Interactive Grid Architecture for Application Service Providers

Sujoy Basu, Vanish Talwar, Bikash Agarwalla[1], Raj Kumar
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto

HPL-2003-84 (R.1)
July 8th , 2003*

E-mail: {sujoy.basu, vanish.talwar, raj.kumar}@hp.com,bikash@cc.gatech.edu

interactive grid, middleware, controlled remote desktop, firewall, affinity scheduling

This paper presents our Interactive Grid architecture for Application Service Providers (I-GASP). We envision IGASP as a solution for making computers available primarily for interactive use in a grid computing environment. A user might access such a computer for running diverse applications such as graphics rendering, scientific visualization or mechanical CAD. I-GASP consists of a grid middleware for provisioning these computers, remote display technology that goes across firewalls and several techniques for making the computers suitable for use in a grid environment, namely controlled shell and desktop, dynamic accounts, admission control, and monitoring and management agents. To minimize the amount of user data that needs to migrate to the assigned computer before an interactive session begins, we present our affinity scheduling algorithm that favors a computer where the user has previously had an interactive session

* Internal Accession Date Only Approved for External Publication

[1] College of Computing, Georgia Tech, Atlanta, GA

Published in and presented at the International Conference on Web Services (ICWS '03) 23-26 June 2003, Las Vegas, Nevada

Interactive Grid Architecture for Application Service Providers

Sujoy Basu, Vanish Talwar, Bikash Agarwalla[*] Raj Kumar
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304, USA
{sujoy.basu,vanish.talwar,raj.kumar}@hp.com

Abstract

This paper presents our Interactive Grid architecture for Application Service Providers (I-GASP). We envision I-GASP as a solution for making computers available primarily for interactive use in a grid computing environment. A user might access such a computer for running diverse applications such as graphics rendering, scientific visualization or mechanical CAD. I-GASP consists of a grid middleware for provisioning these computers, remote display technology that goes across firewalls and several techniques for making the computers suitable for use in a grid environment, namely controlled shell and desktop, dynamic accounts, admission control, and monitoring and management agents. To minimize the amount of user data that needs to migrate to the assigned computer before an interactive session begins, we present our affinity scheduling algorithm that favors a computer where the user has previously had an interactive session.

1 Introduction

Application service providers (ASP) give customers the ability to use applications without owning the software or the infrastructure needed to run the applications. Typically we think of an ASP as providing access to complex business applications, consisting of a backend tier such as databases and legacy applications, a middle tier for business logic and a frontend for presentation logic such as HTML or WAP. These tiers are mapped to servers located in a data center. In this paper, we address the needs of an ASP that caters to a more diverse customer base. We include, in addition to traditional business customers, digital content creators, scientists doing visualization of large data sets, software developers and design engineers using electronic or mechanical CAD tools. These customers expect their primary applications, as well as other programming tools they typically use, to be installed on the computer. They also need access to shells (command interpreters) in addition to the graphical user interface (GUI) of their applications. Thus they need access to the full desktop of a computer to work productively. We believe that an ASP can provide these customers access to a remote *computer's desktop* for interactive use as a *service*. Users will normally have a computer in their office which is adequate for their work most of the time. However, during the phases of a project when computing cycles are the bottleneck, additional computers available through such a service can be invaluable. Besides, these users may need *occasional* access to a computer with a different processor, operating system or application suite or simply a more powerful computer. Using the service of an ASP is ideal for this case also.

Providing computing resources as a service for such interactive use to business users can also be done by their respective IT departments. However the service becomes more cost-effective and at the same time technically challenging when the service provider is a separate company. The business justification for such a service lies in the reduced cost to the enterprise purchasing the service. Rather than buying the computers and software licenses and keeping additional IT staff on their payroll to manage these resources, companies can pay for the service. An ASP can provide the service at a reduced cost because of the efficiencies of the workforce gained from specializing in the service, collocating the service for multiple customers in large data centers, maintaining higher utilization of resources, negotiating volume pricing discounts, etc. It is also helpful to think of such a service in the utility computing paradigm. Utility companies providing water and electricity give us a pay-per-use pricing model allowing us to use the provided resource only to the extent we require. Similarly, the company using the computing resource as service saves money by not over-provisioning for its computing needs. Instead of acquiring computers to satisfy the peak requirements of

