Developing Resource Usage Service in WLCG

Xiaoyu Chen, Akram Khan, *Member, IEEE*, Gary B. Wills and Lester H. Gilbert

***Abstract*— According to the Memorandum of Understanding (MoU) of the World-wide LHC Computing Grid (WLCG) project, participating sites are required to provide resource usage or accounting data to the Grid Operational Centre (GOC) to enrich the understanding of how shared resources are used, and to provide information for improving the effectiveness of resource allocation. As a multi-grid environment, the accounting process of WLCG is currently enabled by four accounting systems, each of which was developed independently by constituent grid projects. These accounting systems were designed and implemented based on project-specific local understanding of requirements, and therefore lack interoperability. In order to automate the accounting process in WLCG, three transportation methods are being introduced for streaming accounting data metered by heterogeneous accounting systems into GOC at Rutherford Appleton Laboratory (RAL) in the UK, where accounting data are aggregated and accumulated throughout the year. These transportation methods, however, were introduced on a per accounting-system basis, i.e. targeting at a particular accounting system, making them hard to reuse and customize to new requirements. This paper presents the design of WLCG-RUS system, a standards-compatible solution providing a consistent process for streaming resource usage data across various accounting systems, while ensuring interoperability, portability, and customization.**

***Index Terms*—Aggregate accounting, Enabling Grids for E-sciencE, Grid accounting, Large Hadron Collider, Open Grid Forum, Resource Usage Service, Usage Record, Worldwide LHC Computing Grid.**

I. Introduction

Accounting in the grid, also known as grid accounting, plays an important role in system administration, resource usage policing, and supporting grid economic models. The main purpose of grid accounting is to meter shared computing resources and to supply usage information in a grid environment. Collective usage information enriches system administrators' understanding and enhances overall resource utilization in a grid system. For most e-Science grids, computing resources are provided by academic institutions for one or more collaborative and non-commercial research projects. Individual projects and participants are granted fixed quotas for resources such as such as computational cycles and storage space. Accounting in such e-Science grid environments enables usage management that prevents grid resources from overexploitation by checking actual resource usage against allocated resource quotas. Resources and services managed within a commercial grid system are utilized on a "pay-per-use" basis. Accounting in this case is mainly used for authorization and provision of usage proof for charging users based on actual resource usage. In addition, grid accounting supports the management of security, Quality of Service (QoS), etc.

A number of grid accounting systems have been developed and deployed. In the Open Science Grid (OSG) [1] project, an accounting system called Gratia [2] operates at each participating site. Accounting Processor for Event Logs (APEL) [3] and Distributed Grid Accounting System (DGAS) [4] are two accounting systems developed by the World-wide LHC Computing Grid (WLCG) [5] and the Enabling Grids for E-sciencE (EGEE) [6] projects. These two accounting systems became part of the gLite middleware, the common software stacks shared by both EGEE and WLCG projects. SweGrid Accounting System (SGAS) [7] is another grid accounting system designed for SweGrid, the national grid test-bed in Sweden, and used as the major accounting solution for the NorduGrid [8] infrastructure.

These grid accounting systems were implemented in various ways based on local understanding of project-specific requirements, making them hard to interoperate. Additionally, in multi-grid environments such as WLCG which involves three grid infrastructures from the OSG, EGEE, and NorduGrid projects, the accounting process is complicated due to the heterogeneity of accounting systems deployed at participating sites. Three transportation methods were therefore introduced as interim solutions to stream accounting data metered by grid-specific accounting systems into the WLCG Grid Operational Centre (GOC) at Rutherford Appleton Laboratory (RAL) in UK. These transportation methods, however, target particular accounting systems, making them hard to reuse and customize to meet evolving requirements. In this paper, we propose a standards-compatible solution, the WLCG-RUS system, which aims at providing the consistent collection of accounting data across various accounting systems in the WLCG project while ensuring interoperability, portability, and customization.

This paper is organized as follows. Section II reviews the current accounting processes in the WLCG project, and

Manuscript received June 15st, 2010. This work was supported in part by Engineering and Physical Sciences Research Council.

X. Chen is with the School of Electronics and Computer Science, University of Southampton, SO17 1BJ, Southampton, UK (e-mail: xc2@ecs.soton.ac.uk).

A. Khan is with the School of Engineering and Design, Brunel University, Uxbridge, UB8 3PH, London, UK (e-mail: akram.khan@brunel.ac.uk).

G. B. Wills is with the School of Electronics and Computer Science, University of Southampton, SO17 1BJ, Southampton, UK (e-mail: gbw@ecs.soton.ac.uk).

L. H. Gilbert is with the School of Electronics and Computer Science, University of Southampton, SO17 1BJ, Southampton, UK (e-mail: lg3@ecs.soton.ac.uk).

identifies problems to be addressed by the proposed WLCG-RUS system. The design and implementation details of the WLCG-RUS system are given in section III and section IV. Section V presents and discusses the unit and functional performance tests. Conclusions are given at the end of this paper.

II. ACCOUNTING IN WLCG

The accounting process in the WLCG project meters, collects, and represents resource usage of the EGEE/WLCG infrastructure as well as the collaborative grid infrastructures (i.e. OSG and NorduGrid) to provide an integrated view across grid boundaries. This accounting process is complicated by the heterogeneity of accounting tools that have various interface definitions and various formats of meter data. The process is further complicated by the restrictive security policies of collaborative grid projects. Some do not allow sharing of detailed resource usage information, instead only permitting anonymized summary usage information. Therefore two accounting models, the job accounting model and the aggregate accounting model, were introduced in the WLCG project. These models provide synchronization of resource usage information on a per batch job basis from EGEE/WLCG infrastructures, and anonymous summary usage statistics from collaborative grid projects. The information and statistics are transported to the WLCG GOC through three methods as illustrated in Fig. 1.

A. Job Accounting Model

In most EGEE/WLCG sites, APEL and DGAS are two widely deployed accounting tools. The APEL accounting tool consists of a number of log processors that meter usage information from log files of gatekeeper and batch systems, and query other relevant information from sites' information services. This information is then merged as complete usage records on a per batch-job basis, and stored in a relational database at each site. The APEL accounting tool also provides a publisher component that automates the collection process and publishes job usage records into WLCG GOC through the Relational Grid Monitoring Architecture (RGMA) [9] protocol. DGAS generates job accounting records in a different format. In order to share DGAS accounting records to WLCG GOC, a lightweight component, DGAS2APEL, was built to transform DGAS accounting records into the APEL usage record format, reusing the APEL publisher module to publish job usage records into GOC through the R-GMA protocol.

