Sigiri: Uniform Abstraction for Large-Scale Compute Resource Interactions

Eran Chinthaka Withana and Beth Plale
School of Informatics and Computing, Indiana University
Bloomington, Indiana, USA.
{echintha, plale}@cs.indiana.edu

***Abstract*—Scientists who conduct mid-range computationally heavy modeling and analysis often scramble to find sufficient computational resources to test and run their codes. The science they carry out is not petascale or even terascale science but the computational needs often go beyond what can be satisfied by their university. With the maturation of Grid computing facilities and recent explosion of cloud computing data centers, mid-scale computational science has more options to satisfy computational needs. This paper focuses on a simple abstraction for interaction with heterogeneous resource managers spanning grid and cloud computing, and on features that make the tool useful for the mid-scale physical or natural scientist. A key aspect of the service is its support for multiple standard job specification languages and the ability for the user to directly interact with the service, removing the delay that can come through layers of services.**

I. Introduction

Scientists who conduct mid-range computationally heavy modeling and analysis often scramble to find sufficient computational resources to test and run their codes. The science they carry out is not at petascale but this kind of application forms the backbone of scientific discovery as its numbers are large. These users have needs that can exceed the computational resources of single university, but may not be well suited to large scale computational resources such as the TeraGrid[13] either because of their size or interactive needs. The GWAVA gradient wind model[31] coupled with the ADCIRC coastal ocean model[41], for instance, has been used with success to model the effect of storm surge on the coastal shores of North Carolina. The model requires about 600 CPU hours per simulation and requires hundreds of simulations to generate enough results to be useful. The ADAS[45] and WRF[32] runs of the LEAD [17] system execute short term forecasts for use in applications in energy and agriculture. The ADAS/WRF workflow requires about 512 CPU hours to execute and generate a few gigabytes of data. These scientists, most notably found across the geosciences, need access to resources that are cheap, available, and most importantly easy to set up and administer.

Grid computing can and has been viewed as an answer to the problem of mid-scale computational research through well known grids such as the Open Science Grid (OSG)[4] and TeraGrid, and the European Grid Initiative[2] that bring together geographically distributed compute resources. Grid computing has advanced cyberinfrastructure and software architectures in significant ways, resulting in solutions to distributed security, metascheduling, and data movement. On the other hand, the computational resources available through Grids are often large batch oriented system that under even moderate loads can result in long queue times. Additionally, grid computing makes no claims to access transparency in that failures in the underlying infrastructure generally propagate upward to the application for handling, complicating its use by domain scientists. The quota system on CPU usage requires that maximum resource needs be known in advance and research projects pre-approved through a grants approval process. Finally working through the cyberinfrastructure to get access to the computational resource requires a high level of computer science expertise if one is to utilize the resource in an optimal way. Cloud computing on the other hand offers a complementary solution to grid computing that features high availability, a pay-as-you-go model, and on-demand allocation of what is claimed to be a limitless resource[8]. It has its own disadvantages. For one, it has a payment policy that does not work well with university research cost models.

One model that has emerged in support of mid-scale science is that of researchers engaging with computational resources through workflow systems. The workflow can be a human task, a semi-structured workflow where both automated components and human components are part of the workflow, it could be a script that executes tasks in some predefined sequence, or it could be a task graph orchestrated by a workflow engine. The utilization of heterogenous compute resources can be accomplished by a hybrid workflow model[36] where a workflow is decomposed and passed to computational resources whereever they are available. For instance, in [36] we demonstrate the hybrid workflow model that partitions a workflow across compute platforms to take advantage of what a compute platform might have to offer. We demonstrated a weather forecast workflow that ran in a strict 1-hour timeframe. It is made up of 16 computational tasks, largely serial, one of which requires 512 cores for 20 minutes, and the ending step is a rendering piece where 8 tasks run in parallel. The rendering piece of the workflow can utilize a local Windows HPC cluster, but the computationally heavy piece requires an HPC resource but only for the 20 minutes where 512 cores are required. The hybrid workflow model gives flexibility to utilize computational resources where they can be found. A resource abstraction layer to which sub-workflows can be submitted is a key piece of the hybrid workflow model. In order to minimize total turnaround time in this heterogeneous environment, the resource abstraction layer must be able to monitor the back end resource well enough to respond instantly when the sub-workflow has completed.

The broader vision of our research is on giving mid-scale science applications options to achieving low cost execution that minimizes turnaround time and maximizes the successful completion rate over a set workflows where the workflow set

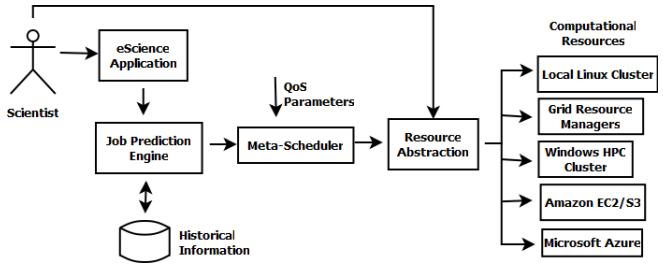


Fig. 1: Research Vision

is drawn from multiple users submitting workflows over a fixed period of time. The solution has two primary pieces as shown in Figure 1: the first is a metascheduler that draws from AppLeS[11] and GRADS[10] in its use of historical information to anticipate future activity. The second is a resource abstraction service, which shares aspects in common with Nimrod/G[12]. Our focus in the latter is on a simpler abstraction to deploying tasks on compute resources that spans grid and cloud computing, and focuses on features that make the tool useable by the small-scale physical or natural scientist who runs mid-sized computational jobs from a small-scale research lab.