[*] Work done during internship at HP Labs. Current affiliation: College of Computing, Georgia Tech, Atlanta, GA. Email: bikash@cc.gatech.edu

its product design cycle, maintaining enough computers for its average requirement is more economical. The rest can be obtained as a service as and when the need arises. However, since the service is provided by a separate company, several issues such as authentication, security, performance and business agreements for service level agreement (SLA), accounting and billing become more important. These are in addition to the resource management issues that need to be addressed whether the solution is provided internally by the IT department or externally by an ASP.

Many of these issues can be addressed by an ASP if it adopts a grid computing [9] environment. Traditionally, grid computing has addressed the needs of long-running scientific computations submitted as batch jobs. Indeed, some of the above mentioned customers may have long-running batch jobs that can be distributed across several nodes in a grid and benefit from parallel execution. But the benefits of grid computing extend to interactive applications also. Grids will provide resource virtualization based on standards developed as part of the Open Grid Services Architecture (OGSA) [8]. The customer will need to authenticate once to the grid. A digital content creator might then specify that he needs access to a computer with Itanium 2 processor running Linux, with the RenderMan application preinstalled. Resource virtualization implies in this scenario that the grid will be able to identify a computer, if available in the grid, that meets the requirements of the content creator. He does not need to specify a hostname or IP address. Following authentication and authorization, he is connected to the resource. For complete resource virtualization, we realize that customers cannot be expected to have accounts on every resource or at every site participating in the grid. Hence we address this issue by incorporating dynamic accounts in our design. A dynamic account on a computer is not associated permanently with a real-life user. Instead it can be assigned for the duration of an interactive session to a real-life user. The association is broken at the end of the session or later, when the system reclaims the dynamic account. It can then be assigned to another user.

Data management is another critical need that is addressed in grid computing environments. The customer's data could be located at the site of a storage service provider (SSP). Even though the SSP and ASP are different organizations, they can ensure interoperability by confirming to OGSA. On being connected to a computer at the ASP's site, the digital content creator can access his files, and have them cached locally at the ASP's site while his interactive session is in progress. In this work, we address the specific issue of caching his files locally with ownership given to the dynamic account assigned to him. We also investigate optimizations possible by assigning subsequent sessions to the same computer.

This paper is organized as follows. We present our requirement analysis in Section 2, followed by the overview of our system architecture in Section 3. Next, the architecture of a resource node is presented in Section 4. Data access is discussed in Section 5, with our affinity scheduling algorithm presented in Section 6. Then we present details of our implementation in Section 7. Related work is in Section 8. We conclude in Section 9.

2 Requirement Analysis

An important aspect of service management for any ASP is a Service Level Agreement (SLA) which uses Quality of Service (QoS) metrics as parameters in a binding business agreement with the customer. For interactive access to a remote computer's desktop, an important metric could be the response time observed for typical applications used by the customer. An application for computer-aided design of mechanical components might provide good performance if the delivered frame rate is at least 10 frames per second. If the computer running the application becomes overloaded, the frame rate can fall below the above threshold. Thus the ASP has to monitor continuously the performance delivered and apply techniques like restricting admission to applications on the remote computer and adjusting the priority of processes. The ASP also has to monitor the network conditions to determine when the problem is due to oversubscription to its network bandwidth.

Since the ASP is giving access to the desktop of a computer to a customer for running multimedia and graphics-rich applications, there are several precautions required against malicious users. Legitimate users will require access beyond the user interface of their applications, to the menus and taskbars of the desktop and to shells (command interpreters) where they can type commands. These are necessary for the engineers and content creators to work efficiently. However, malicious users can use the shell and other applications available through the desktop to probe the computer allocated to them for vulnerabilities. Security can be breached, and also denial-of-service attacks can be potentially launched. To thwart such attempts, we need a multi-tiered solution. As a first line of defense, a *controlled desktop* should allow only safe applications trusted by the ASP to be launched. We also need a *controlled shell* that allows only commands and their options validated by the ASP to be safe. In the next line of defense, we need monitoring agents on the resource to verify compliance to SLA parameters for the customer's session, such as CPU usage, network and disk bandwidth consumption and number of processes and open socket connections. Furthermore, suspect network activity and system calls should be blocked by these agents.