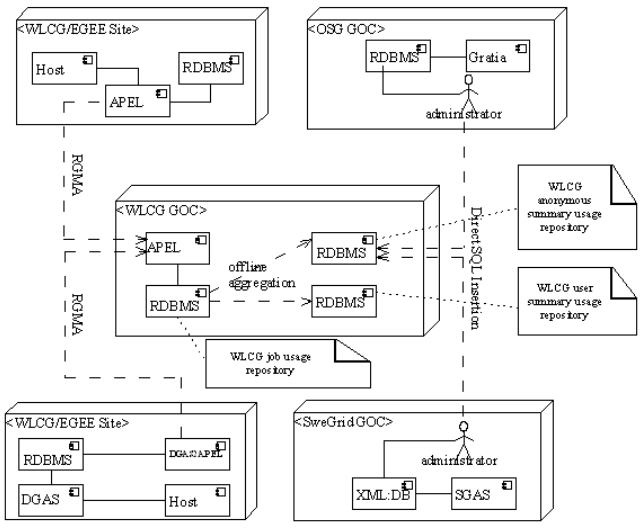


Fig. 1. The current WLCG accounting infrastructure

B. Aggregate Accounting Model

After job usage records are received from sites, an off-line daily scheduled aggregation process at WLCG GOC summarizes usage statistics, which are accessible to communities via a Web portal. This aggregate accounting process acts upon the central database of job usage records and generates two types of summaries: the user summary usage record and the anonymous summary usage record. Generated usage statistics can be used to provide various views of usage statistics for Virtual Organization (VO) managers, VO members, end users, and site administrators. The two summary usage representations along with the APEL job usage representation are collectively defined as the standard WLCG accounting schema [10].

For those sites from collaborative grid projects with restrictive security policies, the WLCG accounting framework introduced a third transportation method, the "direct SQL insertion", allowing grid systems administrators to populate either user summary usage records or anonymous summary usage records by directly executing SQL insertion statements on central databases at WLCG GOC. Unlike automated job accounting process, this method requires human intervention and additional administrative effort.

.

C. Enforcement Activities

During the WLCG job accounting process and the following aggregation process, there is a sequence of enforcement activities which ensure data integrity.

There are over 200 sites across different time zones participating in the WLCG project. To ensure time consistency, date-time properties of every job usage record are required to be published in the ISO8601 format (e.g. 2008-10-01T21:39:28+01:00). A process can then transform these date-time properties into Coordinated Universal Time (UTC) values (e.g. 2008-10-01T20:39:28Z).

In order to normalise the CPU usage data from many disparate sites the enforcement procedure requires every published job accounting record to have a SpecInt [11] value that is taken from the sites' information system. Sites are required to publish a meaningful (non-zero) SpecInt value.

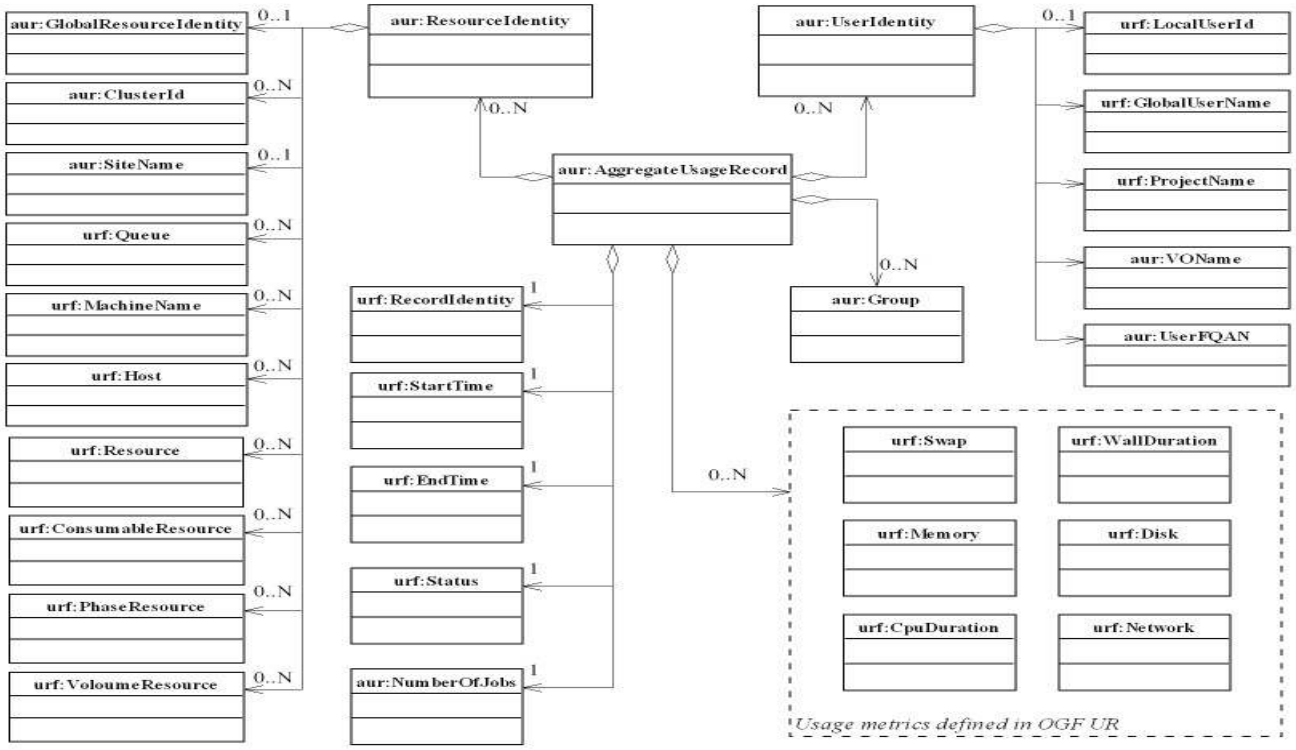


Fig. 2. Content model of proposed AUR standard

When aggregating job usage records into user summary repository, user information, such as user role and group, is extracted from Virtual Organization Membership Service (VOMS) [12].

III. Design of WLCG-RUS System

In this paper, we propose an alternative standards-compatible solution, enabling both job and aggregate accounting models across various accounting systems deployed in the WLCG participating sites in a consistent manner.

A. Design Objectives

The main design goal of the WLCG-RUS system is to provide a standards-based method that automates both WLCG job and aggregate accounting models. In order to achieve this goal, there is a set of objectives to meet.

Standardization

The WLCG-RUS system adopts two standards proposed by the Open Grid Forum (OGF), the Usage Record (UR) Representation [13] standard proposed by OGF UR working group, and the Resource Usage Service (RUS) [14] standard proposed by OGF RUS working group. The OGF UR standard defines a set of core properties for representing job usage records in XML format, while the OGF RUS standard defines a set of core service interface definitions mainly for publishing and querying OGF UR compatible instances based on Web Service Interoperability (WS-I) [15] profile. These two standards focus on job accounting, however, and do not support the WLCG aggregate accounting model.

Back compatibility

The deployment of WLCG-RUS system should not break current WLCG accounting processes, but provide an alternative method for data transportation. This means that the WLCG-RUS system should use existing accounting data repositories at WLCG GOC.

Customization

Considering the evolving nature of WLCG accounting framework, the design of WLCG-RUS system should be flexible enough to adopt possible updates (e.g. changes to WLCG accounting schemas or introduction of new schemas).

