A Self-Organising Federation of Alchemi Desktop Grids

Rajiv Ranjan, Xingchen Chu, Carlos A. Queiroz, Aaron Harwood, Rajkumar Buyya*
GRIDS Lab and P2P Group
Department of Computer Science and Software Engineering
The University of Melbourne, Victoria, 3010, Australia

August 13, 2007

Abstract

Desktop grids presents a next generation platform for aggregating the idle processing cycles of desktop computers. However, in order to efficiently harness the power of millions of desktop computers, the systems or middlewares that can support high level of efficiency, scalability, robustness and manageability are required.

In this paper, we propose a scalable and self-organising desktop grid system, Alchemi-Federation, which uses a Peer-to-Peer (P2P) network model for discovering and coordinating the provisioning of distributed Alchemi grids. Alchemi grids self-organise based on a structured P2P routing overlay that maintains a d-dimensional index for resource discovery. The unique features of Alchemi-Federation are: (i) Internet-based federation of distributed Alchemi grids; (ii) implementation of a P2P publish/subscribe based resource indexing service that makes the system highly scalable; and (iii) implementation of a P2P tuple space-based distributed load-balancing algorithm.

1 Introduction

Grids have emerged as the next generation platform for sharing the topologically and administratively distributed resources, services and scientific instruments. Recently, desktop grids are progressively seen as an alternative or a compatriot to the traditional dedicated grids. Traditionally, Virtual Organisation (VO) based dedicated Grid resource sharing environment consists of few tens or hundreds of resource sharing domains. In comparison to this, desktop grids such as SETI@Home[1], Folding@Home[2] have demonstrated that millions of resources can be aggregated together for extracting large computational power (accounting to hundreds of Tera-Flops). Some of the example applications that have been successfully executed on desktop grid platform includes the Monte-Carlo computation, virtual screening, protein sequence comparison, climate prediction, gravitational wave detection and cosmic rays study. The Internet-wide deployment and adaption of these desktop grids systems clearly proves their utility with respect to Grid resource sharing model.

Current implementation of desktop grid resource sharing system such as SETI@Home, Folding@Home, BOINC[1], Entropia[2] utilises centralised architecture for resource discovery and application scheduling. In this case, a central machine is responsible for managing the system wide operation. If the central machine or the network links leading to it fails then the whole desktop grid system fails to operate hence leading to degraded resource and scheduling performance. Further, they have been designed to solve specific scientific problems (monolithic design) i.e. these systems do not provide support for different kinds of application models. Finally, these desktop grid computing system provide minimum support for efficient application load-balancing across the machines. In other words, the scheduling methodology adopted by these systems are trivial and employ random node selection strategy without taking into account the current load or utilisation scenario. Considering the sheer scale and dynamicity of desktop grid resources, existing systems needs to be augmented with new generation resource discovery and application scheduling techniques.

To overcome the above limitations in the current desktop grid systems we propose Alchemi-Federation system. The Alchemi-Federation system logically connects topologically and administratively distributed Alchemi grids as part of a single cooperative system [9]. The unique features of Alchemi-Federation includes: (i) it is self-organising and scalable, (ii) implementation of Distributed Hash Table (DHT) such as Pastry [10] or

*Contact author: raj@csse.unimelb.edu.au
[1]'SETI@Home'. http://setiathome.ssl.berkeley.edu/
[2]'Folding@Home'.http://folding.stanford.edu/

