The Design and Prototype of RUDA, a Distributed Grid Accounting System

M.L. Chen, A. Geist, D.E. Bernholdt, K. Chanchio, and D.L. Million

Oak Ridge National Laboratory, Oak Ridge, TN, USA
{chenml, gst, bernholdtde, chanchiok, milliondl}@ornl.gov

Abstract. The Grid environment contains a large and growing number of widely distributed sites with heterogeneous resources. It is a great challenge to dynamically manage and account for usage data of Grid resources, such as computational, network, and storage resources. A distributed Resource Usage Data management and Accounting system (RUDA) is designed to perform accounting in the Grid environment. RUDA utilizes fully decentralized design to enhance scalability and supports heterogeneous resources with no significant impact on local systems. It can easily be integrated into Grid infrastructures and maintains the integrity of the Grid security features.

1 Introduction

Modern science and technology are increasingly collaborative and often demand huge computing and storage resources, which individual research organizations may not possess. A distributed infrastructure - Grids - was created in early 2000. Grid technology, such as the protocols and services developed by Globus [1], enables flexible, controlled resource sharing on a large scale. Driven by the demand and attracted by the promising future of Grids, Grid applications multiply swiftly and in turn drive the rapid development of Grid technology. While many Grid services are maturing, Grid accounting still remains a research issue. The Grid environment, which contains a large and growing number of widely distributed sites with heterogeneous resources, poses great challenges to Grid accounting. Rapid evolution of Grid software infrastructures and increasing security concerns add additional complications. Therefore, a Grid accounting system needs to be scalable, flexible, and secure [2]. Though many methods and tools have been successfully used in individual sites for resource usage management and accounting [3-4], they are local, centralized systems, of which the scalability is limited by the load capability and data size of the centralized server and database. These accounting methods and tools do not satisfy the requirements of Grid accounting.

The Science Grid (SG) project is a collaboration involving researchers at Lawrence Berkeley (LBNL/NERSC), Pacific Northwest (PNNL), Argonne (ANL), and Oak Ridge (ORNL) National Laboratories, sponsored by the U.S. Department of Energy (DOE), to explore the Grid environment for use with the large-scale computer and storage systems available at DOE sites. The ability to properly account for resource

O. Gervasi et al. (Eds.): ICCSA 2005, LNCS 3482, pp. 29 – 38, 2005.

usage across such a Grid is crucial to its acceptance and use. To the best of our knowledge at the time of this project started, there was no Grid accounting system that could meet the challenges of the Grid environment.

We have designed and prototyped a Grid Resource Usage Data management and Accounting system (RUDA). RUDA is designed in a fully distributed manner for scalability. It employs customizable interfaces to communicate with local systems to support widely diversified resources without significant impact on them. RUDA leverages security features embedded in Globus for secure wide-area communication, and accommodates current economic models of the Grid with respect to valuation of resources. RUDA is an essentially self-contained system that can be easily integrated into the Grid environment.

The following three sections present the system architecture, the accounting process, the feature design targets and approaches. In later sections, the implementation and experiment of the prototype are discussed.

2 System Architecture

In the Grid environment for which RUDA is designed, resources are provided by computer centers or divisions of individual sites. The users are organized in research projects and each project has charge-account(s) used by the providers to credit/charge the allocations of the project. The allocation in a charge-account may be distributed to the individual users who share this account.

The basic building-block of RUDA is a client/server software package. The multithreaded server consists of core-software and interfaces. The latter collects resource usage data from local accounting systems and the former, isolated from the local system, provides data management, accounting, and web interface services. A RUDA server can be configured in two running modes, called *basic-server* and *head-server* respectively. The client-process provides the services for users/administrators to access/control the server and for other servers in RUDA to communicate with this server. The client-process resides on the same machine with its server. The Globus Toolkit's GRAM [5] commands are employed to remotely run the client-process to communicate with the server from any computer on the Grid.