Next, we address the issue of application performance in the I-GASP environment. The graphics-rich engineering, scientific visualization and digital content creation applications in our target market segments have been witnessing an exponential growth in the complexity of the geometric models that have to be rendered on the screen. With such a fast growth in the amount of the data, it has become clear that only computers with high-end graphics pipelines and local access to the data set will have the capability to render such complex scenes. We observe that the ASP's customer will connect to the computer allocated to him from his desktop or mobile computer which may or may not have the ability to transfer large data sets and render at the required frame rate. This suggests that instead of sending raw geometric data for rendering on the client desktop, it is better to send pixels contained in the frame buffer of the allocated computer after rendering. This will allow us to easily support a heterogeneous client population. A reasonable frame rate can be sustained if the SLA ensures that adequate network bandwidth is available. Also, the application must be guaranteed adequate processing power.

At the end of Section 1, we have already mentioned some of the requirements in a grid computing environment such as resource virtualization, dynamic accounts and data management. These, as well as the requirements outlined in this section, are addressed by our system architecture.

3 System Architecture

Since we want our solution to be widely deployable, we expect the solution to work across firewalls at both the ASP site and the customer's site. Furthermore, we assume that the ASP's service is published as a web service in the Web Service Definition Language (WSDL). The initial request for service must be sent to this service access point accessible outside the ASP's firewall, and this communication should be compliant with the Open Grid Service Architecture (OGSA) [8].

Accordingly we consider a system architecture consisting of an Application Service Provider (ASP) site and a customer site, as shown in Figure 1. Some of the implementation details listed in this figure will be explained in Section 7. The arrows in Figure 1 are labelled in the order in which interaction between different components is initiated. The ASP site has several computing resources R that are available for interactive use. Together, they constitute resource pool RP. There is a firewall F1 that protects the ASP's resources. The customer first connects using the connection software S on his local computer L to the Grid Service Access Point GSAP which has been published externally by the ASP as a web service. From the GSAP, the request is forwarded to the Grid Distributed Resource Management system DRM. The latter must authenticate and authorize the customer based on Grid Security Infrastructure (GSI). Then it can match the resource requirements of the customer to available resources by querying the Grid Information Services GIS. The customer might request an immediate allocation or an advance reservation. At the scheduled time, the DRM system instructs the software agent SA on the selected resource R to start the remote display server RDS and connect to the communication server CS. The customer is then connected to CS using the remote display client RDC. CS facilitates communication between RDS and RDC in our scenario by having an open port in the firewall. RDC displays the desktop of R on the customer's local computer L.

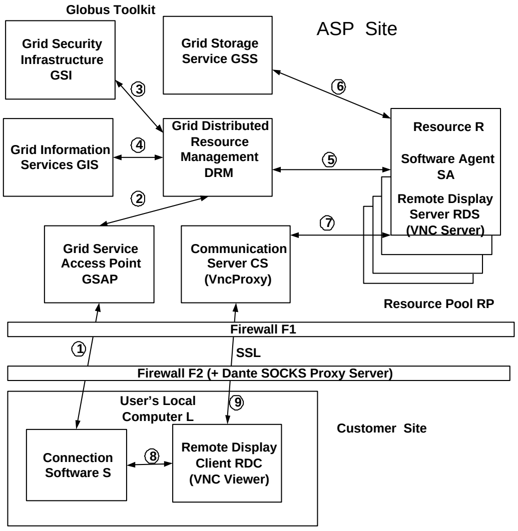
Globus Toolkit
ASP Site
Grid Security Infrastructure GSI
Grid Storage Service GSS
3
6
Grid Information Services GIS
4
Grid Distributed Resource Management DRM
5
Resource R
Software Agent SA
Remote Display Server RDS (VNC Server)
2
7
Grid Service Access Point GSAP
Communication Server CS (VncProxy)
Resource Pool RP
Firewall F1
1
SSL
Firewall F2 (+ Dante SOCKS Proxy Server)
User's Local Computer L
9
Customer Site
Connection Software S
8
Remote Display Client RDC (VNC Viewer)

Figure 1. I-GASP System Architecture

4 Resource Architecture

Each computer in the resource pool must be customized to meet the requirements specified in Section 2. Figure 2 shows the interaction among various components on a resource node that are relevant for this discussion. They are explained in detail in [22, 17]. The five main facets of our design are explained in the subsections below.