The contribution of this paper is the resource abstraction layer implemented as a web service that provides a uniform abstraction layer over heterogeneous compute resources. The service supports the standard job specification languages Job Submission Description Language (JSDL)[7] and The Globus Resource Specification Language (RSL)[22], and directly interacts with resource managers so requires no grid or meta scheduling middleware. We demonstrate experimentally that the architecture is scalable and offers good performance.

The remainder of the paper is organized as follows. In Section II we discuss motivating challenges. Design and architecture of the system is discussed in Section III and Section IV presents the performance evaluation of the implementation. We discuss related work in Section V followed by conclusions in Section VI.

II. Motivation

With the advent of cloud computing, the mid-range scientist has multiple options, more platforms to choose from to run their experiments. However, taking advantage of this plentiful setting is difficult because of numerous barriers. Applications must be ported and accounts and allocations must be secured. If the application is packaged with a guest OS and run in a virtual machine, the package must be prepared. Commercial Cloud resources require monetary expenditure that research grants and universities are not well suited to accommodate. Grid or traditional high performance computing resources are batch oriented, keeping jobs in queues for possibly lengthy periods. If a user has access to a departmental cluster, it may not be sufficient. While we recognize the importance of all these challenges, we focus on a single solution in this paper, a component in a modular and light weight solution that provides a uniform abstraction or interface to heterogeneous back end compute resources.

In our model of execution, the resource abstraction layer (or service) sits between a meta-scheduler and the local scheduler on a compute resource if the local scheduler exists. This is depicted in Figure 1 where an application such as a workflow engine submits jobs to the Job Prediction Engine. The Job Prediction Engine examines historical information about the job, the service history of the set of available resources, and the context of other jobs/workflows in the immediate past, the present, and the future. The computational systems that best maximize the user profile is selected. The job is then passed to the metascheduler which further refines the system choice based on reliability estimates of the systems. The job is handed off to the resource abstraction component which deploys the job to the selected resource and monitors progress. The user and client tools can directly interact with the resource abstraction component ensuring, for instance, that the client is instantly notified when a job completes. For real-time, hybrid workflows, delays in hearing about completion of a job can be costly because workflows must complete multiple steps on a schedule so cannot afford time-wasting delays.

Grid middleware solutions offer an option to abstracting a set of heterogeneous resource managers through a single unified job management interface. However, Grid middleware tends to be highly complex, needing technically sophisticated system administration skills to deploy and maintain these services. Further, many clusters in the academic setting are not part of a larger scale grid and have to be directly accessed by non-uniform vendor specific resource managers.

Meta-scheduling and scheduling algorithms partially address this problem. They are predominantly focused however on improving batch queue times, reducing job execution times and securing resources as and only as needed. With grid computing such as with the TeraGrid, queue times can be significant and be dependent on factors such as total system load, wall clock limits on a job, etc. A batch queue by nature implies mediating access to a finite resource.

Cloud providers provide access to seemingly unlimited resources so wait times are effectively nonexistent. Jobs do however suffer seconds to minutes start up overheads within virtualized environments[14][18][20][29], and thus are a major source of latency that we propose to reduce. Scheduling algorithms also focus on optimal utilization of relatively homogeneous grid or cluster resources through resource allocation algorithms. Maximal utilization of a resource does not make sense either in a domain of unlimited resources, or at least not a problem that a user needs to deal with. But clouds, particularly when taken across available cloud providers[1][3] present a heterogeneous set of resources with different hardware and software configurations that can be maximally utilized to a user's advantage. Cloud computing environments have spawned significant growth in programming paradigms like MapReduce[15] for data intensive job executions. The data locality and the data dependence between map and reduce tasks[44] are unique to cloud environments.

There are a number of requirements that motivate our solution:

Non-Standard Job Managers: As new compute platforms emerge, such as Amazon EC2[1], Nimbus[28], Windows Azure[3] and with wide variety of cluster resource managers, such as LSF[42], Portable Batch System[38], Load-Leveler[26], Simple Linux Utility for Resource Management[43] vendors have introduced wide variety of resource management interfaces. With more options

in hand, complexities in interacting with wide variety of computational resources becomes a challenge for scientists, especially for non-computer scientists.

Startup Overhead: Large-scale computing resources have startup overheads associated with job executions. For example, to execute a scientific job inside virtual machine based cloud computing environments the virtual machines must be started up and prepared before the execution of the jobs. If the startup overheads are significant compared to the job execution times, then reducing these startup overheads can improve the execution times of scientific jobs. Accurate prediction of next jobs to be executed by examining across a set of workflows enables advance allocation and preparation of resources, cutting down or hiding the startup overheads in job executions.

Scalability and Reliability: There are systems implemented to work with different types of computational resources. For example, grid computing middleware abstracts job managers and has succeeded despite battling through scalability and reliability issues [30].

Non-Grid Resources: Scientists have access to computational resources which are neither part of a grid nor have the intention of being part of a grid. Departmental clusters, community clusters are good examples for these types of resources. Also the science gateways, which support a community of users will have to support variety of resources. These scientists are looking for resources which are more readily available than shared resources and they have a need for a lightweight job manager with minimal installation requirements.