In RUDA, a *basic-server* daemon runs on each participating resource and periodically pulls resource usage information of Grid users from the local accounting system. It manages and accounts for the resource usage data, and stores the records in a local MySQL database. *Head-servers* are used to aggregate accounting and usage information for organizations, projects, and other interested entities; they can be deployed on any computer on the Grid. The *head-server* is configured to select the "member" *head-* or *basic-servers*, which contain the resource usage data of interest, and the *head-server* queries desired data segments from each member server by running the client-process of the member remotely with criteria for the specified projects, charge-accounts, and users. The member server writes the requested data set into a RUDA standard data file. Transferring the data back by GridFTP [5], the *head-*

server manages and accounts for the data collectively. Since a *head-server* can collect data from both *basic-servers* and other *head-server*s, a RUDA system with flexible hierarchical structures can be built for large organizations to perform accounting.

RUDA also provides allocation services. In the DOE community, allocation on major resources requires pre-approval of authorized administrators. RUDA server provides web interfaces for project Principle Investigators (PIs) to check available resources and to apply for allocation.

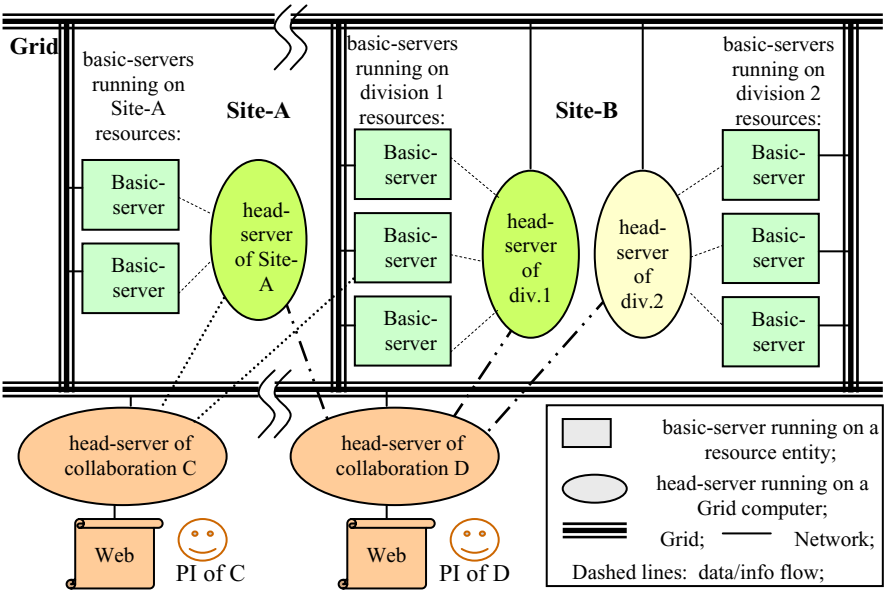

Fig. 1. A simple example of RUDA system

Figure 1 shows a much simplified RUDA system example. Independent and geographically separated sites A and B provide computer resources to the Grid. A *basic-server* on each individual computer collects data and accounts for the Grid resource usage at the levels of job, user, and charge-account. The resource providers, Site-A and two divisions at Site-B, employ *head-server*s to manage and account for their resource usage. Each *head-server* collects data from its member servers and performs accounting collectively at the levels of user and charge-account. These *head-server*s need not to collect detailed data at job level unless desired. Projects C and D use computer resources at both Site-A and Site-B. Each PI runs a *head-server* and configures relevant servers as its members. This also provides a simple example of the hierarchical structure of the system.

3 Accounting Process

3.1 Standard Resource Usage Records and Global User Identifications

Grid accounting systems must manage and exchange resource usage data globally, however there are currently no standards for this type of data. For example, a survey [6] shows that more than 15 technical terms are used for CPU usage records by various local accounting systems in 5 national laboratories, and each of them has its own definition and data type. To perform Grid accounting, a standard set of usage record fields has to be defined and a mechanism to convert the local fields into the standard fields needs to be developed. A standard set of resource usage fields has been defined for RUDA based on the Global Grid Forum (GGF) suggestions, a survey of DOE laboratories [6,8], and a more detailed examination of local accounting systems used at ORNL, PNNL, and LBNL/NERSC. A data structure called *data-cargo* accommodates the standard fields within RUDA servers and a corresponding file format is used when exchanging records within a distributed RUDA deployment. For each new type of local accounting system, a customized reference table needs to be setup to allow the standardization routine of RUDA server interface to convert the local system's input records into a standard data set and store them in a *data-cargo* structure.