4.1 Controlled Shell and Desktop

Grid Interactive Shell (GISH) is a *controlled shell* that provides the first line of defense against malicious users. It also interfaces with Session Admission Control (SAC), described in Section 4.2, so that admission control can be enforced when an application is launched. Commands must belong to an allowed list of commands and runtime arguments before GISH will allow them to be executed. When the command line typed by the user is parsed by GISH, a sequence of checks can be done. In addition to specifying

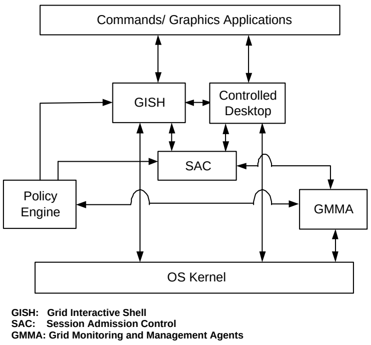
Commands/ Graphics Applications
GISH
Controlled Desktop
SAC
Policy Engine
GMMA
OS Kernel
GISH: Grid Interactive Shell
SAC: Session Admission Control
GMMA: Grid Monitoring and Management Agents

Figure 2. Resource Node Architecture

an allowed list of commands and their options, the system administrator can also customize the directories and files to which the user is allowed access. This might be used to prevent a grid user from looking at files which normal users have access to, but might be used for malicious intent by a grid user. Grid users will also have restrictions on programs compiled by them. In that case the program's source code can be submitted to the ASP for certification as safe. Then the compiled binary can be placed by the system administrator in a special directory, and added to the allowed list of commands. Alternately a virtual machine [3] with controlled access to disk and network can be provided, and the user can launch his binary within the virtual machine, without waiting for certification from the ASP.

In addition to being a shell for a *controlled user*, GISH can also be invoked as a shell for a *controlled superuser*. This privilege will of course be allowed only to users allowed to invoke the superuser shell command. This might be beneficial when certain trusted grid customers are given permission to assume root privilege for installation and updates for specific applications or daemons . In controlled superuser mode, GISH uses another list of allowed commands.

The *controlled desktop* has to be identical to the *controlled shell* in terms of the policies enforced. The desktop's menus and icons can be customized by a file that is owned by root, and the user is not given permission to add or modify menu items or icons. The user will assume that the applications available to him through the window manager are the only ones he is allowed to execute. To prevent the user from running an application he is not allowed to run, the group management feature of the OS can be used. This involves maintaining a separate group for potentially every application. Execution permission for an application is given only to group members. If dynamic accounts are used, the dynamic account must be added to all the application groups the user is allowed at the beginning of a session. When the dynamic account is finally reclaimed by the system, it must be removed from all the groups.

4.2 SAC

The Session Admission Control (SAC) module is called by GISH and the restricted desktop to determine whether an application selected by the user should be allowed to run. Data on current utilization of CPU, storage and bandwidth furnished by grid monitoring and management agents (GMMA), as well as historical data, determines whether the SLA can accommodate the additional load of the application requested.

4.3 GMMA

The Grid Monitoring and Management Agents (GMMA) monitor both session-specific and system-wide parameters and enforce policies. Monitoring agents monitor CPU and wall-clock time accumulated by a session, number of processes, open files and sockets, and suspicious network traffic, system call and command sequences. The management agents use the information gathered by monitoring agents as inputs to network-based and host-based intrusion detection systems. They can block the suspected activities. These agents can be the second line of defense against malicious users, with GISH being the first line of defense. The management agents are also responsible for enforcing QoS parameters.

4.4 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 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 by SAC or enforcement agents.

