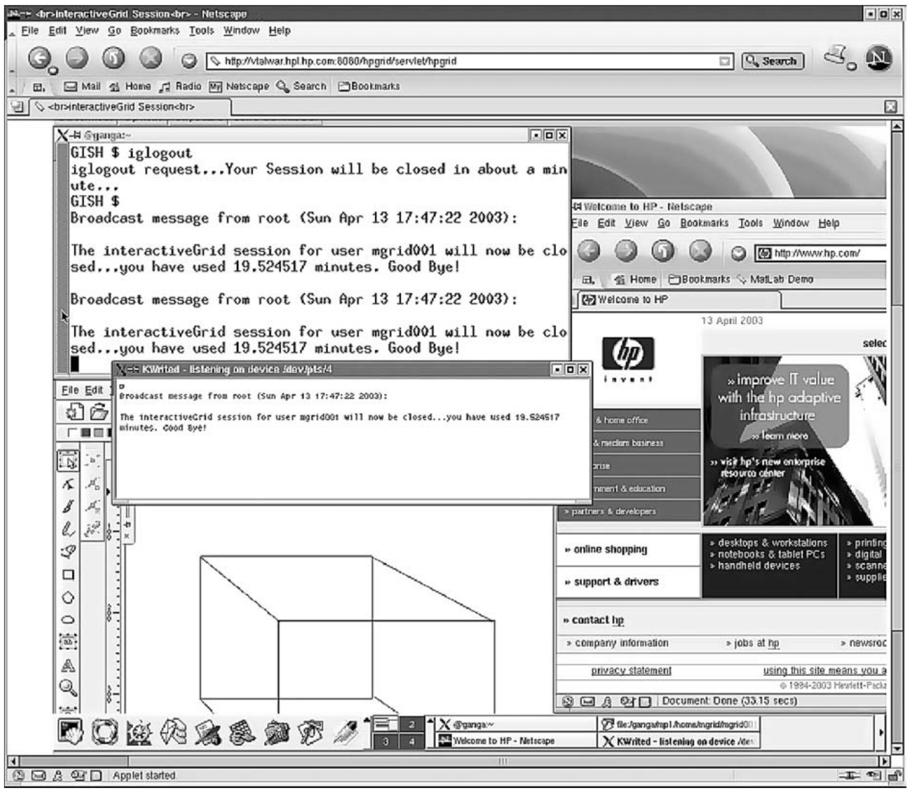
```
GISH $ igsu
Password:
GISH # whoami
root
GISH # iglist

List of allowed commands for controlled
superuser
-------------------------------------------


ls
mkdir
cp
df
cd
clear
whoami
iglist
igsu
igxterm
iglogout
exit
vi
rm

GISH # kill
GISH Output: Access Denied
GISH # rm plugin131_02.trace
GISH # ls
data      Desktop                images
demo.kon  gram_job_mgr_2183.log
GISH # 
```

Figure 15. Screen during an interactive session in progress.



interactiveGrid Session
 - Netscape
File Edit View Go Bookmarks Tools Window Help
http://vtalwar.hpl.hp.com:8080/hpgrid/servlet/hpgrid
Search
Mail Home Radio Netscape Search Bookmarks
@ganga:~
Welcome to HP - Netscape
http://www.hp.com/
Home Bookmarks MatLab Demo
Welcome to HP
13 April 2003
» improve IT value with the hp adaptive infrastructure
» learn more
» visit hp's new enterprise resource center
» online shopping
» support & drivers
» contact hp
» company information
» jobs at hp
privacy statement
Document: Done (33.15 secs)
KWrited - listening on device /dev/pts/4
Welcome to HP - Netscape
Applet started

```
GISH $ iglogout
iglogout request...Your Session will be closed in about a min
ute...
GISH $
Broadcast message from root (Sun Apr 13 17:47:22 2003):

The interactiveGrid session for user mgrid001 will now be clo
sed...you have used 19.524517 minutes. Good Bye!

Broadcast message from root (Sun Apr 13 17:47:22 2003):

The interactiveGrid session for user mgrid001 will now be clo
sed...you have used 19.524517 minutes. Good Bye!
```

```
Broadcast message from root (Sun Apr 13 17:47:22 2003):

The interactiveGrid session for user mgrid001 will now be closed...you have used 19.524517
minutes. Good Bye!
```

Figure 16. Screen at a session logout.