Grid-wide user identification is also critical for Gird accounting. One user may use different user identifications (IDs) on different computers. To uniquely identify users Grid-wide, rules to create global user IDs have to be defined.

RUDA utilizes the Grid user distinguished name (GUdn) as its global user ID. GUdn is defined by Grid Security Infrastructure (GSI) [5] of Globus. For authentication purpose, GSI grants each user a user-certificate, which contains a globally unique GUdn. For access control purpose, each resource has a grid-map file to map each GUdn with local user ID (LUid). Using information extracted from the grid-map file, a RUDA *basic-server* can pair up LUids with GUdns. The standard data record contains both GUdn and LUid for user identification.

When the customized interface of *basic-server* reads data in, it calls the standardization routine to covert the local data into a standard data set, pairs up each GUdn with LUid, and fills the standard data set into a *data-cargo* structure which is ready to be transferred to RUDA server for usage data normalization and accounting.

3.2 Usage Data Normalization and Accounting

Various Grid resources carry different qualities of service and different capabilities. The ability to normalize accounting data across diverse Grid resources is important to the Grid "economy" [2, 9], especially if user resource allocations are fungible across multiple resources. RUDA provides the flexibility to support economy-based pricing of Grid resources. RUDA defines a "RUDA-allocation-unit (RAU$)" as the standard charge unit and each server performs accounting for each job according to a customized formula. The formula currently used in the prototype is

$$total\_charge = \left[\sum_{i=1}^{n} (Usage\_amount(i)) \times (weight(i)) \right] \times \prod_{j=1}^{m} (global\_weight(j))$$

Here, n is the total number of resource categories of the standard resource usage data set and i spanning from 1 to n represents category index. *Weight(i)* presents the weight factor of resource category i. m is the total number of the global weight factors. The *global_weight(j)* is j'th global weight factor, such as the priority or quality of service, usage timing (peak, off-peak), or any factors which effect the overall cost of a job.

Referring the "Demand and supply model" suggested by [2], the weight factors of each resource can be configured by its provider independently according to their site policy and user demand, though the Grid administrator may publish a list of weight factors for a number of commonly used resources as reference. Grid economy research shows that this model would regulate the resource demand and supply automatically, give each site maximum control over the price, and eliminate the necessity of centralized resource evaluation. The formula and the weight factor information are stored in the server's database and available for users upon request.

The *basic-server* on each resource records the usage data and charge of each running job in a dynamic data structure. Upon the completion of a job, its data are moved into a local database. Based on the data of both current and completed jobs, the server calculates the individual category resource usage and total charges accumulated since the beginning of current accounting period for each user, charge-account, and project respectively. By means of *head-server*(s), PIs or users collect the relevant data from the resources they use and account aggregately for their dynamical Grid resource usage and total charges.

4 Feature Design Targets and Approaches

4.1 Scalability

The large and growing scale of Grid environments poses great challenges to the accounting system design. Instead of technically enhancing the capability of the centralized server and database, RUDA takes a distributed approach that focuses on maximally decoupling the loads of the server and database from the scale of Grid. The data collection and storage are performed locally by a *basic-server* on each resource entity. Therefore the *basic-server* load and its database size are independent of the Grid scale. *Head-server*s perform usage data management and accounting for projects or providers. A *head-server* for a provider manages the data on the resources they provided. A *head-server* for a PI only sees the data relevant to his/her project or group. The data collected/managed by a *head-server* and stored in its database depends on the size of the project/group or the resource provider's environment, not on the scale of the Grid. This decoupling strategy eases the scalability limitation caused by database size and server load without requiring breakthrough technologies. The flexible *head-server* architecture also enhances scalability, since servers can be deployed as needed on a per-project, per-provider, or other basis. It is only necessary

for a *head-server* to know which other servers to poll to obtain the desired resource usage information. In the DOE laboratory environment, this is fairly straightforward, since accounts and allocations are typically tracked and already known to the providers/PIs.