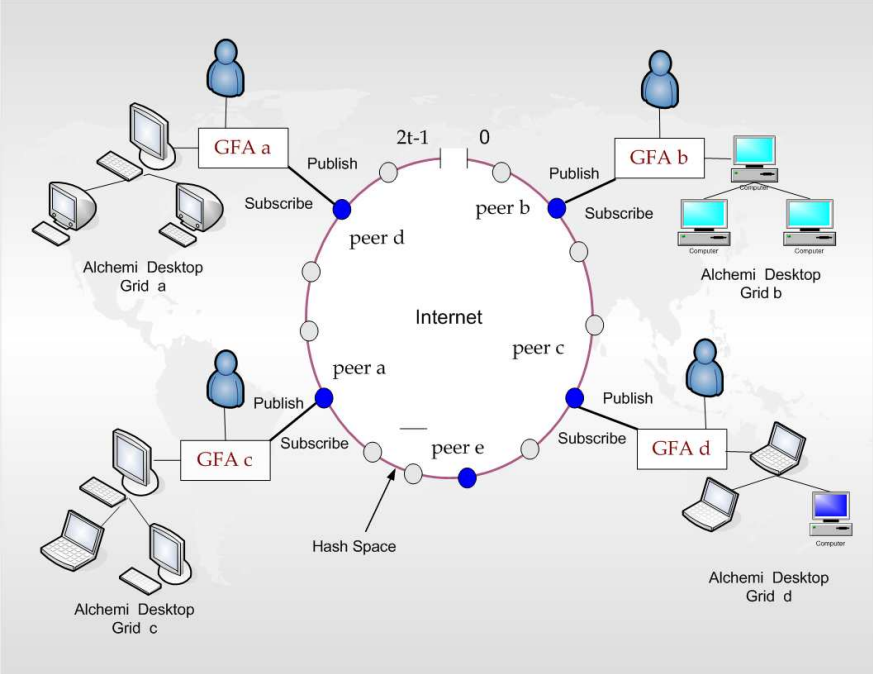


Figure 1: Alchemi GFA and sites with their Grid peer service and some of the hashings to the Chord ring.

Chord [11] based d-dimensional indexing for discovery and monitoring of desktop grid resources, and (iii) implementation of a novel resource provisioning technique the allocates application to the best possible resource sets, based on their current utilisation and availability in the system.

Alchemi-Federation realises the theoretical Grid resource sharing model called Grid-Federation [9]. Grid-Federation system is defined as a large scale resource sharing system that consists of a coordinated federation of distributed Alchemi grids. Fig. 1 shows an abstract model of our Alchemi-Federation over a P2P publish/subscribe resource discovery service. To enable transparent resource sharing between these Alchemi grids, a new resource management system called Grid Federation Agent (GFA) service is proposed. These GFAs coordinate resource discovery and job scheduling activity using P2P based publish/subscribe resource discovery service.

We have also tested the performance of the proposed software system in a resource sharing network that consisted of federation of 5 Alchemi desktop grids distributed over three departmental laboratories. The test application was a windows executable (source code written using c-sharp) that computed whether a given number is prime or not. In order to introduce processing delays, the process was made to sleep for 10 seconds before it could proceed to check the prime condition.

The rest of this paper is organised as follows: we start with brief description of background information on the Alchemi desktop Grid computing system in Section 2. Section 3 presents the overall software architecture of Alchemi-Federation system; including details on the individual components of Alchemi-Federation software. Section 4 discusses the implementation of P2P publish/subscribe based resource discovery service and software interfaces. Section 5 presents brief details on coordinated resource provisioning technique that performs decentralised load-balancing across Alchemi grids. In Section 6 we present details on service deployment and bootstrap. Section 7 includes the discussion on the performance evaluation. Finally, paper ends with a discussion on conclusion and future work.

2 Alchemi: A Brief Introduction

Alchemi [6] is .NET based enterprise Grid computing and runtime machinery for creating a high-throughput resource sharing environment. An Alchemi Manager logically couples the Windows Desktop machines running the instance of Alchemi Executor service. An Executor service can be configured to receive and execute jobs

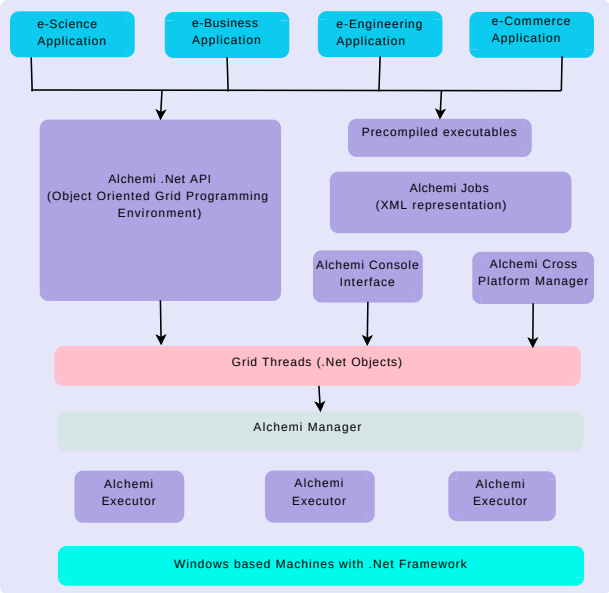


Figure 2: Alchemi architecture.