New Platforms for Scientific Job Executions: Windows HPC clusters are gaining popularity as platforms to run scientific applications. The introduction of distributed job execution frameworks like Dryad [27], scientific workflow workbenches like Trident [9] and with the porting of widely used scientific applications like WRF [32] to run on Windows, scientists are already migrating to these resources. With these advances, scientists should be able to add Windows HPC clusters in to the available set of computational resources to run their scientific experiments. But with the current systems its quite challenging for existing workflow systems and gateways to interoperate with these new clusters. Also cloud computing, as discussed in the introduction, is a potential alternative for on-demand computing beyond the promise of grid computing. The elasticity and pay as you go model will be applicable for scientific needs where experiments have a surge of compute needs.

Finally, our larger vision extends to include the use of prediction techniques that capture and deal with the variability of heterogenous compute resources. While not part of this paper, we envision prediction algorithms using historical data are employed in scheduling to predict the execution times of an application. On a cluster, this can be done by looking at the history of the application alone. But cloud environments with their different hardware configurations, the execution time predictions must factor in the non-uniformity of hardware resources. Grid or time-shared resources are designed to allocate a large fraction of computational resources to be consumed to a single job or user[34] whereas in cloud computing systems only a tiny fraction of total resource availability is being used by a given user or a job at any given time. Cloud schedulers must in response scale to support

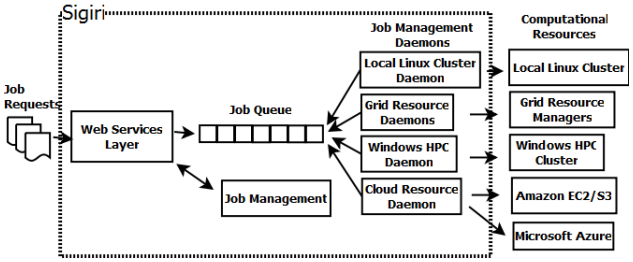


Fig. 2: Sigiri Architecture

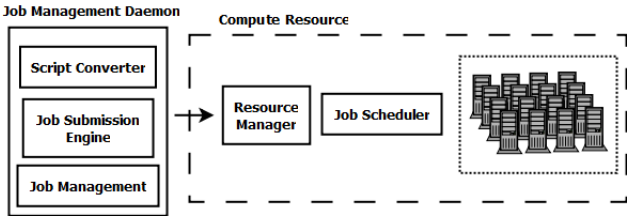


Fig. 3: Sigiri Daemon Interaction with Resource Managers

large number of simultaneous users. Too, the emergence of the semi-structured workflow as a way of bringing the human-into-the-loop of an investigative process results in more smaller-scale and data dependent workflows that are seen in large-scale computations. These workflows present a challenge to the body of work on workflow planning and execution that we discuss in the related work section. Finally, grid computing or time-shared resources provide an abstraction of federated resources[34] in the form of heterogeneous hardware, network and software resources across different administrative domains bound together in support of virtual organizations where the administrative domain supports authentication but membership and authorization are at the virtual organization level. But federated resources or virtual organizations for that matter, are not provided for in a cloud setting.

III. Architecture

We address the requirements of Section II with a light-weight computational resource abstraction framework called Sigri. The design of the service is additionally motivated by the lessons learnt in using grid computing abstractions and dealing with its shortcomings throughout the course of the LEAD project [17].

A. Overall Architecture

Underlying the architecture of Sigiri is an asynchronous eventing model of event publishers and consumers where job requests received at Sigri are published to a queue, as shown in Figure 2, for uptake by daemons representing the heterogeneous computational resources. Each daemon, shown in detail in Figure 3, communicates with the computational resource it represents. Once a job is submitted to the Sigri job queue, the relevant daemon picks up the job, translates it to the job specification language of the computational resource, if any, and submits it to the resource manager. The same daemon is then responsible for monitoring the job to completion and updating the system on the current state of the job. This decoupled architecture enables robustness in Sigiri under failures in underlying computational resources and at

the same time provides an extensible framework capable of being extended to additional computational resources.

The system is distributed in that the components of the system (Figure 2) can be deployed on different machines. If a new computational resource needs to be added to the system, the system administrator can add a new daemon to the system and make it run inside the computational resource itself, without making any changes to the existing Sigiri deployment. This flexibility enables eScience applications already using Sigiri to be minimally impacted by the addition of new computational resources, but also takes advantage of the computational capabilities of a new resource.

At the front end of Sigri is a web service single access point for clients, such as workflow clients, to submit and manage jobs. The job requests are queued and persisted and the request is tracked and responded through a unique internal job identifier. De-coupling the job submission with the client request, the system can be more robust than existing solutions. Also it enables to sustain the initial response time and to surge protect the rate of resource manager's job submission. The sustained rate of acceptance increases the scalability of the job management system greatly empowering support of large scale workflow systems. This decoupling also helps the system to sustain the communication failures of underlying resources and retry and recover when the system returns to healthy state. This asynchronous job acceptance introduces latencies but the robustness and constant performance out weighs this minimally introduced delays. The client either poll for the job status or can register a callback URL or email to receive notification of job status changes. The web service front end also supports job termination and other job management operations. The service persists the client handles and the resource manager handles and correlates between the two accordingly.