B. Design of Aggregate Usage Record

As discussed before, the OGF UR standard focuses on the representation of job usage records. In 2006, we collaborated with researchers from Fermilab and RAL, and proposed an Aggregate Usage Record (AUR) standard [16]. An AUR instance represents summary usage statistics of more than one Unit of Work (UoW), ranging from finest-grained batch jobs to complex service workflows. The collection process at coarse-grained level involves an extra aggregation process, according to a specific grouping criterion, also known as an aggregation strategy.

As illustrated in Fig. 2, the content model of AUR reuses usage properties of UR and defines a set of common aggregate properties, including total number of UoWs aggregated, aggregation interval starting from the start time of earliest

UoW to the end time of the last UoW, and overall status of UoWs aggregated. User properties define the ownership of UoWs within an aggregate usage record instance. In addition to those user properties defined within OGF UR, AUR introduced additional VO-related properties (e.g. VO name, user's role in the VO, and user's subgroup in the VO). Additional resource-related properties are also introduced in the AUR schema to describe the properties of grid-wide resources upon which UoWs were executed. These properties include global resource identity, cluster identity, participating site name, etc.

These non-usage properties can be combined to represent a grouping criterion or aggregation strategy of an aggregate usage record instance. A WLCG summary usage record instance, for example, defines an aggregation strategy that summarizes resource usage of batch jobs on a per VO, per site, per month and per year basis, and can be formatted into an AUR instance as follows:

```
<aur:AggregateUsageRecord ...>
<aur:RecordIdentity ...>
<!--aggregate properties-->
<urf:StartTime>2007-01-01T00:00:00Z<urf:StartTime/>
<urf:EndTime>2007-12-31T23:59:59Z</urf:EndTime>
<urf:Status>completed</urf:Status>
<aur:UserIdentity>
<aur:VOName>Atlas</aur:VOName>
</aur:UserIdentity>
<aur:ResourceIdentity>
<aur:SiteName>UKI-LT2-Brunel</aur:SiteName>
</aur:ResourceIdentity>
...
<aur:AggregateUsageRecord />
```

The AUR schema also defines an extension property, the "*<aur:Group>*", which can be used for the definition of custom aggregate properties that are not defined within the AUR representation. It is worth noting that the use of "group" extensions might undermine the interoperability.

C. Extensions to OGF RUS

The service interfaces defined within the OGF RUS specification are closely coupled to the OGF UR standard. This means a RUS service endpoint can only accept OGF UR instances. In order to publish AUR instances through the standard RUS insertion interface, a "*<xsd:any>*" extension was added to the RUS insertion request message definition. This means a RUS service endpoint can be potentially used for any usage record instances, including AUR instances.

D. Design of System Architecture

As illustrated in Fig. 3, the WLCG-RUS system architecture consists of two subsystems, the RUS service and WLCG-RUS Admin.

RUS service

The RUS service is the core of the WLCG-RUS system and exhibits two standard service interfaces as defined in the OGF RUS specification. The "*RUS::listMandatoryUsageRecord-Elements*" interface is used by a client to query the mandatory elements that must appear in a usage record instance. The "*RUS::insertUsageRecords*" is the interface through which job or aggregate usage record instances can be published to WLCG GOC.

The design of the RUS service is based on a component architecture, consisting of a set of loose-coupled and reusable components. Each component targets a certain functionality and exhibits well-defined interfaces. These components are designed in a loosely-coupled pattern, so that they can be easily customized, upgraded, and replaced to adapt to local deployment requirements.

As the internal design illustrates in Fig. 4, there are four abstract functional components defined within the RUS service. The "Command" component is the main functional component for the execution of RUS logic operations. A single common interface, the "*execute()*", completely decouples the RUS service endpoint from various "Command" component implementations. On receiving a request, a RUS service endpoint delegates the incoming request to different

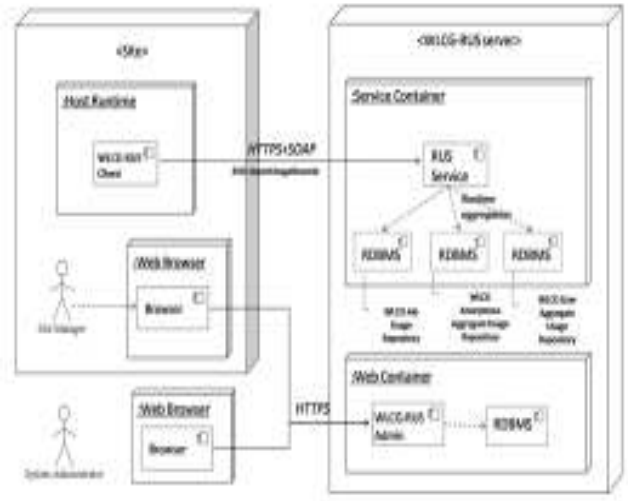


Fig. 3. The WLCG-RUS system components and interactions

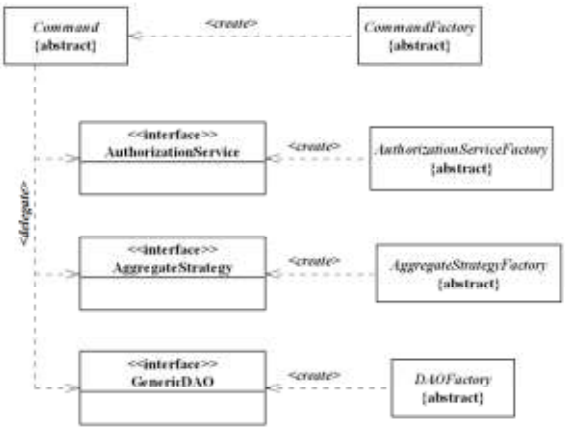


Fig. 4. The internal design of the RUS service

"Command" implementations. A RUS service may chose to implement a single "Command" implementation that serves all RUS requests or to have multiple "Command" implementations for each RUS service interface. The execution of various "Command" implementations shares a common workflow, including checking user permissions, applying an appropriate aggregate strategy, and ensuring data persistence. This common workflow is realized through the three other RUS service components.

The authorization service component provides an interface for fine-grained access control over operation and usage records, allowing the application of different authorization mechanisms. The Data Access Object (DAO) component provides a higher-level abstraction upon the underlying data storage, and can be implemented for data persistence in XML databases, relational databases, file systems, and other storage formats. Different aggregation strategies can be implemented by extending the aggregate strategy interface. Each component of the RUS service has an associated factory interface that creates and instantiates component instances dynamically.

WLCG-RUS Admin

The WLCG-RUS Admin is designed as a Web application based on the Model-View-Controller (MVC) pattern, with models encapsulating domain-specific representations of data, controllers representing domain-specific logics operating upon the data, and views providing Web-based interfaces allowing end-user interactions. The WLCG-RUS Admin Web application is intended to provide administrative and host management facilities for the WLCG-RUS system.

In order to access the WLCG-RUS Admin system, a user must have a valid and recognized X.509 user certificate, and a valid user account. Each user is directed to a specific view according to their granted role. Site managers only have access to host management facilities, which allow host registration, exploring host status, and deleting a host. Newly registered hosts cannot share accounting data or usage records through a RUS service endpoint until their registration request is approved by the system administrator. A site manager only has management authority over owned hosts. A system administrator has an administrative view, which provides facilities for user and host management. A system administrator can create a new role, grant a role to registered users, revoke a user, publish system announcements, and have full control over all hosts registered by site managers.