both in voluntary and non-voluntary modes. Alchemi exposes run-time machinery and a programming environment (API) required for constructing Desktop Grid applications. The core Alchemi middleware relies on the master-worker model - a manager is responsible for coordinating the execution of tasks sent to its executors (desktop machines). The layered architecture of the Alchemi system is shown in Fig. 2.

2.1 Programming and Application Model

Alchemi has supporting APIs for following job execution models: Thread Model and Job model. The Thread Model is used for applications developed natively using the .NET Alchemi application programming framework. This model defines two main classes including GThread and GApplication. A GThread is the simplest unit of task that can be submitted for execution. One or more GThreads can be combined together to form a GApplication such as executing parallel threads over Alchemi to do distributed image rendering. The Job Model has been designed to support legacy tasks developed using different programming platforms (such as C, C++, Java). These legacy tasks can be submitted to the Alchemi through the Cross Platform Manager. ASP.NET web service interface hosts the Cross Platform Manager service which can be invoked by generalised Grid schedulers such as GridBus broker [12].

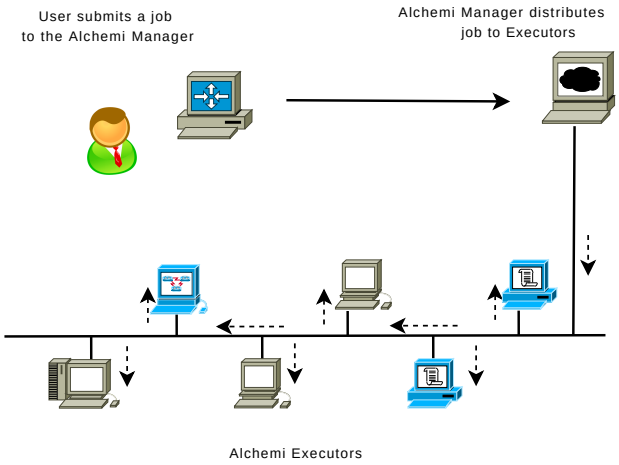


Figure 3: Job submission and execution on Alchemi.

Fig. 3 illustrates the job submission and execution process involving Alchemi Manager, Executor and users. Application users submit their job directly to the local Alchemi Manager. This submission can be done either through Alchemi's API if invoked from .NET platform or Cross Platform Manager's web service interface. Once the job is submitted, the Manager queues it for future consideration by the scheduler. The Alchemi scheduler queries the status of each executor and finally dispatches the job to the available one. After processing, executors send back back the job output to the owner via the central Manager.

3 Alchemi-Federation System Design

This section presents comprehensive details about the software services that govern the overall Alchemi-Federation system. Fig. 4 shows the layered architecture of the proposed software system. We start with describing the Grid-Federation Agent Service, the core resource manager responsible for Alchemi-Federation wide resource discovery and application scheduling.

3.1 Grid-Federation Agent Service

The GFA service is composed of three software entities including a Grid Resource Manager (GRM), Local Resource Management System (LRMS) and Distributed Information Manager (DIM) or Grid Peer.

3.1.1 Grid Resource Manager (GRM)

The GRM component of a GFA exports the local Alchemi site to the federation and is responsible for coordinating federation wide application scheduling and resource allocation. The GRM is responsible for scheduling the locally submitted jobs in the federation. Further, it also manages the execution of remote jobs in conjunction with the local resource management system. The finer details on the general Alchemi-Federation resource sharing model and GFA service can be found in the paper [9]. Here, we only focus on the software implementation details of the components. This software module is implemented in C-sharp. As shown in Fig. 4, GRM interacts with other software modules including LRMS and Grid peer. Both LRMS and Grid peer software modules are implemented in C-sharp so they have no inter-operational issues.