References

1. I. Foster and C. Kesselman (eds.), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kauffman Publishers, 1999.
2. I. Foster, C. Kesselman and S. Tuecke, “The Anatomy of the Grid: Enabling Scalable Virtual Organizations”, *International Journal of SuperComputing Applications*, Vol. 15, No. 3, 2001.
3. I. Foster and C. Kesselman, “Globus: A Metacomputing Infrastructure Toolkit”, *International Journal of SuperComputing Applications*, Vol. 11, No. 2, pp. 115–128, Summer 1997.
4. Nasa ipg. http://www.ipg.nasa.gov.
5. Teragrid project. http://www.teragrid.org.
6. Eurogrid project. http://www.eurogrid.org.
7. Entropia. http://www.entropia.com.
8. S. Basu, V. Talwar, B. Agarwalla and R. Kumar, “Interactive Grid Architecture for Application Service Providers”, in *Proceedings of the International Conference on Web Services (ICWS)*, June 2003.
9. Crossgrid. http://www.crossgrid.org.
10. A.R. Butt, S. Adabala, N. Kapadia, R. Figueiredo and J.A.B. Fortes, “Fine-grain Access Control for Securing Shared Resources in Computational Grids”, in *IPDPS*, April 2002.
11. http://www.citrix.com.
12. http://wwws.sun.com/sunray/sunrayl/.
13. T. Richardson, Q. Stafford-Fraser, K.R. Wood and A. Hopper, “Virtual Network Computing”, *IEEE Internet Computing*, Vol. 2, No. 1, pp. 33–38, 1998.
14. Globus mds. http://www.globus.org/mds.
15. P. Mundur, R. Simon and A. Sood, “Integrated Admission Control in Hierarchical Video-on-demand Systems”, in *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, June 1999, pp. 220–225.
16. Pitbull lx white papers. http://www.argus-systems.com.
17. N. Edwards, J. Berger and T.H. Choo, “A Secure Linux Platform”, in *5th Annual Linux Showcase and Conference*, November 2001.
18. E. Bugnion, S. Devine, K. Govil and M. Rosenblum, “Disco: Running Commodity Operating Systems on Scalable Multiprocessors”, *ACM Transactions on Computer Systems*, Vol. 15, No. 4, pp. 412–447, 1997.
19. R. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor and R. Wolski, “A Grid Monitoring Architecture”, GGF Document Series available from http://www.gridforum.org.
20. Fipa. http://www.fipa.org/repository/index.html.
21. Bigus and Bigus, *Constructing Intelligent Agents Using Java*, 2nd edn, John Wiley, 2001.
22. J. Zinky, D. Bakken and R. Scantz, “Architectural Support for Quality of Service for Corba Objects”, *Theory and Practice of Object Systems*, Vol. 3, No. 1, 1997.
23. Quorum. http://www.dist-systems.bbn.com/projects/QuOIN/.
24. J. Novotny, “The Grid Portal Development Toolkit”, *Concurrency-Practice and Experience*, 2000.
25. http://www.gridpp.ac.uk/gridmapdir.
26. Gsi-ssh. http://www.ncsa.uiuc.edu/Divisions/ACES/GSI/openssh/.
27. Sudo. http://www.courtesan.com/sudo/.
28. Sgi opengl vizserver. http://www.sgi.com/software/vizserver.

29. B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee and M. Thompson, "A Monitoring Sensor Management System for Grid environments", in *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing*, IEEE Computer Society, Washington, DC, pp. 97–104, August 2000.
30. A. Waheed, W. Smith, J. George and J. Yan, "An Infrastructure for Monitoring and Management in Computational Grids", in *Selected Papers from the 5th International Workshop on Languages, Computers, and Run-time Systems for Scalable Computers*, Springer-Verlag, London, pp. 235–245, 2000.
31. R. Wolski, N. Spring and J. Hayes, "The Network Weather Service: A Distributed Performance Forecasting Service for Metacomputing", *Future Generation Computer Systems*, Vol. 15, Nos. 5–6, pp. 757–768, 1999.
32. T.J. Hacker and B.D. Athey, "A Methodology for Account Management in Grid Computing Environments", in *2nd International Workshop on Grid Computing*, November 2001, Springer-Verlag.
33. "An Accounting System for the Datagrid Project Version 3.0". http://server11.infn.it/workload-grid/docs/DataGrid-01-TED-0115-3_0.pdf.