Another important usage of WLCG-RUS Admin is to specify RUS service configurations, including the creator of RUS service functional components, maximum usage records per insertion, and mandatory elements for validating incoming usage records.

IV. Implementation

The following gives implementation details of the WLCG-RUS system.

A. RUS service

The implementation of RUS service reuses the three WLCG accounting data models defined by the WLCG accounting framework to ensure backward compatibility. In order to upload accounting data through the standard RUS insertion interface, a data mapping mechanism is triggered at runtime to transform the XML-formatted usage records into WLCG relational data representations. The RUS service accepts OGF UR or AUR instances, and uses an Object-Relational Mapping (ORM) mechanism to save them into WLCG accounting storage. Hibernate [17] is employed as the ORM engine. These class models also implement the enforcement activities described in section III.C to ensure data consistency.

As illustrated in Fig. 5, three command implementations are provided in the default RUS service, serving as the main components of the WLCG job and aggregate accounting processes. A lightweight authorization service is provided to perform fine-grained access control based on user-role mapping information maintained by the WLCG-RUS Admin system. Two aggregation strategies are implemented to enable runtime aggregations for WLCG anonymous and user aggregation strategies. Each object includes an associated DAO implementation, which provides data persistence through the Hibernate ORM engine.

An example runtime aggregation model is given in Fig. 6 and involves a sequence of interactions as follows:

1) Host client sends a "*RUS::insertUsageRecords*" SOAP request message to a RUS service endpoint.
2) On receiving an insertion request, the RUS service endpoint instantiates command, authorization service, DAO, and aggregate strategy components through configured factory classes, and loads mandatory element configurations into runtime.
3) The RUS service endpoint delegates the insertion request to the command component through *execute( )* interface.
4) For each usage record instance, the command component firstly checks for user authority to perform an insertion.
5) Once authorized the command component then validates the current usage record against the mandatory element configurations.
6) If the received usage records are OGF UR instances, an aggregate strategy is triggered. This generates one or more instances of the target aggregate class,instances of WLCG anonymous aggregate records in this example. Otherwise, the command component creates an instance of the target aggregate class by passing the current OGF AUR instance to the "*LcgSumRecord*" constructor.
7) The command component then invokes the save method of "*LcgSumRecordDAO*" and passes the "*LcgSumRecord*" instance.
8) The DAO object makes the "*LcgSumRecord*" instance persistent into a local relational database and returns a record identity.