At the back end, each managed compute resource has a light-weight daemon which periodically checks the job request queue, translates the job specification to a resource manager specific language, submits the pending jobs and persists the correlation between the resource manager's job id with internal id. The daemon also handles resource manager specific faults and propagates them to the service to notify the clients. Because of the extra complexities of handling cloud computing resources, Sigiri provides a flexible framework to interact with different providers. The Job management process continuously monitors the state of submitted jobs using a resource manager specific API to retrieve the information. Additional features include the following:

Extensible Daemon APIs: Sigiri implementation supports wide variety of resource managers [26][38][43] found in grid computing resources, Windows HPC clusters, Amazon EC2 [1]. The extensibility of the daemon API enables the integration of wide range of resource managers while keeping complexities of these resources managers transparent to the end users of these systems.

Ability to Integrate Seemlessly with Existing Systems: Sigiri Web service interface, developed as an Apache Axis2[35] Web service, is capable of interoperating with wide variety of clients including workflows, gateway environments, script based environments, etc., Sigiri also has the flexibility of extending its Web service to support different job specification languages. Our current implementation understands both JSDL[7] and RSL[22].

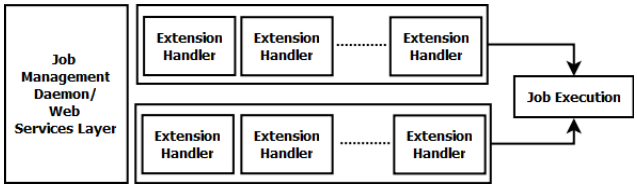


Fig. 4: Sigiri Daemon Architecture for Cloud Resources

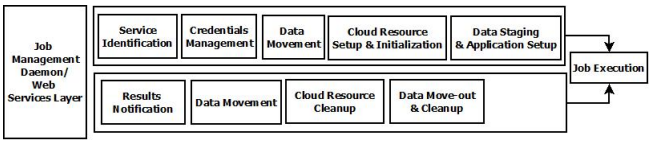


Fig. 5: Prototype Implementation for Amazon EC2

B. Integration with Cloud Computing Resources

We propose an extension based framework where different providers are handled using different extensions which are dynamically loaded and configured to handle security, run jobs and perform required data movements. Since failures are part of job execution, we also incorporated failure handling into the framework so that the extension authors can register these failure handlers to the framework to clean up the resources after a failure. This framework enables scientists to interact with multiple cloud providers within the same system. This extensible framework can be used to add support for new providers and even help to work with different data movement and managements services, without binding to a given provider.

The cloud extension has also the ability of exposing itself as a web service, while also being a daemon that can be accessed using Sigiri Web service.

Figure 4 presents the architecture of Sigiri cloud extension. The in-path of the system, should perform the steps related to preparation of the resource until job submission. The out-path, on the other hand, handles the steps from the completion of job execution to notification of the client. Extensions can be written as modules independent of other extensions, typically to carry out a single task. For example, UserCredential extension can be used to retrieve the user's credentials from a registry and put in to current context. Users will then write a template file, listing the sequence of these extensions. User has the flexibility to write their own extensions or use existing extensions from the pool of extensions that are pre-installed in Sigiri. Sigiri, upon receiving a request to execute a job in a cloud computing resource, looks for a template that can serve the request. Then it loads the extensions defined in this template dynamically and invoke them in order. With the Chain of Responsibility Pattern [39] used within Sigiri cloud extension, we enabled clients to dynamically add extensions to the system and also inject new dependencies in to the system.

Figure 5 presents a typical combination of extensions that can be used to execute jobs in Amazon EC2.

C. Fault Handling

While interacting with cloud computing resources, the system can fault at various stages. For example, after starting

virtual machines in the cloud computing resources, there can be errors in data movement. In these cases, it is important to roll-back the actions performed up to that point. If not cleaned up properly, users will end up paying for resources in-vain and might also create trouble while creating future instance. To address this, we integrated fault handling also in to the extensions and mandated every extension to implement a revoke method. During the fault handling process these revokes are called for already executed extensions, in the reverse order they were invoked.

D. Security

To interact with multiple computational resources, each of which might implement a different security mechanism, poses a challenge to provide security for the clients who access our service and for Sigiri to access the supported computational resources. We provide two levels of security: client level security to secure access to our service, and compute resource security to control access to the computational resources by Sigiri on behalf of the clients.

Client security is handled between the client and the web service enabling both transport level and message level security using SSL and WS-Security[33] respectively. With the support of the Web container, it can be enabled to provide authentication to use Sigiri service. WS-Security, on the other hand provides authentication, authorization and message level encryption capabilities. With the support of Apache Axis2 capabilities, it is also possible to enable WS-Security policy and make the clients negotiate security policies with Sigiri server.

Compute Resource Security. Different computational resources require different types of security credentials. For example, accessing Windows HPC resources require the user to provide basic authentication using username/password combinations whereas cloud computing providers require users to provide security credentials using certificates. Sigiri can be configured to store these credentials internally, remote registry or even in an identity server [40]. This extensibility to work with different security providers enables Sigiri to use different credential types without making them vulnerable to attacks while enabling secure access to the computational resources.

IV. Performance Evaluation

This section details the test scenarios and evaluation metrics of Sigiri over grid computing resources. The test scenarios are similar to the ones used previously by Marru et al.[30], but with the addition of baseline measurements of Sigiri.