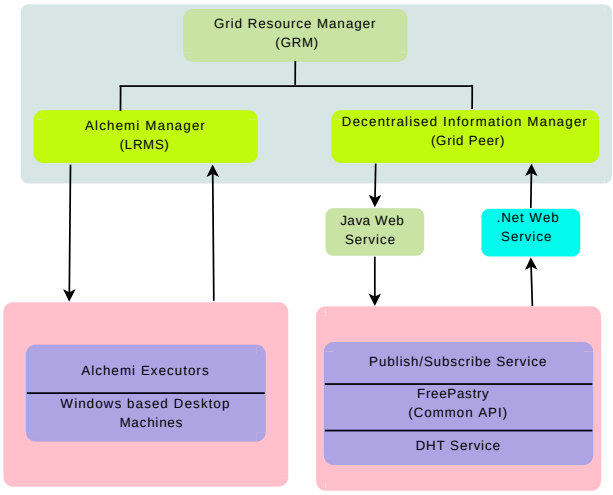


Figure 4: A block diagram of Alchemi GFA software interfaces and their interactions.

3.1.2 Local Resource Management System (LRMS)

The LRMS software module extends the basic Alchemi Manager module through object oriented software inheritance. Additionally, we implemented the following methods for facilitating federation job submission and migration process: answering the GRM queries related to job queue length, expected response time and current resource utilization status. LRMS inherits the capability to submit applications to Alchemi executors from the basic Alchemi Manager module. The Alchemi executors register themselves with the Manager. This in turn keeps track of their status and availability. In the Alchemi system, a job is abstracted as a Thread object that requires a sequential computation for a fixed duration of time. The executors return the completed threads directly to the LRMS module which in turn sends it to the GRM module. Finally, the GRM module directly returns the thread output to the responsible remote GRM in the federation. In case the job thread was submitted by a user local to the Alchemi cluster, then the LRMS directly returns the output without GRM intervention.

3.1.3 Grid Peer or DIM

The Grid peer module in conjunction with publish/subscribe indexing service performs tasks related to decentralised resource lookups and updates. The details on how the GRM component encapsulates the Resource Lookup Queries (RLQs) and Resource Update Queries (RUQs) can be found in the paper [7]. Here we discuss the details on interaction protocols between the software modules including Grid peer and publish/subscribe service. Grid peer module is implemented in C-sharp while the publish/subscribe service is implemented using the Java platform. To resolve the inter-operational issues between these two services we implemented web service interfaces for both the modules. Publish/subscribe index service exposes the method for invoking RLQ and RUQ processes through a web service interface (refer to Fig. 4).

Apache Tomcat container hosts the publish/subscribe application service. Apache Tomcat is the servlet container that implements the Java Servlet and JavaServer Pages technologies. The specifications for Java Servlet and JavaServer Pages are developed by Sun under the Java Community Process. We utilised the Apache Axis 1.4 SOAP (Simple Object Access Protocol) engine for parsing the XML messages. SOAP is a communication protocol put forward by W3C for exchanging structured information among software entities running in different hosting environment. It is an XML based protocol that is based on three specifications: an envelope that defines a framework for describing what is in a message and how it should be processed, a set of encoding rules for expressing instances of application-defined data types and methods, and a convention for representing remote procedure calls (RPCs) and responses. Similarly, Grid peer implements a .NET web service for receiving the query responses from the publish/subscribe index service. This web service is implemented using ASP.NET and is hosted by the Microsoft Internet Information Service 6.0 (IIS).

4 Resource Discovery Service

The resource discovery service organises data by maintaining a logical d-dimensional publish/subscribe index over a network of distributed Alchemi GFAs. Specifically, GFAs create a Pastry overlay, which collectively maintains the logical publish/subscribe index to facilitate a decentralised resource discovery process. We have presented the finer details about the resource discovery service and spatial index in the paper [7]. Here, we only focus on implementation details such as design methodology, programming tools, libraries etc. The resource discovery service was developed using the core Java programming libraries and FreePastry[3] P2P framework. We utilised the Eclipse Integrated Development Environment (IDE) for system implementation and testing.