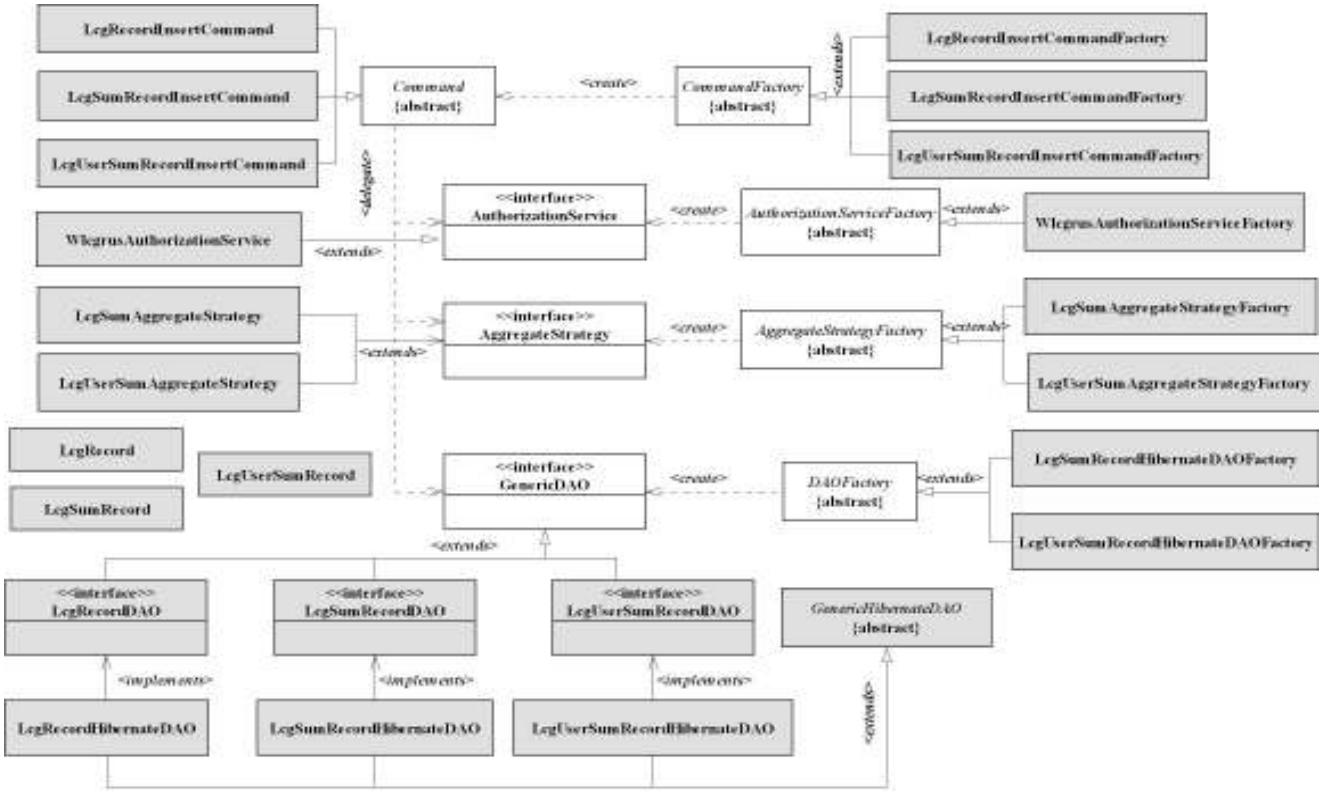


Fig. 5. Implementations of internal components of the RUS service

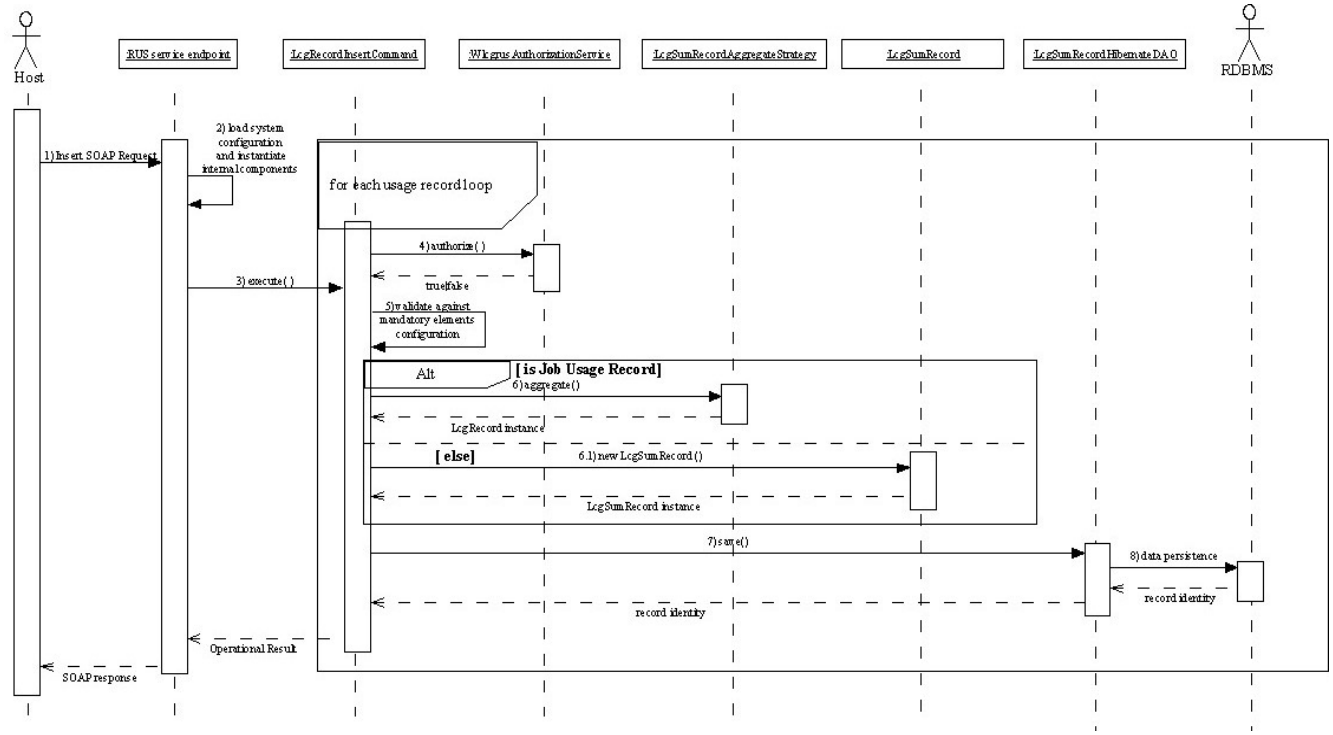
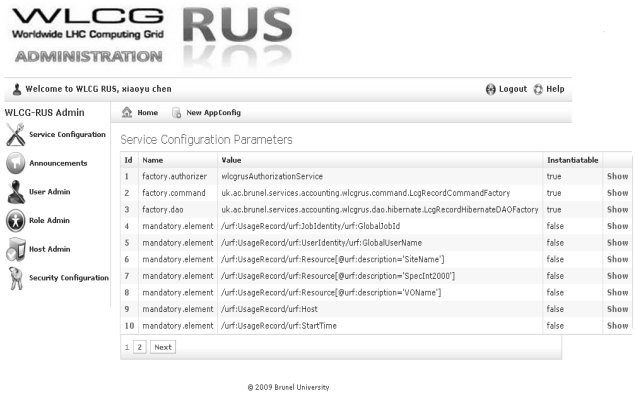
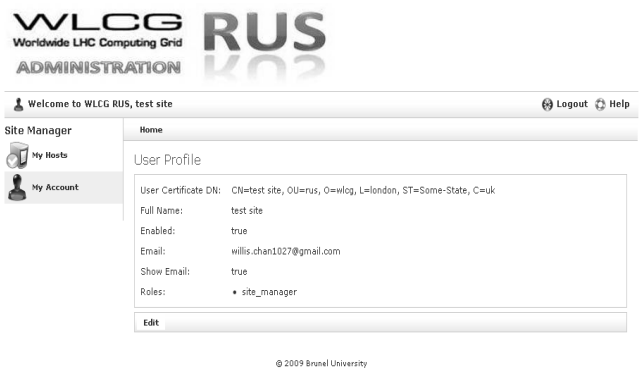


Fig. 6. Example workflow of aggregate accounting implemented in the WLCG-RUS system.



(a)



(b)

Fig. 7. (a) The administration view (b) The management view .

B. *WLCG-RUS Admin Web Application*

The WLCG-RUS Admin Web application is implemented based on a Grails [18] framework and uses the Groovy [19] script language, a perfect combination for agile development with well-maintained and featured plug-ins for Web 2.0 and Web service applications. Based upon the Grails framework, the WLCG-RUS service backend and the WLCG-RUS admin web application can be integrated and delivered as a single package.The implementation adopts the passive MVC model with one controller exclusively manipulating one model and refreshing changes of the model to views.

WLCG-RUS Admin provides Web interfaces for site mangers and system administrators through two views, the manager view (Fig. 7a) and administrative view (Fig. 7b). A WLCG-RUS system administrator can configure the runtime of a RUS service endpoint by specifying factory classes of individual functional components, editing the mandatory element list, and managing RUS client authorities. A registered site manager adds, edits, and deletes hosts that share accounting data.

C. *Client Interface*

The WLCG-RUS system provides a Command-Line interface (CLI), the WLCG-RUS client, allowing access to the RUS service endpoint through standard RUS service interfaces. The WLCG-RUS client is implemented using the Java programming language, and is wrapped by a shell script.

Before using the client, users must configure their environment to provide all required information to establish mutual authentication. A configuration file allows users to specify the location of a site, trusted Certificate Authority (CA) certificates, and access passphrases.

The client accepts a set of arguments. A least one of the two actions, “list” and “insert”, must be used every time the client is triggered. A mandatory argument, the “service_uri” is used for both actions to specify the URI of the target RUS service endpoint. The “insert” action can be combined with additional arguments providing more controls over the action. As in the following example, the “insert” action is combined with three additional parameters to publish all usage record instances stored in a local directory with 10 usage records per transaction, and delete successfully inserted usage records.

```
>wlcgrus --service-uri http://localhost:8080/wlcgrus
         --insert --dir /opt/usages
         --max-elements 10
         --delete-after-insertion
```

If errors are encountered during execution, the target file name is changed and appended with an “ERROR” suffix, and server-side error messages are also appended. This feature ensures reliability of data delivery. At the server side, each insertion is dealt as an atomic transaction; therefore failures at any step (i.e. validation of mandatory usage record elements, authorization, aggregation, etc) during insertion would result in the overall failure of the overall usage record. However, the failure of a single usage record should not affect insertion of other usage records within the same transaction. System administrators can then check local usage record directory to examine the failure status. Automatic retry mechanisms can be also implemented using the WLCG-RUS client interfaces. The client can also be used by host machines to upload usage records to a RUS service endpoint automatically by scheduling the shell client as a “cron” job and publishing usage records periodically.

V. PERFORMANCE

This section provides performance evaluation of the WLCG-RUS system. The test results are intended to provide reference guidance for deployment of WLCG-RUS system to obtain optimal performance.