4.2 Security

Significant efforts in Grid software development have gone into secure network communications and remote interactions. The Globus Grid Security Infrastructure (GSI) is based on existing standard protocols and APIs [5] and extends these standards to single sign-on and delegation, forming a comprehensive security system [10]. RUDA employs the Globus toolkit, and GSI in particular, to enable secure authentication and communication over the open network. For instance, to collect data from a remote *basic-server*, a *head-server* issues a GRAM command to call the *basic-server*'s client-process remotely for data query. The Globus servers on both ends then handle mutual authentication and secure remote communication for the RUDA servers. During the data transfer procedure, GridFTP manages the mutual authentication and insures the communication integrity. In this way, RUDA is based upon standard Globus security features.

The authorization for data access is performed by the RUDA server in two layers. One layer is applied to authorize a *basic-server* collecting data from the local system. Each *basic-server* owns a user/project map provided by the local administrator through a configuration file. The map lists the users and projects of which the data is allowed for RUDA to access, and the server interface limits queries to the authorized data only. The other layer is control of the access to the RUDA server's database. The database contains the user attributes originally obtained from the local system. When a remote user queries for data, the server performs the data access control according to the user status. For example, an ordinary user can only access his/her own data, while a PI can access the data of their entire project/group.

4.3 Fault Tolerance

We have chosen a fail-over mechanism as a short-term solution to fault tolerance. We run backup daemons on backup machines, which periodically probe the existence of running daemons. A backup daemon of *basic-server* also keeps a copy of the current resource usage data of the running daemon. Once a failure is detected, the backup daemon can conclude that the original daemon or machine has died and inform the other daemons that are interacting with the original daemon that it has taken over. Simultaneously, the backup daemon informs the system administrators that the fail-over has occurred. The backup daemon of a *basic-server* only provides the latest accounting data upon request. The administrator is expected to troubleshoot and recover the original server or machine as soon as receiving the failure information.

Although the above mechanism is easy to implement and sufficient for a small group of RUDA daemons, it can not distinguish network partition from machine failure, handle simultaneous failures of the original and backup daemons, or deal with

the complications in a large distributed environment. We have chosen the group membership management mechanisms as the long-term approach to fault tolerance [11, 12]. In such a system, the RUDA daemons will monitor one another and operate self-healing mechanisms once they agree that a group member has failed.

4.4 Flexibility and Manageability

RUDA's design includes interfaces that can be customized to communicate with local accounting systems, allowing RUDA to support heterogeneous resources with various local accounting systems with a minimum local impact. To avoid the complications caused by the modification of Grid software infrastructure, RUDA software is built to be essentially self-contained. The utilization of Grid infrastructure is limited to its high level APIs and the modifications behind the APIs have no effect on RUDA. Furthermore, the APIs are called in a couple of customization routines. By modifying the customization routines, RUDA can utilize various versions of Globus or other Grid infrastructures.

The RUDA server is fully configurable, such as its user/project map, data polling period, data backup method, and so on, and supports runtime reconfiguration. By means of GRAM and RUDA command line interfaces, the administrator can configure and control the server remotely from any computer on the Grid.

5 Prototype Implementation

A prototype of RUDA has been developed on an SG testbed. It contains most of the major components of RUDA to test the feasibility of RUDA design, but the fail-over mechanism is not implemented due to lack of backup machines. The functions of prototype's client/server package are briefly described in the following.

The server chooses one of the three data collection interfaces (Fig. 2) according to its configuration. The interface shown on the left side of Fig. 2 accepts RUDA standard data files transferred between RUDA servers. The one in the middle is the major data input port of *basic-server*. By means of the customized routine, the interface inputs data from a local accounting system through CLIs/APIs provided by the system, converts them into a standard data set, and loads the server's *data-cargo*. The interface on the right side is designed for situations where the additional security requirements are imposed. To secure sensitive information, some sites do not allow foreign access to the local systems and sensitive information must be filtered out before resource usage data reach the Grid. This interface provides an independent daemon process run by authorized site administrators. To periodically pull data from the local system, the daemon calls a customized routine, which filters out the sensitive information and coverts the local data into the standard data set. The daemon then loads the data into its MySQL database, which RUDA server is authorized to access. On receiving a *new-data-ready* signal from the daemon, the server reads in the data from the daemon database.