Our resource discovery system implementation followed a layered approach known as OPeN architecture. The OPeN architecture consists of three layers; the Application layer, Core Services layer and Connectivity layer. The Application layer implements all the logic that encapsulates the query requirements of the underlying Alchemi Federation environment. The Core services layer undertakes the consistency and management of virtual d-dimensional indices. The Connectivity layer provides services related to Key-based routing, overlay management and replica placement. The Application service, in conjunction with the Core services, undertakes the resource discovery tasks including distributed information updates, lookups and virtual index consistency management. While the maintenance of Connectivity layer is left to the basic DHT implementations such as FreePastry, the modules for Application and Core services layer is developed using the standard Java libraries. For Connectivity layer services we utilised the FreePastry framework.

4.1 FreePastry

FreePastry is an open source implementation of well-known Pastry routing substrate. Pastry protocol was proposed by Microsoft's systems research Group Cambridge, United Kingdom and Rice University's distributed system group. Pastry offers a generic, scalable and efficient routing substrate for development of P2P applications. It exposes a Key-based Routing (KBR) API and given the Key K, Pastry routing algorithm can find the peer responsible for this key in $\log_b n$ messages, where b is the base and n is the number of peers in the network. Nodes in a Pastry overlay form a decentralised, self-organising and fault-tolerant circular network within the Internet. Both data and peers in the Pastry overlay are assigned Ids from 160-bit unique identifier space. These identifiers are generated by hashing the object's names, a peer's IP address or public key using the cryptographic hash functions such as SHA-1/2. FreePastry is currently available under BSD-like license. FreePastry framework supports the P2P Common API specification proposed in the paper [3].

Common API abstracts the design of P2P applications into three layers tier 0, tier 1 and tier 2. Key-based routing at tier 0 represents the basic capabilities that are common to all structured overlays. The Common API specification hides the complexity of the low level P2P protocol implementation by defining a common set of interfaces to be invoked by higher level application services. These application services can invoke standard KBR procedures independent of the actual implementation. In other words, a KBR implemented using the Chord, Pastry or CAN will not make any difference to the operation of the higher level application service. At tier 1 abstracts more higher level services built upon the basic KBR or structured overlays. Examples include DHTs, Decentralised Object Location and Routing (DOLR), and group anycast/multicast (CAST). Application services at tier 3 such as CFS, PAST, Scribe can utilise one or more of the abstractions provided by tier 2.

5 Coordinated Scheduling

The resource discovery system is extended to provide the abstraction/facility of a P2P tuple space for realising a decentralised coordination network. The P2P tuple space can transparently support a decentralised coordination network for distributed Alchemi GFAs. The P2P tuple space [5] provides a global virtual shared space that can be concurrently and associatively accessed by all participants in the system and the access is independent of the actual physical or topological proximity of the tuples or hosts. The Grid peers maintaining the tuple space undertake activity related to job load-balancing across the Alchemi grids. Alchemi GFAs on behalf of local users inject *Resource Claim* object into the decentralised coordination space, while Alchemi grids update their resource status by injecting a *Resource Ticket*. These objects are mapped to the DHT-based coordination services using a spatial hashing technique.

A coordination service on a DHT overlay is made responsible for matching the published resource tickets to the subscribed resource claims such that the resource ticket issuers are not overloaded. Every Coordination service in the overlay owns a part of the spatial space governed by the overlay's hashing function (such as SHA-1).

[3] http://freepastry.rice.edu/FreePastry

In this way, the responsibility of the load-distribution and coordination is completely decentralised. Note that, both resource claim and resource ticket objects have their extent in the d-dimensional space. The finer details on how decentralised coordination is enabled among distributed Alchemi grids can be found in the article [8].