A. *Testbed*

In order to better demonstrate the performance of WLCG-RUS system, a testbed was set up in the Brunel Information Technology Laboratory (BITLab) at Brunel University, one of

the UK tier-2 sites, to simulate the accounting process in the production WLCG environment. The testbed consists of two workstations that are interconnected by Local Area Network (LAN). One dedicated workstation is used to host WLCG-RUS server, which keeps listening insertion requests from clients. The hardware and runtime environment details of the WLCG-RUS server are listed in TABLE I. On the other workstation, a number of clients along with a usage record generator were deployed to simulate the accounting process at Grid participating sites. The usage record generator simulated the metering process and generates standard OGF UR or AUR instances into the local file system. One or more WLCG-RUS clients were then scheduled to read usage record instances from that directory and populate them to the WLCG-RUS server simultaneously through the standard RUS::InsertUsageRecords interface. A thread pool was also provided to hold multiple WLCG-RUS client threads and ensured a fixed number of threads that interrogate the WLCG-RUS server at a time.

TABLE I. TEST SERVER HARDWARE AND RUNTIME SPECIFICATION

Component	Description
Processor	Genuine Intel (R) Duo Core (1.66 GHz)
Memory	1024 MB
Operating System	Ubuntu 32-bit
Web Container	Apache Tomcat 5.5.23
Service Container	Apache Axis 1.4
DBMS	MySQL 5.1

Based on the testbed, a series of tests were conducted to:

- Evaluate the performance of individual WLCG-RUS runtime components (as discussed in section III.D). The result of which is to be used by deployers to have a detailed picture on how WLCG-RUS system perform, and by developer to improve system performance through custom implementation of particular runtime components.
- Evaluate how the WLCG-RUS system's insertion performance varies with different deployment options, in particular the number of usage records per insertion transaction, known as bulk size, and the number of client threads. The result of the insertion performance test is expected to be used by deployers to make decisions on how to deploy WLCG-RUS system to obtain optimal performance.

B. Unit Performance

Fig. 8 plots the performance of runtime component units of different accounting models, both job accounting and aggregation accounting models. Multithreading was intentionally avoided in these tests so that overall costs of individual runtime components in different accounting models, both job and aggregate accounting models, can be fairly observed and compared.

As summarized in Table II, the average performance of authorization, messaging and validation processes are similar

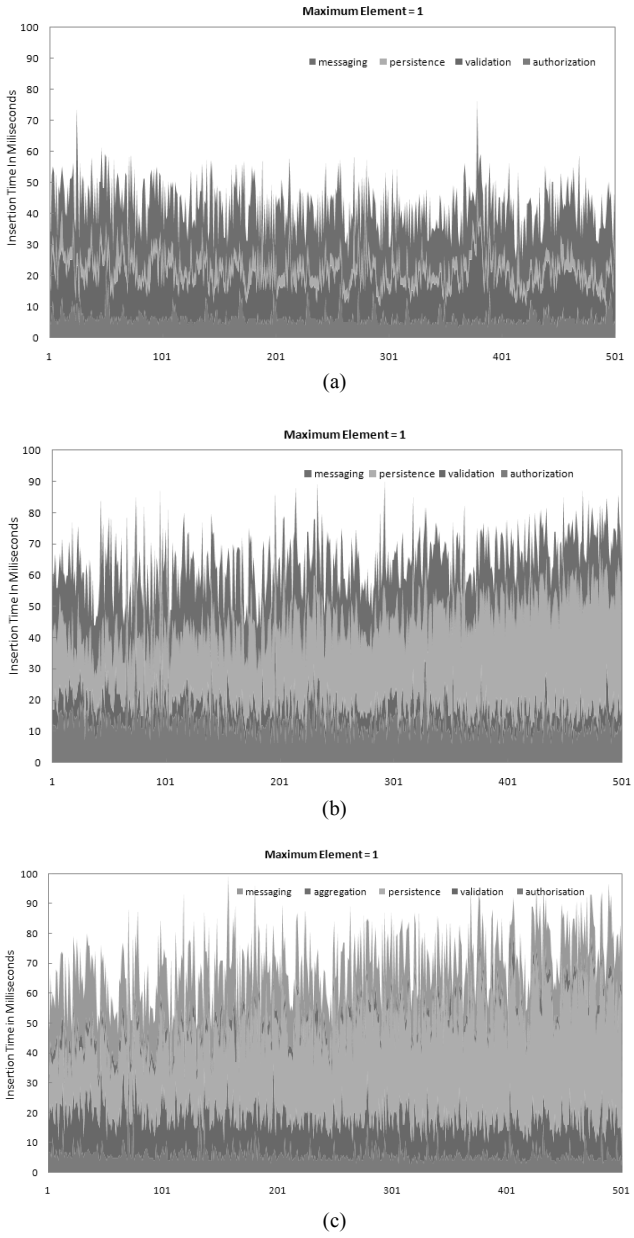


Fig. 8. (a) Unit runtime costs of job accounting process. (b) Unit runtime costs of aggregate accounting process without runtime aggregation. (c) Unit runtime costs with runtime aggregation.

TABLE II
COMPARISON OF AVERAGE RUNTIME COSTS OF JOB AND AGGREGATE ACCOUNTING PROCESSES

Model	Average Costs (Time in Milliseconds)					
	Messaging	Authorization	Validation	Aggregation	Persistence	Total
Job	19.924	6.334	13.330		3.858	42.446
Aggregate accounting (no runtime aggregation)	18.146	6.266	12.542		28.880	65.834
Aggregate accounting (runtime aggregation)	18.492	6.194	13.298	3.480	30.868	72/332

with slight difference less than 0.008 second. Comparing to job accounting model, aggregate accounting models exhibits worse performance mainly because of additional complexity introduced on the data persistence process. On receiving an insertion request of an aggregate usage record, the WLCG-RUS system runtime requires check whether there is an existing aggregate usage record using same aggregate strategy. In the case of WLCG anonymous aggregate strategy for example, the WLCG-RUS runtime is required to the existence of an aggregate usage record with certain month/year, certain VO and certain executing site. If an existing record found, the WLCG-RUS runtime is then add usage information to the existing record, and change the aggregation starting and ending time accordingly. Therefore the data persistence process introduces average 0.02 second overhead. In the aggregate accounting model with runtime aggregation, additional 0.003-second overhead is introduced by the enforcement of the WLCG anonymous aggregation strategy. However this figure can be quite different depending on the complexity of an aggregation strategy implementation.

C. *Insertion Performance*

The WLCG-RUS system runtime can be configured to accept one or more usage records per insertion transaction. The number of usage records per transaction is also called bulk size. The first part of the insertion performance test is to evaluate the WLCG-RUS system performance with different bulk size. In this test, the client machine continuously inserts 35,000 job usage records to the WLCG-RUS server. Successive execution time is logged when finishing insertion of 5,000, 10,000, 15,000, 20,000 25,000, 30,000 and 35,000 usage records. As the performance plot described in Fig. 9, the insertion time decreases gradually with the increasing bulk size until the bulk size is 10, and then increases exponentially. Based on the test results, the maximum elements should be set between 10 and 15 in order to gain optimal insertion performance, as illustrated in Fig. 10.