A. Experimental Setup

1) Sigiri Hosted Environment: Sigiri daemon is hosted in the gatekeeper of Indiana University BigRed HPC computer. The gatekeeper is a quad-core IBM PowerPC (1.6GHz) with 8GB of physical memory. Sigiri Web service and the database are co-hosted in a box with 4 2.6GHz dual-core processors with 32GB of RAM. Both these nodes were not dedicated for our experiment when we were running tests.

2) Client Environment: The test clients were run on a 128 node Odin Cluster (Research cluster in School of Informatics and Computing in Indiana University). Each node on Odin is a Dual AMD 2.0GHz Opteron processor with 4GB physical memory. All the client nodes were used in dedicated mode and each client is running on separate java virtual machine to eliminate any external overhead. The client side concurrent invocations are managed by SLURM job manager.

B. Evaluation Test Cases

We use a number of baseline, system, and scalability tests to evaluate the performance of Sigiri and to compare the performance with Globus WS Gram v4.0.6 (with additional patches) [19]. We use the following two test scenarios against the three benchmarks below.

Test Case 1: Jobs arrive at Sigiri system as a burst of concurrent submissions from a controlled number of clients. Each client waits for all the jobs to finish before submitting the next set of jobs. For example, during the test with 100 clients, each client sends 1 job to the server making 100 jobs coming to the server in parallel. The job submission script waits until all the 100 jobs to finish before invoking the clients to send next batch of jobs.

Test Case 2: Each client submits 10 jobs having varying execution times in sequence with no delay between submissions, that is, the client does not block upon submission of a job. The failure rate and the server performance, from the clients point of view, are measured and the number of simultaneous clients will be systematically increased. This test case will simulate a loosely parallel job submission with Sigiri system processing medium to low concurrent jobs but without any breathing room between batches of submissions.

1) Baseline Measurements: Figure 6 presents the states of a job within our system and Table I defines the baseline measurements we will be using to evaluate Sigiri.

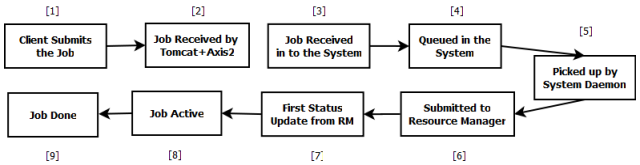


Fig. 6: States of a Job within Sigiri

2) System Benchmarks: Table II defines the system level benchmarks we will be using to evaluate Sigiri.

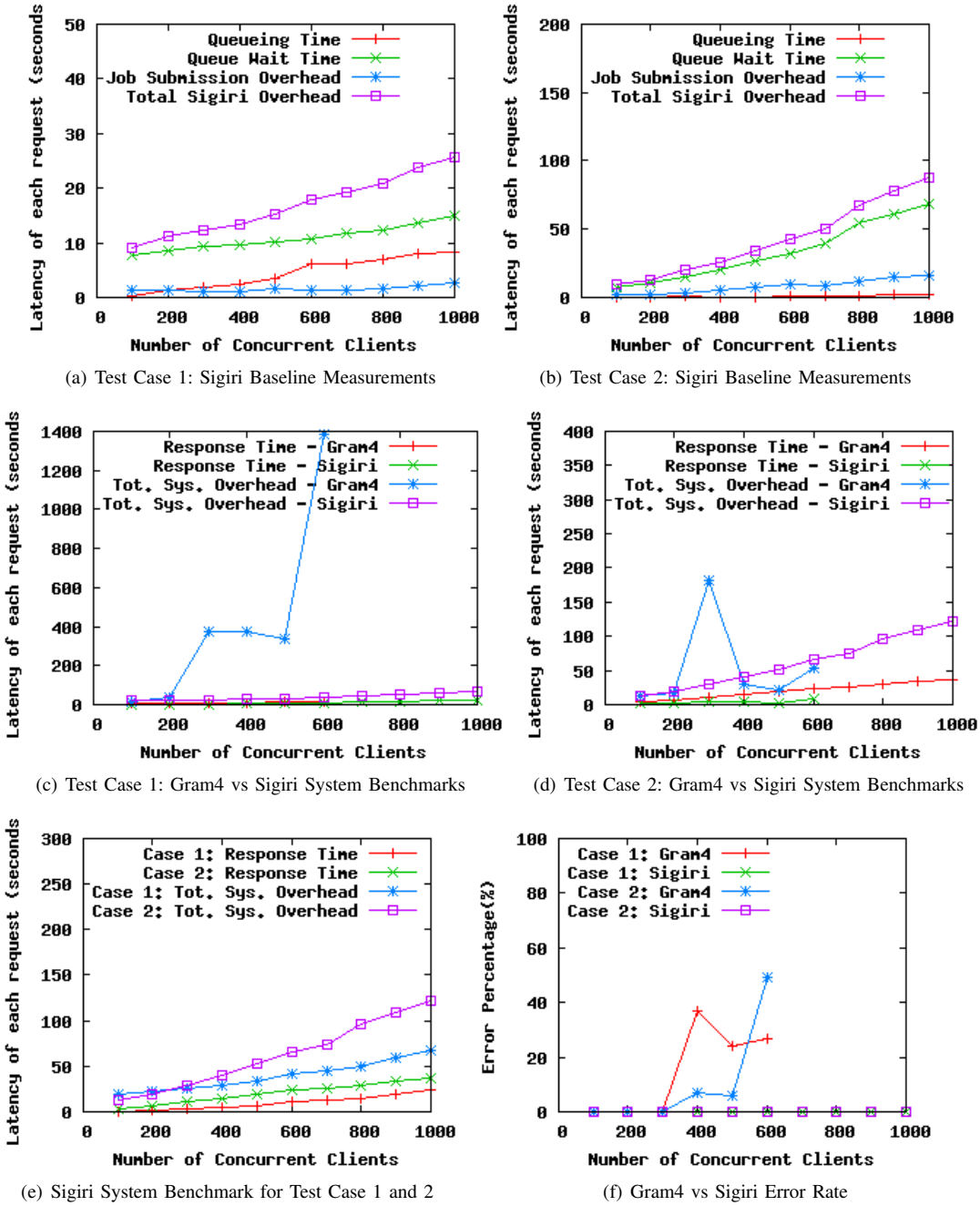
3) Scalability Measurements: In this section we evaluate Sigiri's scalability in comparison to Globus WS-Gram v4.0.6 (Gram4) using both test cases 1 and 2. The metrics are defined in Table III, and they include failure percentage from which one can derive a sense of overall system reliability.