6 Deployment and Bootstrap

6.1 ManagerContainer

The ManagerContainer Class loader is responsible for instantiating the Classes that implement the GFA functionality (such as the GRM, LRMS, Grid Peer, and Alchemi Executors) in the Alchemi-Federation system. Additionally, ManagerContainer also initialises the Publish/Subscribe Index web service. The Index service initialisation process includes: (i) booting the node hosting the index service into the existing Pastry overlay if one exists, otherwise start a new overlay; (ii) if this is the first node in the overlay then also compute the division of logical index space at the fmin level else send a node.join(keys) message to the overlay to undertake the ownership of Index keys. Note that FreePastry takes care of the tasks related to routing table, leaf set and neighbour set maintenance. Our Application service is only concerned with coordinating proper distribution and migration of logical Index keys.

6.2 Tomcat Container

Tomcat servlet container hosts the Publish/Subscribe Index service. It exposes an API called TriggerService (int PortName, String BootStrapServerName, int BootStrapPort) to the ManagerContainer service for invoking the Index service. The values for API call parameters PortName, BootStrapServerName and BootSTrapPort are maintained in a configuration file accessible only to the ManagerContainer. Other APIs that Tomcat container exposes include SubmitRLQ(String Object) for submitting RUQs, SubmitRUQ(String Object) for submitting RUQs and SubmitURLQ(String Object) for unsubscribing from the Index service once an application has been successfully scheduled. These methods are invoked by the Grid peer service whenever an application is submitted to the GRM for scheduling consideration.

7 Performance Evaluation

In this section, we evaluate the performance of the software system in a resource sharing network that consisted of federation of 5 Alchemi desktop grids as shown in the Fig. 5. These desktop grids were created in three different Laboratories (Labs) including Microsoft .Net Lab, Masters student Lab 1 and 2 within the Computer Science and Software Engineering department at the University of Melbourne. The machines in these Labs are connected through Local Area Network (LAN). The LAN environment has a data transfer capability of 100 MB/sec (megabytes per second). Ethernet switches of these Labs inter-connect through the firewall router. Various system parameters were configured as follows:

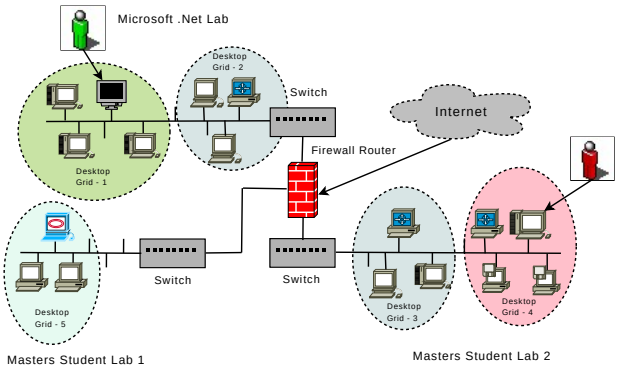
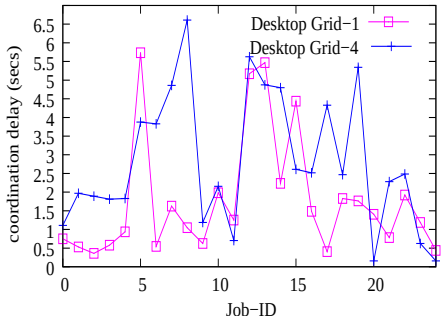


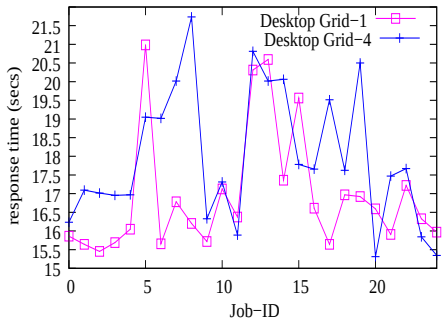
Figure 5: Alchemi-Federation testbed setup.