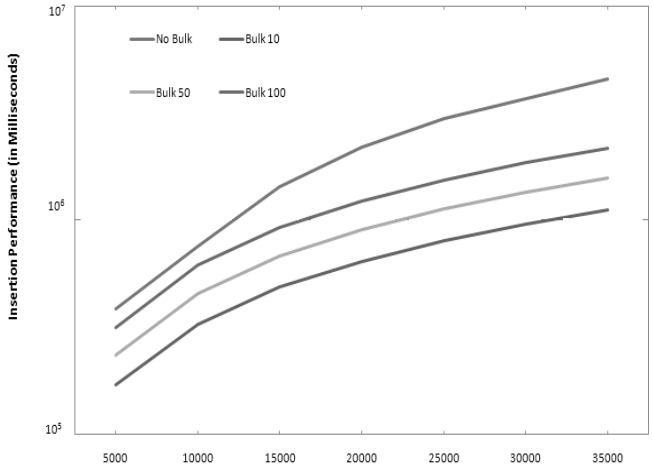
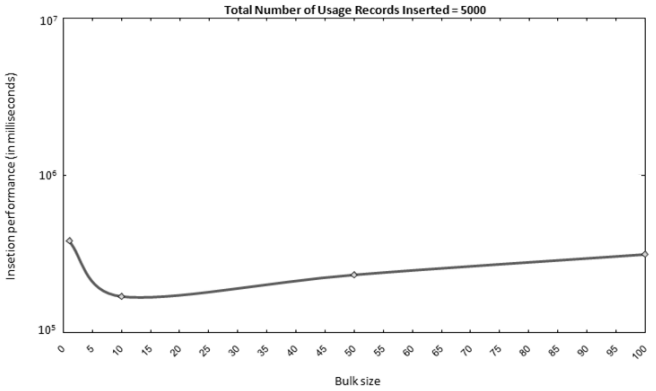
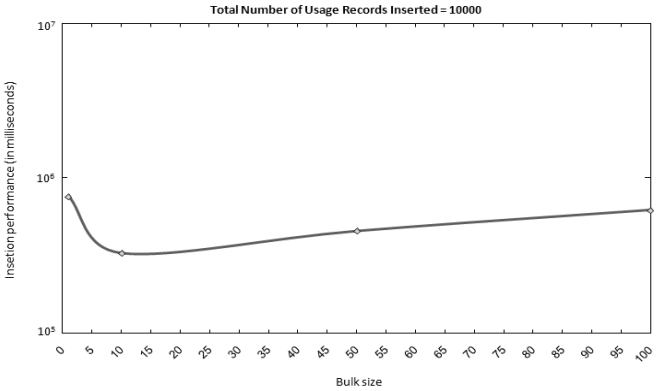


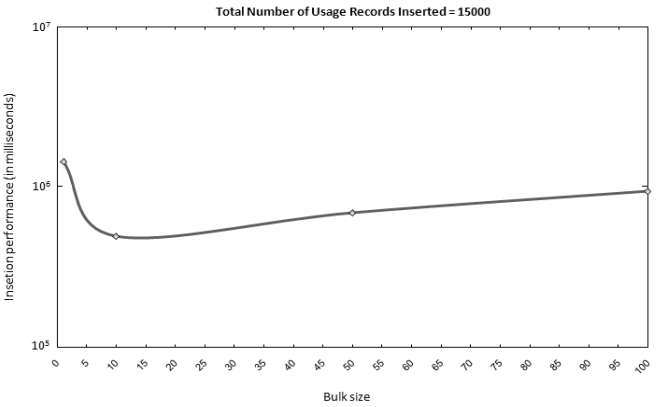
Fig. 9. Insertion performance against different granularities of usage records per transaction.



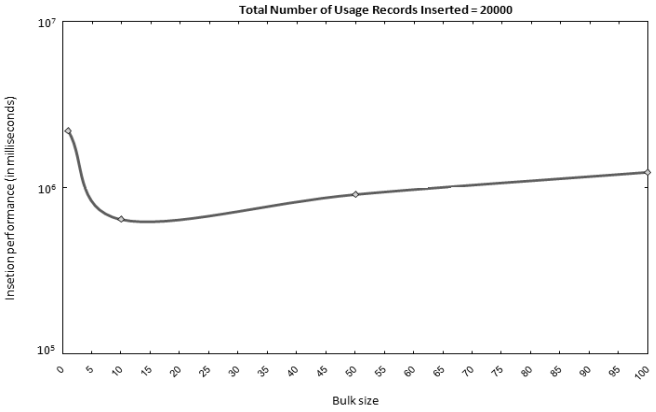
(a)



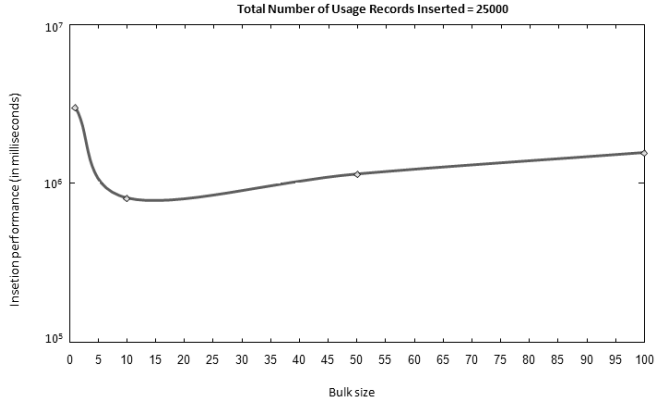
(b)



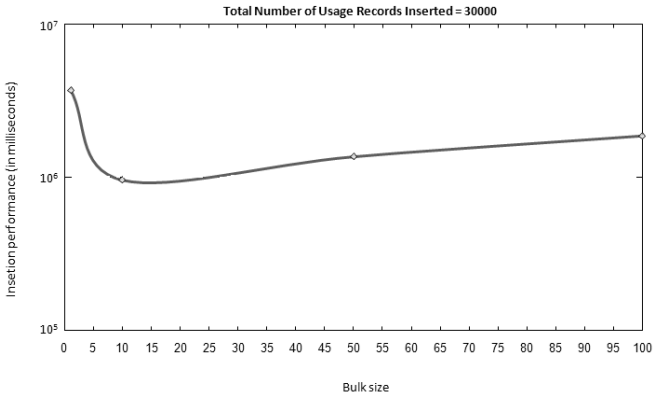
(c)



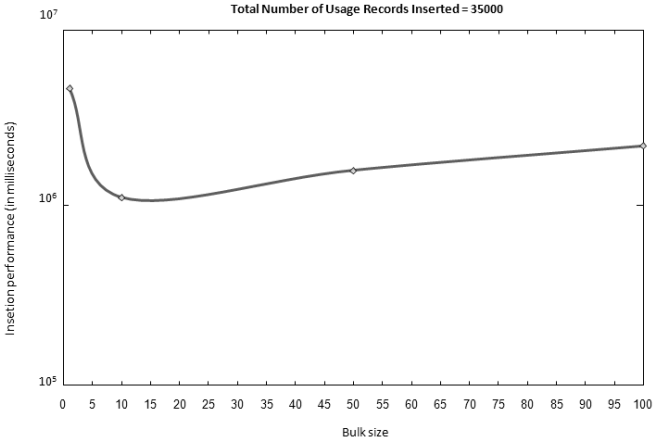
(d)



(e)



(f)



(g)

Fig. 10 (a) insertion performance of 5,000 usage records against bulk size (b) insertion performance of 10,000 against bulk size (c) insertion performance of 15,000 usage records against bulk size (d) insertion performance of 20,000 against bulk size (e) insertion performance of 25,000 usage records against bulk size (f) insertion performance of 30,000 against bulk size (g) insertion performance of 35,000 usage records against bulk size.