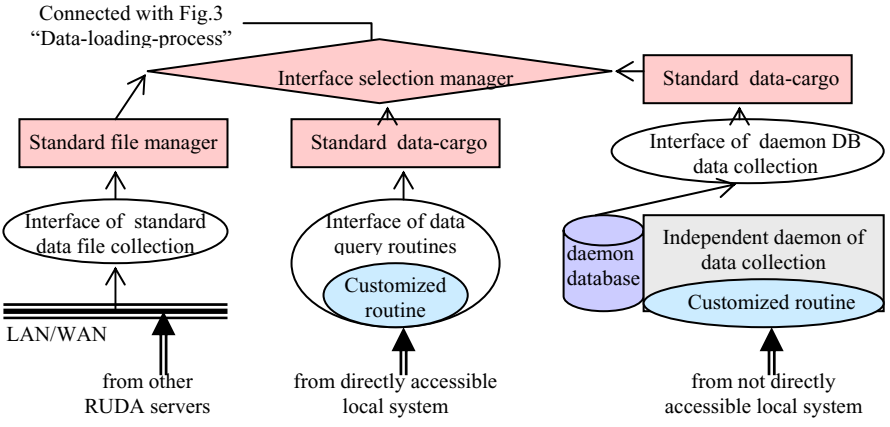

Fig. 2. RUDA server block diagram part 1: Server interfaces

Upon receiving new data, the server core-software (Fig. 3) utilizes the customized formula shown in Section 3.2 to calculate the charge of individual jobs, and summarizes the usage data and charges from both current jobs and the jobs completed within the accounting period for users, charge-accounts, and projects. The updated current data are stored in server's data structure and also copied into a local database called mirror-site, which can be used to recover the current data in case the need arises.

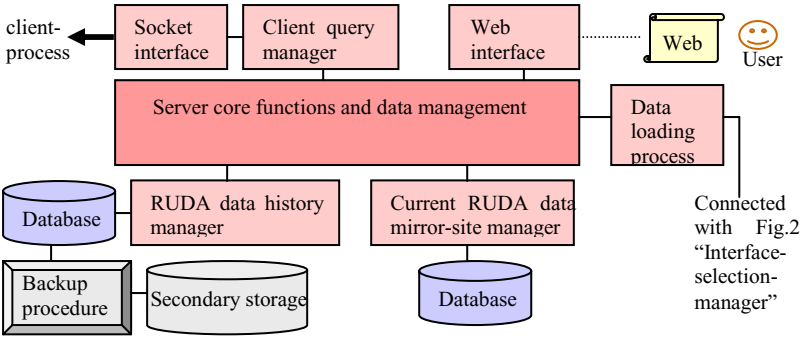

Fig. 3. RUDA server block diagram part 2: Server core-software

The client-process only communicates with the server through local socket connections. The Globus toolkit GRAM is employed to run the client-process from a computer on the Grid to perform remote communications with the server.

The prototype provides web interfaces for users to conveniently access their resource usage data and for PIs to check resource availability and apply allocations.

The command line interfaces are also provided for users query resource usage data, and for administrators to configure and manage the server locally or remotely.

6 Experimental Results

The prototype system has been experimented on the Earth System Grid (ESG). The ESG project uses Grid technology to support the climate research community to discover, access, and analyze large-scale global climate model simulation results in a distributed and heterogeneous computational environment. The RUDA prototype has been deployed on ESG computer resources at ORNL in Tennessee and National Center for Atmospheric Research in Colorado. The computer platforms include IBM AIX, Sun Solaris, and Linux redhat, on which the RUDA software is portable. Globus is the software infrastructure of ESG. GRAM and GridFtp of Globus Toolkit (version 2.2.4) with embedded GSI are enabled on these machines, and MySQL and Apache web server are available.