4.5 Dynamic Account Manager

We use dynamic or template accounts 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. For group accounts[1], we

[1] A group account is a group-wide common account shared by all the members of the group.

add the entry for the group once in this directory. Any site that participates in that grid will check the user's membership with the directory during authentication, and authorize the user as a dynamic account if she does not have a static account. The dynamic account is chosen from the pool of dynamic accounts maintained on each computer participating in the grid. Each dynamic account is a full-fledged Unix account created on the computer, but without a permanent real-world user associated with it.

Each pool is associated with a set of policies customized for 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 dynamic account is freed at the termination time agreed upon for the session that is using the dynamic account. At the termination time, GMMA agents kill the interactive processes still running with this account as owner , and archive all files owned by the account on a grid storage service using the user's grid credentials. The dynamic account is then returned to the pool. Subsequent sessions for this user retrieve the files from the archive. 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.

5 Data Access

The user's files have to be present on a filesystem accessible from the computer assigned to him. Furthermore, access rights for the user's files have to be given to the dynamic account assigned to him. In fact, the files may need to be transferred to the computer since the user's filesystem within his enterprise will not be visible on computers allocated by the ASP. Even if the files are visible, due to their being hosted by a Storage Service Provider (SSP), most likely these files will be transferred and locally cached by the ASP site to optimize performance. For interactive sessions on a computer allocated by the ASP, 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.

Since the transfer of files will often be necessary, we have assumed that the files are archived in the Grid Storage Service (GSS) in Figure 1. To minimize the time taken to transfer files at the beginning of an interactive session, it makes sense for the DRM to schedule interactive sessions with *affinity* to the computer used during the previous interactive sessions by the same user. Assuming that the computer 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.

6 Affinity Scheduling Algorithm

The steps involved in the algorithm for allocation of a computer with dynamic account, in response to a user's request for allocation sent to the grid middleware, are illustrated in a flowchart in Figure 3. Affinity scheduling is possible only for frequent users for whom a computer has been allocated at this site previously and is available. This is shown in step F. All other users are assigned computers 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 computers that are expected to be available during those time periods. After selection of the computer, a check is done on the computer 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.

7 Implementation

In Section 3, we presented an overview of our system architecture. Here we present details of our implementation. Our assumptions, mentioned in Section 3, and illustrated in Figure 1, include the presence of two firewalls. One protects the ASP's resources. The other has to be considered since the customer could be in an enterprise that protects its resources with a firewall.

We assume that the customer is inside a firewall that has a SOCKS [20] proxy server. The reason for choosing SOCKS is the flexibility of the SOCKS protocol, and its standardization by the IETF. SOCKS allows networking applications to communicate between hosts without direct IP reachability. We chose Dante [7] SOCKS server for our testbed. Note that no source code modification was needed in Dante. This validates our client application as being usable in any intranet configured for external access using a SOCKS server. If our client application is not made aware of the SOCKS server, it will work anywhere with direct IP connectivity to the ASP site. It will also work in intranets where hosts have their OS configured to send external traffic transparently to a SOCKS server. The client application, labelled Remote Display Client (RDC) in Figure 1, is VNC [18] viewer in our testbed.

We also assume that the ASP protects its resources with a firewall. However, in the demilitarized zone of the firewall, the ASP has set up a communication server as shown in Figure 1. In our testbed, this server is VncProxy [23], which implements an application-level proxy for the RFB protocol [19] implemented by VNC [18]. The remote display server, labelled RDS in Figure 1, is the VNC server in our testbed. Customers will expect the security and privacy of their connection with the ASP site. Hence we use the Secure Socket Layer (SSL) protocol. A X.509 certificate, based on Public Key Infrastructure (PKI), is issued to the communication server. SSL allows our VNC viewer to verify the communication server's identity based on this certificate, and then keep the communication encrypted using a session key shared between the viewer and communication server.

