A RESOURCE MANAGEMENT FRAMEWORK FOR INTERACTIVE GRIDS

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

ABSTRACT

Traditional use of Grid Computing Systems has been for batch jobs in the scientific and academic computing. We envision the next generation Grid computing systems to support graphical interactive sessions. In this paper, we propose a resource management framework for supporting graphical interactive sessions in a Grid computing system. We describe the high level architectural resource management framework distributed among the submission node, central scheduler node, and the execution node. We then describe in detail the resource management framework on the execution node. The description of the resource management framework on the scheduler node is kept at a high level in this paper. The framework on execution nodes consists of resource management agents, an admission control system and application predictor system. The agents on the execution node are startup agents, sensor agents, monitoring agents, aggregator agents, enforcement agents and registration agents. The session admission control system is responsible for determining if a new application session can be admitted to the execution node. An application predictor system is responsible for predicting the resource utilization behavior of applications based on data obtained from the resource management agents. The proposed framework allows for implementation of a scalable and extensible middleware for interactive grid resource management. It supports fine grained performance guarantees specified in service level agreements and brings forth some important and novel contributions to enable graphical interactive sessions on Grids.

1. INTRODUCTION

Grid Computing technology [1] provides resource sharing and resource virtualization to end-users, allowing for computational resources to be accessed as a utility. Resource Management is one of the key research areas for Grid Computing. Traditionally, Grid technologies have been used for executing batch jobs in the scientific and academic community. We believe that the application domains addressed by Grid technologies need to be extended to include graphical, interactive sessions. We propose interactive grids - next generation grids addressing the needs of graphical, interactive sessions. Interactive Grids permit end-users to access and control a remote resource eg. remote workstation in the Grid for graphical, interactive use. Such an interactive session on a remote workstation can be used for graphics visualization applications, engineering applications like CAD/MCAD, digital content creation, streaming media, video games, text editing, command line interactions, e-mail applications. Applications execute on the remote workstation, and the end-user can view the graphical output of the applications using remote display technologies like VNC [2]. Distributed Resource Management is one of the key research areas for the design of interactive grids. The resource management problem in our work is broken as :

(i) Wide-area Scheduling of the job requests for graphical interactive sessions onto execution nodes in the Grid.

(ii) Fine grained resource management on execution nodes, during the progress of graphical interactive sessions.

In this paper, we first present a high level architecture of our proposed framework, and then focus on the second problem of fine grain resource management on execution nodes. The key contribution of our paper is the resource management framework on the execution node consisting of resource management agents, a session admission control system, and an application predictor system.

2. REQUIREMENTS

We are considering interactive grids [3] - Grid computing systems that extend the application domain to include graphical interactive sessions. Specifically, an Interactive Grid Computing System allows the end-user access and control of a remote resource in the Grid for graphical, interactive use. To enable such grids, we require a resource management architecture that effectively manages the vast heterogeneous resources across administrative domains, as well as effectively manages the resources during the graphical interactive session. This leads to the following requirements:

(1) 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 allocation. (f) Job dispatching. (g) Global session state management.
(2) Local resource management of allocated resources during a graphical interactive session to perform (a) Fine grained monitoring of resources. (b) Enforcement of Service Level Agreements (SLAs) and Quality of Service (QoS) guarantees for graphical interactive sessions and applications. (c) Fine grain admission control for per-application sessions. (d) Per application session state management.

3. IMPORTANT ISSUES

The important issues to consider for a resource management framework for interactive grids, as compared to traditional batch-oriented grids are: (i) Providing QoS guarantees for graphical sessions. (ii) Guaranteeing SLAs per graphical session. (iii) Accurate prediction of application behavior and resource load.