The WLCG-RUS system can be deployed in two ways in the context of the WLCG accounting process. It can be either deployed at the GOC centre as a singleton entry point or hierarchically deployed at each regional site responsible for region-wide accounting purposes while streaming accounting data to the main WLCG-RUS server at GOC. For both cases, the WLCG-RUS system is required to serve multiple client

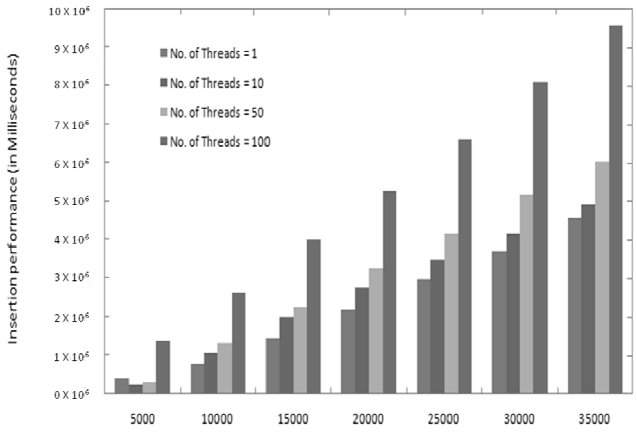


Fig. 11. Insertion performance against the number of simultaneous client threads

requests at a time. In order to figure out the performance of WLCG-RUS system when dealing with multiple client requests simultaneously, and find out which way is of best performance for the WLCG accounting process, a multi-threading test is conducted to evaluate WLCG-RUS system performance against different number of client threads. As the performance plot illustrated in Fig. 11, the WLCG-RUS system performance decreases with the increasing number of client threads. In the case of 100 client threads insert usage records at same time, the total time cost for insertion of 35,000 usage records reaches 2.6 hours (0.27 second per transaction), comparing to 1.26 hours (0.13 second per transaction) when using a single client thread. In the case of WLCG accounting, it is better to adopt the hierarchical deployment manner, with multiple WLCG-RUS server deployed at regional sites and one central WLCG-RUS server deployed at GOC site to accept requests from regional sites only. It is worth noting that the performance of WLCG-RUS system may gain better performance when deployed on modem server machines with multi-core or multi-CPU supports.

VI. Conclusion

Accounting in a multi-grid environment such as the WLCG project involves the collection of usage records generated by heterogeneous accounting systems. These usage records are represented in various formats. The accounting process in the WLCG project is further complicated by project-specific security policies. Two accounting models were introduced in the WLCG project for sharing both job and aggregate usage records from participating sites to WLCG GOC through three data transportation methods. These transportation methods were defined on a per-accounting system basis, and require additional administrative effort.

In order to provide a consistent solution for automating the collection of usage records across various grid accounting systems, while accommodating local security policies, this paper proposes the WLCG-RUS system to provide an alternative but standards-based way to automate WLCG job

accounting and aggregate accounting processes. The design of the WLCG-RUS system consists of two subsystems, based on a loosely-coupled component-based architecture to provide default implementations compatible with OGF UR and OGF RUS specifications. The work described in this paper also contributes to a proposed standard aggregate usage record representation. The performance tests illustrated the effectiveness of the WLCG-RUS system and provide guidance notes for system deployers who are interested in employing the WLCG-RUS system as a part of their accounting solutions on how to deploy WLCG-RUS system to obtain optimal performance.

REFERENCES

[1] Open Science Grid. http://www.opensciencegrid.org/

[2] P. Canal, S. Borra, M. Melani, "GRATIA, a resource accounting system for OSG", *Proc. of Computing in High Energy and Nuclear Science 2006 (CHEP06)*, Mumbai, Inida, February 2006.

[3] R. Byrom, R. Cordenonsi, and L. Cornwall, "APEL: An implementation of Grid accounting using R-GMA", *UK e-Science All Hands Conference,* Nottingham, September 2005.

[4] R. M. Piro, M. Pace, A. Ghiselli, A. Guarise, E. Luppi, G. Patania, L. Tomassetti, and A. Werbrouck, "Tracing Resource Usage over Heterogeneous Grid Platforms: A Prototype RUS interface for DGAS", *Proceedings of International Conf. on e-Science and Grid Computing*, Dec. 2007, pp93-101.

[5] Wordwide LHC Grid. http://lcg.cern.ch/lcg

[6] Enabling Grids for E-sciencE. http://www.eu-egee.org/

[7] P. Gardfjall, E. Elmroth, L. Johnsson, and O. Mulmo, "Scalable Gridwide capacity allocation with SweGrid Accounting System" *Concurrency and Computation: Practice and Experience*, John Wiley and Sons Ltd, June 2008.

[8] Nordic Data Grid Facility, http://www.ndgf.org/ndgfweb/home.html

[9] B. Coghlan, A. W. Cooke, A. Datta, et. al., "R-GMA: A Grid Information and Monitoring System" *Conf. on UK e-Science all hands*, Sheffield, 2-4 September 2002.

[10] R. Byrom and D. Kant, "LCG Accounting Schema", *EGEE Support and Management Activity (SA1) document*, Oct. 19th, 2004. Available online at: http://www.egee.cesga.es/EGEE-SA1-SWE/accounting/guides/apel-schema.pdf.

[11] Standard Performance Evaluation Corporation (SPEC) "Computer benchmark specification for CPU's integer processing power,", Oct. 2001.Avaliable at http://www.spec.org/cpu2000/.

[12] R. Alfieri, R. Cecchini, V. Ciaschini, et. al, "VOMS: an Authorisation System for Virtual Organisations", *Procs. of Computing in High Energy Physics (CHEP)*, India, 2004.

[13] L. McGinnis, R. Mach, R. Lepro-Mez, and S. Jackson, "Usage Record-Format Recommendation version 1.0", *Open Grid Forum Usage Record Working Group,*GFD.58, September 2006. Available online at: https://forge.gridfourm.org/projects/ur-wg/

[14] J. Ainsworth, S. Newhouse, and J. MacLaren, "Resource Usage Service Based on WS-I Basic Profile 1.0 (draft)", *Open Grid Forum Resource Usage Service Working Group,* August, 2006.

[15] K. Ballinger, D. Ehnebuske, et. al., "Web Services Interoperability Basic Profile Verion 1.0", Web Services Interoperability Organization. Available at http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html

[16] X. Chen, R. M. Piro, P. Canal, et. al, "Aggregate Usage Representation Version 1.0", *OGF Usage Record working group*, Dec. 2006. Available online at: https://forge.gridfourm.org/projects/ur-wg/

[17] Hibernate, https://www.hibernate.org/

[18] Groovy, http://groovy.codehaus.org/

[19] Grails, http://www.grails.org/