We ran each test, number of clients * 10 times and average the results to get one data point. Each parameter is tested for 100 to 1000 concurrent clients. During this evaluation we ran a total of 110,000 tests. We use the Gram4 experiment results produced in Gram4 evaluation paper[30].

C. Results

Figure 7(a) and 7(b) show Sigiri performance on baseline measurements (Table I) for test cases 1 and 2, respectively. Sigiri performance is steady and has low overhead across varying number of concurrent clients for all the baseline measurements.

Figure 7(e) show the overall system performance of Sigiri for the two test cases. Sigiri is performing better in test case 1, compared to test case 2. Further profiling of the test scenarios



(a) Test Case 1: Sigiri Baseline Measurements

(b) Test Case 2: Sigiri Baseline Measurements

(c) Test Case 1: Gram4 vs Sigiri System Benchmarks

(d) Test Case 2: Gram4 vs Sigiri System Benchmarks

(e) Sigiri System Benchmark for Test Case 1 and 2

(f) Gram4 vs Sigiri Error Rate

Fig. 7: Performance Evaluation Results

revealed that the difference is due to the thread allocation and maintenance inefficiencies of Apache tomcat server and Axis2 server.

Figure 7(c) and 7(d) compares Sigiri system performance against the Gram4 performance for system benchmarks. Even though Gram4 response time is comparable to Sigiri performance, Gram4 Total system overhead is a lot worse than Sigiri. Sigiri could maintain a steady increase in total system overhead with the increase of number of clients. Gram4 server could maintain its performance and also the 0% error rate up to 200 concurrent requests, but beyond that point Gram4 performance has started to degrade. According to previous observations[30], Gram4 server at 600 concurrent requests, for both test cases 1 and 2. Because of this we could not obtain data for Gram4, beyond 600 concurrent clients.

Figure 7(f) shows the variation of error rate against increasing number of clients for Gram4 and Sigiri. For all the test cases, Sigiri never encountered an error and could serve all 110,000 requests. Gram4 on the other hand, encounters trouble when processing reaches more than 400 concurrent clients requests.

In addition to the main goals of providing a uniform

Parameter	Description
Sigiri Job Queueing Time ([4]-[3])	The elapsed time between Sigiri Web service getting the request from Web service's container, queueing in Sigiri queue and returning the job identification number.
Sigiri Queue Wait Time ([5]-[4])	The time a job waits in Sigiri queue before being picked up by Sigiri daemon and submitted to Bigred resource manager.
Sigiri Job Submission Overhead ([6]-[5])	The elapsed time between retrieval of a job from the job queue, convert it to a resource manager specific script, submission of the job to the resource manager and, receipt and update of the first states in the database. In other words this approximately corresponds to the time between the user getting the job identification number from the system and the user getting the first status from the resource manager
Total Sigiri Overhead ([6]-[3])	Total overhead added by Sigiri, during job submission.

TABLE I: Sigiri Baseline Measurements

Parameter	Description
Response Time ([4]-[1])	The elapsed time from receipt of a client request, store that in the system job queue and return the job identification number to the user. In addition to sigiri job queueing overhead, this time also includes the overhead of creating and sending a web service request on client side and tomcat and axis2 overhead on assigning request processing threads and processing the request.
Total System Overhead ([9]-[1])	The elapsed time between client sending the request and getting the job done notification. Since we experienced long queue wait times on Bigred and application execution times are varied, we remove the job queue time and execution time from this measure.

TABLE II: System Benchmarks

Parameter	Description
Response Time	The elapsed time from receipt of a client request and the client receiving job pending status with a job identification from the system.
Total System Overhead	The elapsed time between client submitting a job request and client getting job completed notification. Since we experienced long queue wait times on Bigred and application execution times are varied, we remove the job queue time and execution time from this measure.
Failure Percentage	The percentage of job failures.