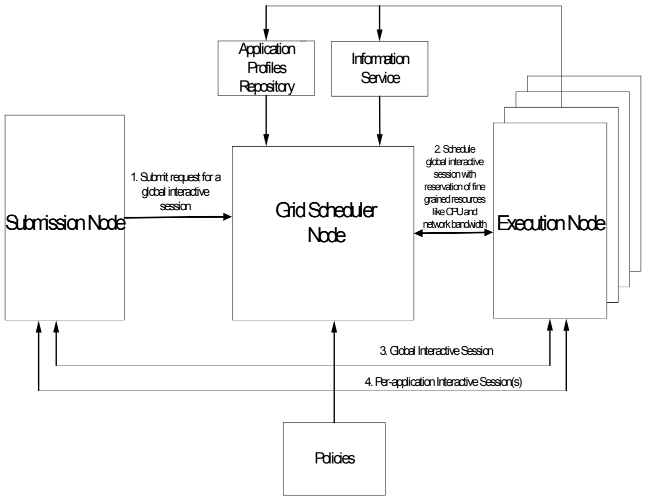

Application Profiles Repository
Information Service
Submission Node
1. Submit request for a global interactive session
Grid Scheduler Node
2. Schedule global interactive session with reservation of fine grained resources like CPU and network bandwidth
Execution Node
3. Global Interactive Session
4. Per-application Interactive Session(s)
Policies

Figure 1: High level overview of the Interactive Grid computing system

4. PROPOSED ARCHITECTURE

We first define some terms as used in the remainder of the paper (refer to Figure 1 for a context):

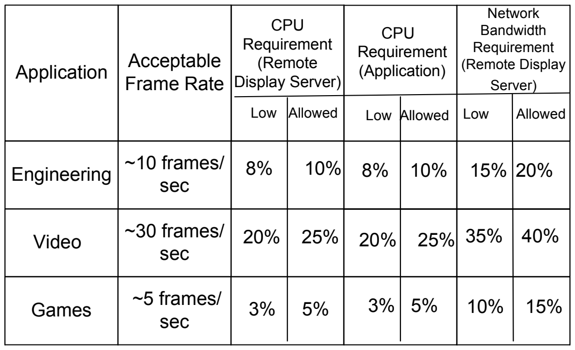
4.1. Definitions

Interactive Grid Computing System: An interactive grid computing system is a Grid computing system supporting Graphical Interactive sessions to remote nodes. At a high level, it consists of Submission nodes, a Distributed Resource Management System, Execution nodes and Storage nodes. The end-user submits job requests through a submission node, and is given access to a remote execution node for graphical, interactive use.

Global Interactive session: A global interactive session constitutes the association between the end-user and the remote execution node, wherein: the end-user interacts with the remote execution node to launch one or more applications, and subsequently interacts with the launched applications through per-application sessions. An example of a global interactive session is the VNC remote display session wherein the graphical desktop of the remote node is exported. We are most interested in such graphical global interactive sessions.

Per-application interactive session: A per-application interactive session for an application executing on the remote execution node, constitutes the association between the end user and the executing application, wherein: the end-user interacts directly with the application. A per-application interactive session occurs in the context of a global interactive session. We are most interested in graphics application sessions. However, our proposed solution would also work with text only applications as a special case. (Note: 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.)

Application Profiles: The application profiles contain the estimated CPU and bandwidth required for various classes of applications to provide acceptable frame rate and performance levels while executing remotely in an Interactive Grid computing system. 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. Figure 2 shows example application profiles.



Application	Acceptable Frame Rate	CPU Requirement (Remote Display Server)		CPU Requirement (Application)		Network Bandwidth Requirement (Remote Display Server)	
		Low	Allowed	Low	Allowed	Low	Allowed
Engineering	~10 frames/ sec	8%	10%	8%	10%	15%	20%
Video	~30 frames/ sec	20%	25%	20%	25%	35%	40%
Games	~5 frames/ sec	3%	5%	3%	5%	10%	15%

Figure 2: An example of application profiles

4.2. High Level Overview

Figure 1 shows the high level overview of the proposed architecture for interactive grids. It consists of submission nodes, Grid scheduler node, and execution nodes. An information service stores the information about the resources in the system. An application profiles repository contains the application profiles in the system. The distributed resource management framework is distributed across the submission nodes, Grid scheduler node, and execution nodes. The user submits the request through submission nodes, for a new global interactive session along with the set of applications desired to be launched through the global interactive session. The request for global interactive session is scheduled onto an execution node in the grid by the Grid scheduler. A global interactive session is then established between the selected execution node and the end-users' submission node. The end-user now submits requests for per application interactive sessions through this global interactive session.