- Pastry network configuration: Both Grid peer nodeIds and publish/subscribe object IDs were randomly assigned and uniformly distributed in the 160-bit Pastry identifier space. Other network parameters were configured to the default values as given in the file *freepastry.params*. This file is provided with the FreePastry distribution.
- Resource Configuration: Every Alchemi GFA was configured to connect to different number of executors (refer to Fig. 5). The Alchemi manager periodically reports the resource status/configuration to the GFA as given by the resource ticket publish interval. The Alchemi grids running the GFA component had Windows XP as the operating system running on Intel chips. The processor were allocated to the jobs in the space-shared mode.
- Publish/Subscribe index space configuration: The minimum division f_{min} of logical d-dimensional publish/subscribe index was set to 2, while the maximum height of the index tree, f_{max} was constrained to 5. The index space had provision for publishing resource information in 4-dimensions including

number of processors, p_i their speed, μ_i, operating system type, ϕ_i, and processor architecture, x_i. This index configuration resulted into 16 (2^4) Grid index cells at f_{min} level. On an average, 3 index cells are hashed to a Grid peer's publish/subscribe index service in a network of 5 Alchemi sites.



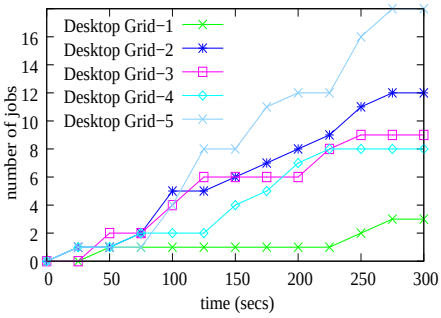
(a) Job-ID vs average coordination delay (secs)



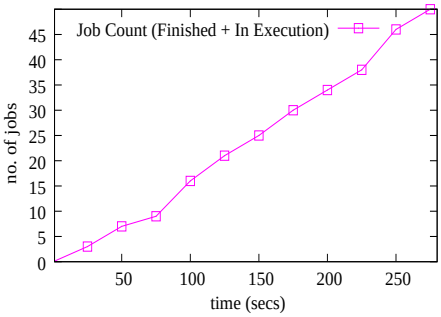
(b) Job-ID vs average response time (secs)

Figure 6: Job perspective.

- Workload configuration: The test application was a windows executable (source code written using c-sharp) that computed whether a given number is prime or not. In order to introduce processing delays, the process was made to sleep for 10 seconds before it could proceed to check the prime condition. A simple brute force algorithm was implemented to check the prime condition for an number. The brute force algorithm consists of dividing the number by every possible divisor, up to the number. If exactly 2 factors are found, it's prime. However, if more than 2 factors are found, then the number is not prime (it is composite).
- Resource claim and resource ticket injection rate: The GFAs inject resource claim and resource ticket objects based on the exponential inter-arrival time distribution. The value for resource claim inter-arrival delay ($\frac{1}{\lambda_l^{in}}$) was fixed to 10 secs. While the



(a) no. of jobs vs time (secs)



(b) no. of jobs vs time (secs)

Figure 7: Resource perspective.

inter-arrival delay ($\frac{1}{\lambda_u^{in}}$) of resource ticket object was fixed to 15 secs. The inter-arrival delay in ticket injection is considered same for all the GFAs/Grids in the system. We configured 2 GFAs/Grids (Desktop Grid-1 and Desktop Grid-4) to insert resource claim objects into system with the delays as described. The users in Desktop Grids - 1 and 4 submit 25 resource claim objects over the experiment run at an exponential inter-arrival delay. While the injection of resource ticket object is done by all the GFAs/Grids in the Alchemi-Federation system.

7.1 Discussion

The experiment measured the performance of the software system with respect to the following metrics: average coordination delay and average response time. The performance metric coordination delay sums up the latencies for: (i) resource claim to reach the index cell; (ii) waiting time till a resource ticket matches with the claim; and (iii) notification delay from coordination service to the relevant GFA. While the average response time for a job is the delay between the submission and arrival of execution output.

Fig. 6(a) depicts results for the average coordination delay in seconds for each job submitted during the experiment period. We observed that jobs across the Desktop Grids -1 and 4 experienced varying degree of coordination delay. As described earlier the coordination delay directly affects the overall response time for jobs which is evident from the Fig. 6(b).