TABLE III: Sigiri System Peformance Comparison

resource abstraction for large-scale systems, one of the design goals of Sigiri was to make it a light-weight and scalable framework. The baseline measurement performance results show the low overhead of Sigiri and also its scalability. A 0% error rate across all the test scenarios for all the benchmarks gives strong indication that Sigiri is a reliable framework for use in an eScience environment. Driven by the light-weightness, scalability and reliability of Sigiri, our team has been using it, in parallel with Globus Gram4, within LEAD system as the job submission framework for Grid computing resources. With the LEAD II efforts, we are moving towards the usage of wide-variety of computational resources, including Windows HPC and cloud computing resources and Sigiri seems to be a promissing candidate in that effort.

V. Related Work

The primary objective of the Sigiri job management system is to enable eScience applications to have a choice of back-end computational resources through a unified job management service that scales better than existing approaches and is easy to install and maintain. The extensibility of the Sigiri components eases the task of incorporating new resource manager managed compute resources in a way that minimizes administration overhead. Sigiri in its current form targets filling the need for a light weight job management solution, hence is not directly comparable to comprehensive job management solutions such as the Grid Resource Allocation and Management (GRAM)[19], Condor-G[21], Nimrod/G[12], GridWay[25] and SAGA[23] and Falkon[37] all of which provide uniform job management APIs, but are tightly integrated with complex middleware to address a broad range of problems. Gram, for example, enabled the usage of next generation distributed computing within large-scale scientific applications. It attempted to address needed abstractions but the challenges have increased and the middleware has suffered scalability and reliability issues [30]. We are also aware that some of these problems are being addressed with constant improvements with Gram5. Nimrod/G is using globus toolkit and has the capability of working with other grid middleware services and also is capable of scheduling based on the concepts of computational economy. SAGA has an adaptor model that can be used to interact with different grid computing middleware. Falkon[37] provides a high throughput and scalable solution which also incorporates multi-level scheduling. Sigiri, on the other hand, provides standalone functionality which in itself has attracted academic system administrators to incorporate their resources into eScience systems. Sigiri also supports wide variety of other computational resources, including cloud computing resources, Windows HPC resources, departmental clusters, in addition to the grid computing resources to submit and manage scientific jobs.

Cloud computing providers [1][3] enable scientists to access on-demand computational resources and scientists are exploring[24][28][16][5] the capabilities of using cloud computing resources for eScience applications. Systems are developed to enable access to cloud computing resources, but the capability to enable different types of computational resources, using a single interface still exists as a challenge. Sigiri, enables the usage of these new computational resources and technologies with minimal effort enabling scientists to experiment with these new technologies.

CREAM [6] provides job management through a Web services interface but is designed to support a custom job description language and assumes a grid environment for job submission. Custom job description languages are a burden on workflow system managers, hence Sigiri supports for popular job specifications like JSDL[7] and The Globus RSL[22].

Furthermore Sigiri directly interacts with resource managers so assumes no grid or meta scheduling middleware.

VI. Conclusion

Sigiri will help the mid-scale scientists obtain access to resources that are cheap and available. More importantly it strives to do so with a tool that is easy to set up and administer. It will further enable researchers to run scientific experimentation on a wide-variety of computational resources. With the scalability shown here, and architecture for reliability and light-weight functioning, Sigri can also be used as a complementary framework with other job management frameworks, such as Gram, to help the scientist tap in to additional computational resources, and enable researchers to try out new technologies and computational resources such as Windows HPC, cloud computing infrastructure and new local clusters.

Acknowledgment

The authors would like to thank Suresh Marru and Patanachai Tangchaisin in Pervasive Technology Institute, Rob Henderson and system staff in the School of Informatics and Computing at Indiana University Bloomington, Jenett Tillotson and the administrators of BigRed cluster, Srinath Perera alumni of Indiana University and the members of Data to Insight Center for their support and valuable feedback for this paper and performance evaluations.

References

[1] Amazon elastic computing cloud. Http://aws.amazon.com/ec2/.

[2] The Future European Grid Infrastructure - Towards a common sustainable e- infrastructure. In *Vision Paper prepared for the EGI Workshop - 2007*. EGI Preparation Team.

[3] Windows azure platform. http://www.microsoft.com/windowsazure/.

[4] The open science grid, osg executive board. *Journal of Physics: Conference Series 78:012057*, 2007.

[5] D. J. Abadi. Data management in the cloud: Limitations and opportunities. *IEEE Data Eng. Bull.*, 32:3–12, 2009.

[6] P. Andreetto et al. CREAM: a simple, grid-accessible, job management system for local computational resources. *CHEP 2006, Mumbay, India*, 2006.

[7] A. Anjomshoaa, F. Brisard, et al. *Job Submission Description Language (JSDL) Specification v0.3*. Global Grid Forum, 2004.

[8] M. Armbrust et al. Above the clouds: A berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley., 2009.

[9] R. Barga et al. Trident: Scientific workflow workbench for oceanography. In *IEEE Congress on Services-Part I, 2008. SERVICES'08*, pp. 465–466. 2008.

[10] F. Berman, A. Chien, et al. The GrADS project: Software support for high-level grid application development. *International Journal of High Performance Computing Applications*, 15(4):327, 2001.

[11] F. Berman et al. Adaptive computing on the grid using apples. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):369–382, 2003. ISSN 1045-9219.

[12] R. Buyya, D. Abramson, et al. Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid. In *hpc*, p. 283. Published by the IEEE Computer Society, 2000.

[13] C. Catlett. The philosophy of TeraGrid: building an open, extensible, distributed TeraScale facility. In *ACM International Symposium on Cluster Computing and the Grid*. Published by the IEEE Computer Society, 2002.