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 Scheduler node, and a per-application session admission control module at the execution node. The Global and per-application admission control modules make admission control decisions for global and per-application sessions respectively. 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 resource requirements, session requirements, and the desired list of applications to be launched during the session. This request is submitted to the Grid Scheduler node.

2. The request is received by a Grid Scheduler running on the Grid Scheduler 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 [4]), 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 selects the best execution node that can admit the requested global interactive session satisfying the QoS requirements for the desired list of applications to be launched during the global session. During this step, the Grid Scheduler interfaces with the Global Admission Control system, which performs the admission, check for the requested global interactive session, [5].

4. A reservation is made on the selected execution node for the requested global interactive session. The reservation is also made for fine grained resources such as CPU, network bandwidth etc.

5. At the requested time, the selected execution node is allocated to the end-user, and the job dispatcher dispatches the request for the new global interactive session to the execution node along with the SLA for the session.

6. A configuration process configures the system before launching the global interactive session. This also involves the creation of a dynamic account by the Dynamic Account Manager. A global interactive session is then initiated between the allocated execution node and the end-users' submission node. 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. After the user has been authenticated, he may be authorized to use a normal static account if the gridmap-file has an entry mapping his identity obtained from his certificate during the authentication phase into this static account. If such an entry is missing, he may be assigned a dynamic account if the gridmap-file entry for his identity specifies a pool of dynamic accounts. Alternately the user's membership in a virtual organization (VO) may be verified by a directory service maintained by the VO. In that case, a dynamic account from the pool maintained for that VO can be assigned to the user. This approach is more scalable since every user joining or leaving a VO does not require the addition or deletion of a gridmap-file entry on all the resources made available to the VO. We can adopt a flexible approach of allowing the user to authenticate with a certificate that specifies the VO the user belongs to. The user's membership in the VO still needs to be verified by the VO's directory service. For further flexibility, we can assume that a community authorization service (CAS) [6] allows the user to authenticate to the resource with a restricted proxy certificate [7]. The policy specified in this restricted certificate can then be used to assign a dynamic account to the user, and customize the system policy files governing the dynamic account.

7. The end-user can now request for new per-application interactive sessions directly through the started global interactive session.

8. The requests for per-application interactive sessions are verified for access control checks, and if successful are passed onto the Session Admission Control system on the execution node.

9. 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.

10. The Resource Management Monitoring Agents monitor the global interactive session and per-application session utilization values. The monitored data is aggregated by aggregator agents. 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.

11. The enforcement agents end the global interactive session at the time specified in the SLA.

12. The execution node is now freed up to execute a new global interactive session if scheduled by the Grid Scheduler.

In the next few sections, we describe the resource management framework on the submission node, Grid scheduler node, and execution node. We focus mainly on the resource management framework on the execution node.

5. RESOURCE MANAGEMENT FRAMEWORK

5.1. Submission Node

The submission node contains the job submission client, which is responsible for submitting requests to the Grid scheduler node. There is also a session management agent that would co-ordinate with the resource management agents on the allocated execution node. This would be used, for example, for network bandwidth monitoring during application sessions [8].

5.2. Grid Scheduler Node

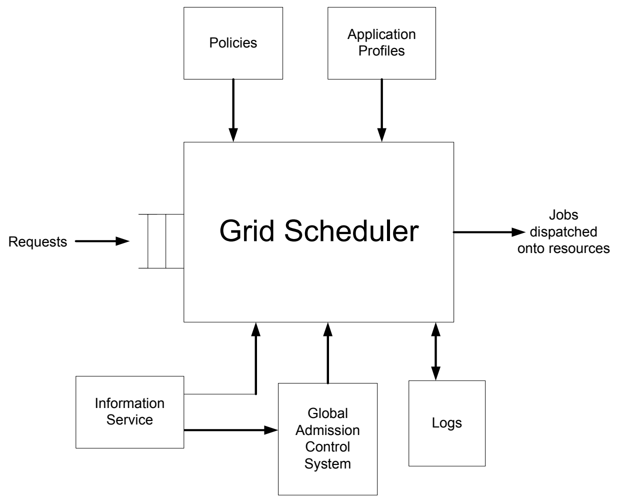


Figure 3: High level overview of the scheduler

The Grid scheduler node hosts the Grid scheduler. Figure 3 shows the high level structure of the Grid scheduler. The Grid scheduler accepts requests from the end-user. An Information Service maintains global information about all the resources in the Grid. This information is obtained by resource management agents distributed across the grid. The information is used by the Grid Scheduler while making scheduling decisions. In addition, the Grid scheduler is fed in with the global application profiles for the applications installed on the Grid. The Grid Scheduler also interfaces with a global admission control system, for making admission control decisions for admitting new global interactive sessions.