A total of five *basic-server* daemons ran in this experiment on a mixture of IBM AIX, Sun Solaris, and Redhat Linux systems, and a *head-server* daemon running on an IBM AIX machine configured them as its members. The average number of accounted jobs running simultaneously on each machine was at a level of a few tens to a hundred at any moment, with a duration varying from a few minutes to several days. Though a full set of standard resource usage fields had been defined in the prototype, only CPU time, wall time, and the memory usage (requested or high-water-mark) were captured in this experiment. Artificial weight factors (see section 3.2) were assigned to simplify the process of checking accounting results.

The experiment first ran for four weeks and concentrated on checking server functions. During this period, an error was reported by the *head-server* on data transfer from one of its member servers and it was caused by the downtime of that remote member computer. As mentioned, the fail-over mechanism has not been implemented. When a member server is down, the *head-server* uses the latest data set collected from that server as current data (with original data collection timestamp) until the member server is up and running again. The memory and CPU usage of the servers were also measured. According to the SG project's survey of resource providers and users, updating resource usage data and accounting records every hour would fully satisfy the requirement of dynamic accounting. With this configuration, the CPU time was less than 190 seconds per day for the *head-server*, and 10 seconds for each *basic-server*, respectively. The memory usage (high-water-mark) of all individual servers is less than 3 MB.

The experiment then continuously ran for 12 more weeks. All servers were configured to collect data every 4 minutes for experimental purposes. The snapshots of resource usage data and accounting results were taken and checked on a daily basis. All servers ran smoothly through the whole period and no accounting error was found. The RUDA web interfaces were used daily to monitor the system. The allocation application functions were also tested through the web interface, though the responses of administrators were performed automatically by a piece of simulation software.

7 Conclusion and Acknowledgement

A Grid Resource Usage Data management and Accounting system, RUDA, has been designed for the DOE Science Grid project. A prototype RUDA system has been implemented and tested to demonstrate the feasibility of the design. With more customized data collection routines, RUDA can be deployed onto computers with various local accounting systems and onto other types of resources such as mass storage systems. We are planning to perform further experiments on a larger RUDA system, with fault tolerance features implemented, in the near future.

We thank Scott Jackson and David Cowley at PNNL, the ORNL local accounting system development group at ORNL and East Tennessee State University, and Francesca Verdier at LBNL/NERSC for their support of our local accounting system investigation. The support from ESG project on the RUDA experimental deployment is also very much appreciated.

Science Grid project is sponsored by the U.S. DOE Office of Science under the auspices of the Scientific Discovery through Advanced Computing program (SciDAC). ORNL is managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

1. Foster, I. & Kesselman, C. Globus, “A Toolkit-Based Grid Architecture”, ibid, p. 278, Morgan-Kaufmann (1999).
2. Bill Thigpen & Tom Hacker, “Distributed Accounting on the GRID”, Global Grid Forum (GGF), 4-7 March 2001.
3. http://www.emsl.pnl.gov/docs/mscf/qbank
4. http://www.nersc.gov/nusers/accounts/nim
5. http://www-unix.globus.org/toolkit
6. Mi young Koo, “Usage Record Fields – Survey Results and Proposed Minimum Set”, GGF Oct., 2002
7. http://www.gridforum.org
8. Laura F. McGinnis, “Resource Accounting – Current Practices”, GGF Feb., 2001.
9. Rajkumar Buyya, David Abramson, and Jonathan Giddy, “A Case for Economy Grid Architecture for Service Oriented Grid Computing”, 10th Heterogeneous Computing Workshop, Apr. 23-27, 2001
10. http://www.globus.org/security
11. M. Franceschiti and J. Bruck, “A Group Membership Algorithm with a Practical Specification,” IEEE Transactions on Parallel and Distributed Systems, vol 12, no 11, 2001
12. K. Chanchio, A. Geist, M.L. Chen, “A Leader-based Group Membership Protocol for Fault-tolerant Distributed Computing”, submitted to the 14th International Heterogeneous Computing Workshop, 2005