The changes required in the source code of VncProxy and the VNC viewer are minimal. VncProxy required only one change: the socket opened had to be changed to be an instance of the SSLSocket class in Java. Similarly, the VNC viewer (Java version) used in our testbed had to be modified to use *jsocks*, a Java SOCKS client library [15]. The socket opened by the viewer for connection to its server (VncProxy in our case) had to be modified to be a socksified socket, implemented as SocksSocket class by *jsocks*. Next a SSLSocket was opened by using the just created socket as a tunnel, through the SOCKS server, between the viewer and VncProxy. In our current implementation, we require only the VncProxy to prove its identity to the viewer using its certificate.

Most of the remaining components in our system architecture (Figure 1) are currently implemented [2] using the web services paradigm. We also have a more recent implementation [22], that uses the Globus Toolkit [10] Version 2. We would like to move this part of the system to Globus Toolkit Version 3, so that we can continue to use the web services paradigm, and remain compliant with Open Grid Service Architecture (OGSA) [8].

8 Related Work

The majority of the work in the area of grid computing has been for batch jobs. Recent projects like CrossGrid [6] are working towards extending the Grid infrastructure for interactive applications. However, they do not address the possibility of an interactive ”session” to remote nodes, and hence do not address the access control problems as outlined in this paper.

The Punch project [4, 16] has also addressed fine grained access control in Grid environments. Our work is more focused on graphics and multimedia-rich interactive applications. Hence we adopt a comprehensive access control and account management system for such environments, including the enforcement of QoS guarantees. Sudo [21] also allows restricted superuser privileges for running commands. However, unlike GISH, it does not provide the functionality of restricted superuser shell. Hence it cannot provide the fine-grain access control we need in a Grid environment.

Our work on monitoring and management agents can leverage existing work. For example, our GMMA agents can be interfaced to systems like Network Weather Service [24].

The remote display solution used in our implementation is VNC [18]. However, we can leverage other commercial products that go across firewalls such as [5, 11]. Our firewall traversal is strictly for the purpose of providing access to a remote computer's desktop to the customer. An orthogonal issue is connecting the grid middleware running at the ASP site, or any other site behind a corporate firewall, to a grid site at an academic institution. A solution for this problem using SSH-tunneling has been demonstrated in [12].

Dynamic account management has been described in [14, 1]. We use [13] as a starting point for our implementation. However we differ from the prior work in using the dynamic account as a component of our policy-driven customizable grid environment. The same dynamic account will give different access rights and QoS guarantees depending on the user to whom it has been assigned.

9 Conclusions

We have presented I-GASP, our Interactive Grid Architecture for Application Service Providers. It consists of a Grid middleware that can be used by the ASP for provisioning computers. Remote display technology that goes across firewalls is another component of our architecture. We also need several techniques for making the computer suitable for use in a grid environment, namely controlled shell and desktop, dynamic accounts, admission control, and monitoring and management agents. Finally we present our affinity scheduling algorithm that favors a computer where the user has previously had an interactive session. This minimizes the amount of user data that must be migrated to the assigned computer before an interactive session begins.

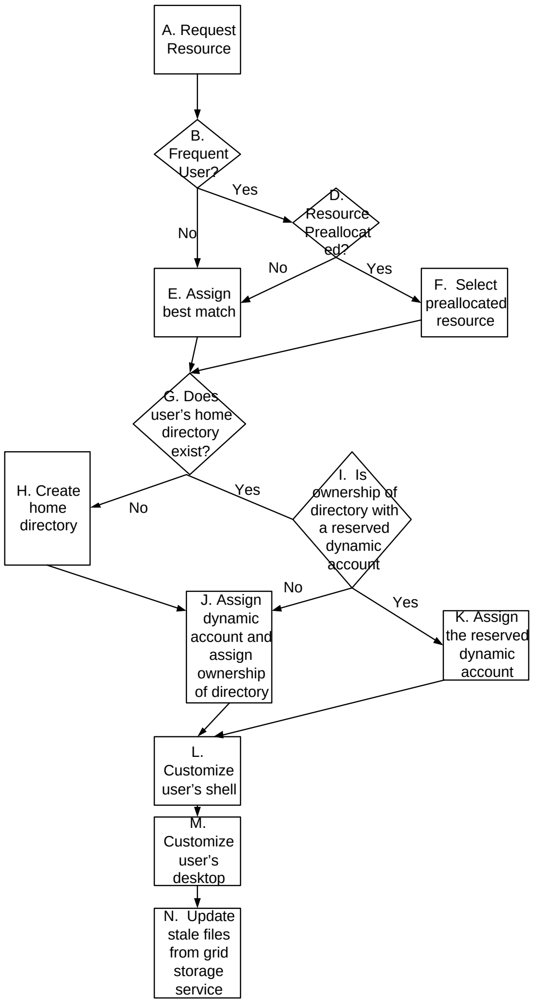
A. Request Resource
B. Frequent User?
Yes
No
D. Resource Preallocated?
No
Yes
E. Assign best match
F. Select preallocated resource
G. Does user's home directory exist?
H. Create home directory
No
Yes
I. Is ownership of directory with a reserved dynamic account
No
Yes
J. Assign dynamic account and assign 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 3. Flowchart describing affinity scheduling algorithm.