[14] J. S. Chase, D. E. Irwin, et al. Dynamic virtual clusters in a grid site manager. In *HPDC*, pp. 90–103. IEEE Computer Society, 2003. ISBN 0-7695-1965-2.

[15] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[16] E. Deelman, G. Singh, et al. The cost of doing science on the cloud: the montage example. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pp. 1–12. IEEE Press, Piscataway, NJ, USA, 2008. ISBN 978-1-4244-2835-9.

[17] K. Droegemeier, D. Gannon, et al. Service-oriented environments for dynamically interacting with mesoscale weather. *Computing in Science & Engineering*, 7(6):12–29, 2005.

[18] R. J. Figueiredo, P. A. Dinda, et al. A case for grid computing on virtual machines. In *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, p. 550. IEEE Computer Society, Washington, DC, USA, 2003. ISBN 0-7695-1920-2.

[19] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *IFIP International Conference, Beijing, China*, 2005.

[20] I. Foster, T. Freeman, et al. Virtual clusters for grid communities. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, pp. 513–520. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2585-7.

[21] J. Frey, T. Tannenbaum, et al. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, 5(3):237–246, 2002.

[22] Globus Alliance. *The globus resource specification language (RSL), specification*. http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html.

[23] T. Goodale, S. Jha, et al. SAGA: A Simple API for Grid Applications. High-level application programming on the Grid. *Computational Methods in Science and Technology*, 12(1):7–20, 2006.

[24] C. Hoffa, G. Mehta, et al. On the use of cloud computing for scientific workflows. *eScience, IEEE International Conference on*, 0:640–645, 2008.

[25] E. Huedo, R. Montero, et al. The GridWay framework for adaptive scheduling and execution on Grids. *Scalable Computing: Practice and Experience*, 6(3):1–8, 2005.

[26] International Business Machines Corporation, Kingston, NY. *IBM Load Leveler: Users Guide*, September 1993.

[27] M. Isard, M. Budiu, et al. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, p. 72. ACM, 2007.

[28] K. Keahey, R. Figueiredo, et al. Science clouds: Early experiences in cloud computing for scientific applications. *Cloud Computing and Applications*, 2008, 2008.

[29] K. Keahey, T. Freeman, et al. Virtual workspaces for scientific applications. *Journal of Physics: Conference Series*, 78:012038 (5pp), 2007.

[30] S. Marru, S. Perera, et al. Reliable and Scalable Job Submission: LEAD Science Gateways Testing and Experiences with WS GRAM on TeraGrid Resources . *TeraGrid Conference*, June 2008.

[31] C. Mattocks and C. Forbes. A real-time, event-triggered storm surge forecasting system for the state of North Carolina. *Ocean Modelling*, 25(3-4):95–119, 2008.

[32] J. Michalakes et al. The weather research and forecast model: Software architecture and performance. In *Proceedings of the 11th ECMWF Workshop on the Use of High Performance Computing In Meteorology*, pp. 156–168. Citeseer, 2004.

[33] A. Nadalin, C. Kaler, et al. Web services security: SOAP message security 1.0 (WS-Security 2004). *OASIS Standard*, 200401:1–20010502, 2004.

[34] D. Nurmi, R. Wolski, et al. The eucalyptus open-source cloud-computing system. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid-Volume 00*, pp. 124–131. IEEE Computer Society, 2009.

[35] S. Perera et al. Axis2, middleware for next generation web services. In *Web Services, 2006. ICWS'06. International Conference on*, pp. 833–840. 2006.

[36] B. Plale, C. Herath, et al. Towards proxy workflow execution in environmental research: Application to vortex2. *Environmental Research Workshop*, July 2010.

[37] I. Raicu, Y. Zhao, et al. Falkon: a Fast and Light-weight tasK executiON framework. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing-Volume 00*, pp. 1–12. ACM, 2007.

[38] Veridian Systems, Mountain View, CA. *OpenPBS v2.3: The portable batch system software*.

[39] S. Vinoski. Chain of responsibility. *IEEE Internet Computing*, 6(6):80–83, 2002.

[40] V. Welch, F. Siebenlist, et al. Security for grid services. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, p. 48. IEEE Computer Society, Washington, DC, USA, 2003. ISBN 0-7695-1965-2.

[41] J. Westerink, R. Luettich Jr, et al. ADCIRC: An Advanced Three-Dimensional Circulation Model for Shelves, Coasts, and Estuaries. Report 3. Development of a Tidal Constituent DataBase for the Western North Atlantic and Gulf of Mexico. 1993.

[42] M. Q. Xu. Effective metacomputing using lsf multicluster. In *1st International Symposium on Cluster Computing and the Grid*, p. 100. IEEE Computer Society, Washington, DC, USA, 2001. ISBN 0-7695-1010-8.

[43] A. Yoo, M. Jette, et al. SLURM: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, pp. 44–60. Springer, 2003.

[44] M. Zaharia, D. Borthakur, et al. Job scheduling for multi-user mapreduce clusters. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55, Apr*, pp. 2009–55, 2009.

[45] J. Zhang, F. Carr, et al. ADAS cloud analysis. In *Preprints, 12th Conf. on Numerical Weather Prediction, Phoenix, AZ, Amer. Meteor. Soc*, pp. 185–188. 1998.