5.3. Execution Node

The resource management framework on execution node is responsible for providing QoS and SLA guarantees for per-application sessions, and the global interactive sessions, launched on this node. Figure 4 shows the resource management framework on the execution node. At a high level, this consists of resource management agents, a session admission control system, and an application predictor system. These are shown separately in Figure 5, and Figure 6. Figure 7 shows how some of the components co-ordinate and interact with each other. This co-ordination model is based on the producer-consumer paradigm [9]. 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 Application Predictor System, Enforcement Agents, and Session Admission Control System.

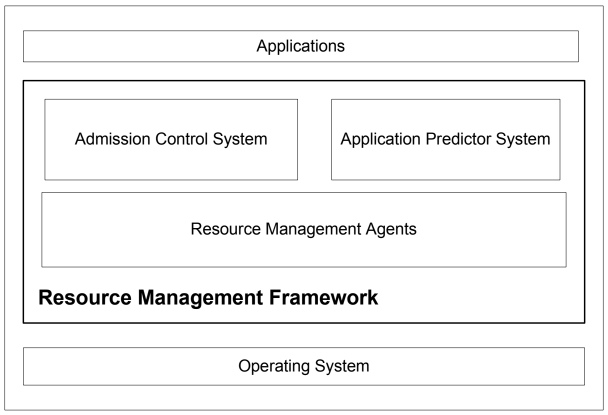

Applications
Admission Control System
Application Predictor System
Resource Management Agents
Resource Management Framework
Operating System

Figure 4: High level overview of the resource management framework on the execution node

The agent implementations could follow open standards like FIPA [10], [11]. The system can be extended to support a registry service, which would aid in supporting information publication about components, and discovery of components. Figure 7 shows these components residing on a single node. We now describe these components in detail in the following subsections.

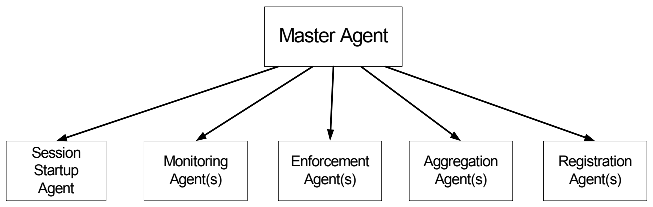


Figure 5: Resource management agents on the execution Node

5.3.1. Resource Management Agents

Figure 5 shows the agents on the execution node. We assume a master agent that is responsible for all of the agents on the execution node. We describe the agents below.

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.

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.

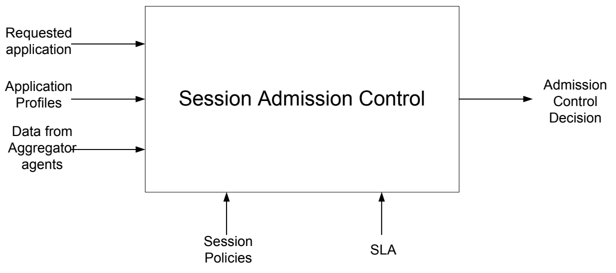

Requested application
Application Profiles
Data from Aggregator agents
Session Admission Control
Admission Control Decision
Session Policies
SLA

Figure 6: Session Admission Control System

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 [9] 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.