Fig. 7(a) shows how the job load was distributed over the Alchemi grids. We observed that Desktop Grid -1 executed least number of jobs i.e. 3 jobs, while Desktop Grid -5 located in Master's student Lab 1 executed highest number of jobs i.e. 18 jobs over the experiment run. Overall, the resources performed reasonably well as it is seen in the Fig. 7(a). In Fig. 7(b), we show the details on number of jobs finished and under execution across the Alchemi-Federation over the experiment run time.

8 Conclusion

In this paper, we have described an Alchemi-Federation software system. We have strictly followed an Object Oriented Design (OOD) methodology in architecting and implementing the Alchemi-Federation system. Our existing Alchemi-Federation testbed consisted of Alchemi Grids distributed over three different Labs of the department. These Labs are protected from the malicious users by a firewall router that inhibits any connection from or to the machines that do not belong to the domain. In future work, we intend to overcome this limitation of Alchemi GFA service by implementing the cross-firewall communication capability. Such extension to the Alchemi GFA will support creation of Internet-based federation of Alchemi Grids that belong to different firewall control domains.

Our software platform can be utilised to develop other distributed applications such as P2P auction and distributed storage framework. Currently, our platform provides services for aggregating the distributed computational resources. We also intend to study the query load-imbalance issues at the peers in a windows-based Grid computing environment where the resource attribute distribution tends to be skewed. In future work, we intend to incorporate decentralised reputation management frameworks such as PeerReview [4] and JXTA Poblano [13] in the Alchemi-Federation system. These reputation management systems will aid in facilitating a secure and trustworthy system for the participants to interact. Further, we are also considering integrating the PeerMint credit management system. PeerMint is a decentralised credit management application that has been developed using the FreePastry platform.

References

[1] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, pages 4–10. IEEE Computer Society, Los Alamitos, CA, USA, 2004.

[2] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel Distributed Computing*, 63(5):597–610, 2003.

[3] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common api for structured peer-to-peer overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, February 2003.

[4] P. Durschel. The Renaissance of Decentralized Systems, Keynote talk at the 15th IEEE International Symposium on High Performance Distributed Computing, Paris, France. 2006.

[5] Z. Li and M. Parashar. Comet: A scalable coordination space for decentralized distributed environments. In *HOT-P2P '05: Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 104–112. IEEE Computer Society, Los Alamitos, CA, USA, 2005.

[6] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal. *Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework, High Performance Computing: Paradigm and Infrastructure*. Laurence Yang and Minyi Guo (editors), Wiley Press, New Jersey, USA. Fall 2004., 2004.

[7] R. Ranjan, L. Chan, A. Harwood, R. Buyya, and S. Karunasekera. A scalable, robust, and decentralised resource discovery service for large scale federated grids. Technical Report GRIDS-TR-2007-6, Grids Laboratory, CSSE Department, The University of Melbourne, Australia, 2007.

[8] R. Ranjan, A. Harwood, and R. Buyya. Peer-to-peer tuple space: a novel protocol for coordinated resource provisioning. Technical Report GRIDS-TR-2007-14, Grids Laboratory, CSSE Department, The University of Melbourne, Australia, 2007.

[9] R. Ranjan, A. Harwood, and R. Buyya. A case for cooperative and incentive based coupling of distributed clusters. *Future Generation Computer Systems, In Press, Accepted Manuscript, Elsevier Science, The Netherlands*, Available online 15 June 2007.

[10] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware'01: Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–359. SpringerLink, Heidelberg, Germany, 2001.

[11] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, San Diego, California, USA*, pages 149–160. ACM Press, New York, NY, USA, 2001.

[12] S. Venugopal, R. Buyya, and L. Winton. A Grid Service Broker for Scheduling distributed e-Science Applications on Global Data Grids. *Concurrency and Computation: Practice and Experience, Wiley Press, New York, NY, USA*, 18(6):685–699, 2006.

[13] W. Yeager and J. Williams. Secure peer-to-peer networking: The JXTA example. *IT Professional, IEEE Educational Activities Department, Piscataway, NJ, USA*, 4(2):53–57, 2002.