References

[1] C. Anglano, S.Barale, L.Gaido, A.Guarise, S.Lusso, and A.Werbrouck. An Accounting System for the Datagrid Project version 3.0. http://server11.infn.it/workload-grid/docs/DataGrid-01-TED-0115-3_0.pdf .

[2] S. Basu, S. Adhikari, R. Kumar, Y. Yan, R. Hochmuth, and B. E. Blaho. mmgrid: Distributed resource management infrastructure for multimedia applications. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS)*, Apr. 2003.

[3] E. Bugnion, S. Devine, K. Govil, and M. Rosenblum. Disco: Running commodity operating systems on scalable multiprocessors. *ACM Transactions on Computer Systems*, 15(4):412–447, 1997.

[4] A. Butt, S. Adabala, N. Kapadia, R. Figueiredo, and J. Fortes. Fine-Grain Access Control for Securing Shared Resources in Computational Grids. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS)*, apr 2002.

[5] Citrix. http://www.citrix.com.

[6] Crossgrid. http://www.crossgrid.org.

[7] Dante. http://www.inet.no/dante/.

[8] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, June 22 2002. Available at http://www.globus.org/research/papers.html#OGSA.

[9] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001. Available at http://www.globus.org/research/papers.html#anatomy.

[10] Globus Toolkit. http://www.globus.org/toolkit/.

[11] GoToMyPC. http://www.gotomypc.com.

[12] S. Graupner and C. Reimann. Globus Grid and Firewalls: Issues in a Utility Data Center Environment. Technical Report HPL-2002-278, HP Labs, 2002. Available from http://www.hpl.hp.com/techreports/2002/.

[13] http://www.gridpp.ac.uk/gridmapdir/.

[14] T. Hacker and B.D.Athey. A Methodology for Account Management in Grid Computing Environments. In *Proceedings of the 2nd International Workshop on Grid Computing (GRID 2001), Available as Lecture Notes in Computer Science (LNCS) Volume 2242*, nov 2001.

[15] JAVA SOCKS Server. http://jsocks.sourceforge.net/.

[16] N. H. Kapadia and J. A. B. Fortes. Punch: An architecture for web-enabled wide-area network-computing. *Cluster Computing: The Journal of Networks, Software Tools and Applications; special issue on High Performance Distributed Computing*, 2(2):153–164, sep 1999.

[17] R. Kumar, V. Talwar, and S. Basu. A resource management framework for interactive grids. In *Proceedings of the Workshop on Middleware for Grid Computing (MGC)*, June 2003.

[18] T. Richardson, Q. Stafford-Fraser, K. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, Jan/Feb 1998.

[19] T. Richardson and K. Wood. The rfb protocol, version 3.3, 16 July 1998. Available at http://www.uk.research.att.com/vnc/protocol.html.

[20] SOCKS Protocol. http://www.socks.permeo.com/.

[21] Sudo. http://www.courtesan.com/sudo.

[22] V. Talwar, S. Basu, and R. Kumar. An environment for enabling interactive grids. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*, jun 2003.

[23] VNC Proxy. http://www.wilson.co.uk/Software/vnc/proxy/VncProxy.htm.

[24] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance. *Future Generation Computer Systems*, 15(5-6):757–768, 1999.