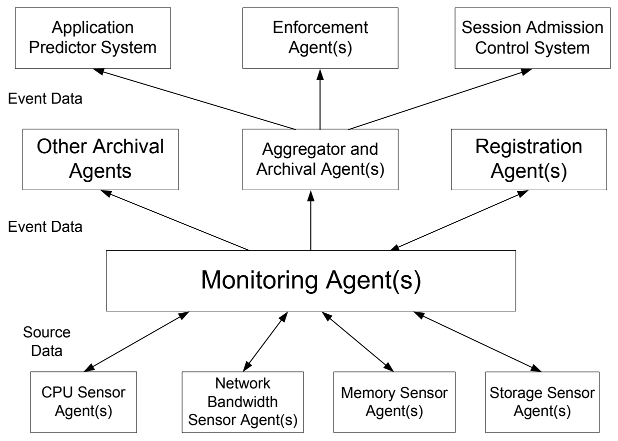


Figure 7: High level overview of the co-ordination model for resource management components on the execution node

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. This would allow compaction of data to minimize storage, as well application of filtering for interpreting data at various granularities. 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 periodicity 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. For example, this aggregation could be per time, per-process, or per-session based. Further, for each global interactive session, the total current session resource usage values can be determined like number of processes launched during the session, session wall-clock usage time, session CPU utilization, session bandwidth utilization, session storage utilization etc. The aggregated data is then archived into persistent storage. A prediction model could also be run on the data to obtain the forecasted resource utilization per global session, assuming current set of processes for this global session. 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 [9]. The event data format, and communication paradigm is implementation dependent.

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 periodicity 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.

Input: Application request, reservation agreement from SLA, data from Aggregator 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 Aggregator 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 8: An algorithm for SAC with CPU and network bandwidth utilization as the session parameters

Registration Agent

The Registration Agent behaves as a 'Consumer' to receive the resource usage value from the Monitoring Agent. Typically, the data of interest is that of overall CPU load. The periodicity for receiving the data is determined through policies, and is agreed upon in the subscription. The Registration Agent registers this information to a global information service so as to be used by the Grid Scheduler while making scheduling decision. The Registration agent could supply a prediction model to the Monitoring agent for forecasting the predicted CPU load based on load measurement history.

5.3.2. Session Admission Control System

A session admission control system (SAC) is responsible for determining if a global interactive session can admit a new per-application session. Figure 6 shows the admission control system. The inputs to a Session Admission Control system are:

(i) Requested application: The graphics application, which the user is requesting to be launched in the considered global interactive session. This request would be typically provided through a shell.

(ii) SLA: The Service Level Agreement for the global interactive session in progress. The SLA is determined prior to the start of the session.

(iii) Application profiles: Each resource has a copy of the application profiles for the applications installed on that resource.

(iv) Data from Aggregator agents: The aggregated resource usage data for this global session obtained from the aggregator agent using a query/response model. SAC could also supply the Aggregator agent with a prediction model for forecasting the resource utilization for this global session based on session load measurement history, and assuming current set of processes.

(v) Policies: The session policies in place for the session.

Given these inputs, the session admission control system checks for availability of resources in compliance to SLAs, before starting the requested application session. It checks the following global session parameters: (a) Number of processes launched during a session. (b) Usage time for a session. (c) Disk quota usage for a session. (d) CPU utilization percentage for a session. (e) Network bandwidth utilization percentage for a session.

The limiting values for these global session parameters are specified in the SLA for the considered global interactive session. SAC compares the current values of these global session parameters with the limiting values agreed upon in the SLA, for the considered global interactive session. 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, SAC makes an 'Allow' decision for the application. Figure 8 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 proposed a framework for SAC, and have presented a few session parameters that we envision to be necessary in an Interactive Grid computing system.

5.3.3. Application Predictor System

An Application predictor system predicts the QoS requirements for the applications and remote display servers to deliver an acceptable frame rate and performance to the end user. The application predictor system behaves as a 'Consumer' and receives the application resource usage data from the aggregator agents. The application predictor system applies a prediction model on the history of this data, to make its prediction. The newly made predictions are finally reflected in the application profiles. The application profiles are used by the enforcement agents and Session Admission Control System.

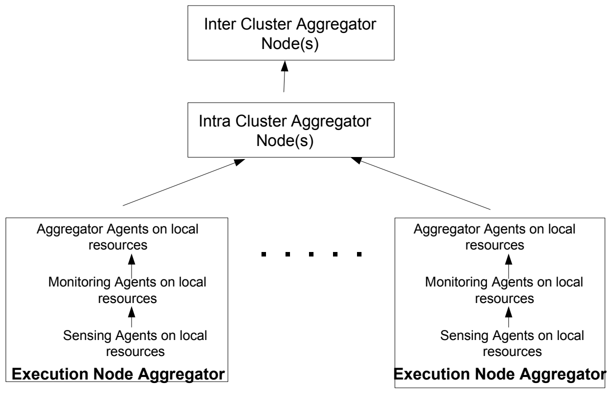

Inter Cluster Aggregator Node(s)
Intra Cluster Aggregator Node(s)
Aggregator Agents on local resources
Monitoring Agents on local resources
Sensing Agents on local resources
Execution Node Aggregator
. . . . .
Aggregator Agents on local resources
Monitoring Agents on local resources
Sensing Agents on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

Figure 9: Hierarchical aggregator nodes

5.4. Aggregator Node

The aggregator nodes host the aggregator agents. Figure 9 shows a hierarchy of aggregator nodes. The lowest level corresponds to execution nodes. The aggregator agents on the execution nodes 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.

6. RELATED WORK

The majority of the work in the area of grid computing has been for batch jobs. Recent projects on interactive applications like Crossgrid [12] do not address the problems and scenarios as presented in this paper. Our architecture leverages the architectural framework being defined in the Grid Monitoring Architecture [9]. Unlike the Grid Monitoring systems being developed [13], [14], [15], our monitoring infrastructure addresses the goal of providing QoS guarantees for graphics applications. We plan to leverage the low level hardware and software sensors to collect CPU, memory, network bandwidth measurement data. Other scheduling and resource management systems do not address supporting graphical interactive sessions on a Grid.

7. CONCLUSIONS

In this paper, we have presented a resource management framework for Interactive Grids. We presented the high level architectural framework across submission nodes, Grid scheduler node, and execution nodes. We focused on the framework on the execution node, consisting of an admission control system, application predictor system, and resource management agents. We introduced the notions of hierarchical admission control consisting of global admission control, and per-application session admission control. This implies one global session per user, and one or more per application sessions per user. Finally, we proposed a framework for monitoring, and managing a grid in terms of a hierarchy of agents.

8. REFERENCES

[1] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In International J. Supercomputer Applications, 15(3), 2001.
[2] T. Richardson, Q. Stafford-Fraser, K.R Wood, and A. Hooper. Virtual Network Computing. In IEEE Internet Computing, Vol 2. No. 1, pp. 33-38, Jan/Feb 1998.
[3]Vanish Talwar, Sujoy Basu, Raj Kumar. An Environment for Enabling Interactive Grids. Accepted in HPDC 2003, Seattle, Washington, June 22-24, 2003.
[4] Globus MDS. http://www.globus.org/mds/.
[5] P. Mundur, R. Simon, A. Sood. Integrated Admission Control in Hierarchical Video-on-Demand Systems. In IEEE International Conference on Multimedia Computing and Systems Volume I-Volume 1, June 07 - 11, 1999, Florence, Italy.
[6] L. Pearlman, V. Welch, I. Foster, C. Kesselman, S. Tuecke, A Community Authorization Service for Group Collaboration. In Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2002.
[7] S. Tuecke et. al., Internet X.509 Public Key Infrastructure Proxy Certificate Profile. In IETF PKIX Working Group Draft.
[8] Fabio Kon, Tomonori Yamane, Christopher Hess, Roy Campbell, and M. Dennis Mickunas. Dynamic Resource Management and Automatic Configuration of Distributed Component Systems. In Proceedings of the 6th USENIX Conference on Object-Oriented Technologies and Systems (COOTS 2001), pages 15 30, San Antonio, Texas, February 2001.
[9] R. Tierney et al. A Grid Monitoring Architecture. GGF Document series available from http://www.gridforum.org.
[10] FIPA. http://www.fipa.org/repository/index.html.
[11] Bigus, and Bigus, Constructing Intelligent Agents Using Java, 2nd edition, Wiley, 2001.
[12] CrossGrid. http://www.crossgrid.org.
[13] B.Tierney et al. A Monitoring Sensor Management System for Grid Environments. In HPDC-9, August 2000.
[14] A. Waheed et al. An Infrastructure for Monitoring and Management in Computational Grids. In Proceedings of 2000 Conference on Languages, Compilers, and Runtime Systems, 2000.
[15] R. Wolksi et al. The Network Weather Service: A Distributed Performance Forecasting Service for MetaComputing. In Future Generation Computing Systems, 1999.