Sigiri: Towards A Light-Weight Job Management System for Large Scale Systems

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

***Abstract*—e-Science applications are often compute and data intensive, requiring large-scale compute systems for execution. Large-scale systems, however, support a variety of resource management interfaces that an end user must adapt to for compute job submission and management. Grid[1] middleware solutions abstract these heterogeneous resource managers and offer 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.**

With the goals of providing a simple, reliable and highly scalable uniform job management, we introduce Sigiri, a light-weight job management and abstraction service. Sigiri supports existing popular job specifications like JSDL[2] and RSL[3]. A Web Service Interface is provided to easily integrate with various scientific systems and each step in job submission and management is decoupled to increase scalability. In this paper, we discuss Sigiri architecture, its simplicity and experiences with incorporating academic research clusters (managed by part-time system administrators) into large-scale escience systems.

I. Introduction

Large scale eScience systems often employ worklfow systems as a means to execute complex sequences of compute tasks without requiring intensive user intervention. Workflow systems support flexible task reordering, reuse, and reconfiguration of the task sequence graph. Workflows are required to encapsulate one or more compute intensive jobs that require large-scale systems to execute in an efficient and timely manner. As large-scale compute resources become more abundant (i.e., Amazon EC2 [4], Teragrid [5], Open Science Grid [6], a private cluster), workflow systems should be capable of working with all possible and available resources. These multiple options maximize turnaround time but throw challenges at workflow adoption by presenting multiple non-interoperable access interfaces. A higher level abstraction across all possible platforms is an ideal solution, but premature as the underlying technologies are still emerging.

From national resources to departmental clusters, vendor-specific resource managers such as Load-leveler[7], Portable Batch System (PBS)[8], Simple Linux Utility for Resource Management (SLURM)[9], and Sun Grid Engine (SGE)[10], are in use. Grid middleware such as Globus[11] provides a uniform interface to interact with compute platforms. In addition to addressing the interoperability of access mechanisms, Grid middleware also enables the sharing, selection, and aggregation of a wide variety of geographically distributed resources including supercomputers, storage systems and data sources with a single security sign-on. Due to the complex and diverse requirements the middleware itself has grown complex and the scientific gateway communities that depend on the Grid for a production environment are demanding more than the best effort service. Moreover the complex middleware requires a high level of technical sophistication in system administrators, which is often not the case in academic departments.

The contribution of this paper is the introduction of a job management solution that supports user needs for a choice of back-end resources on which to execute applications, from Grids to the Cloud architectures which are quickly gaining broad adoption. We have developed a service, called Sigri, that provides a single interface to compute resources. The design goals are for a lightweight easy to manage system, and one that supports the popular job description specifications. The service is additionally designed to be highly scalable through an asynchronous architecture. In this paper we demonstrate the early performance evaluation of the service's adaptability on two highly diverse back-end computational resources. In future work we evaluate Sigri's scalability and ease of use.

The remainder of the paper is organized as follows. In Section II of this paper, we discuss the architecture of the middleware service through identification of various components and client and server interfaces. Section III discusses the strengths of the system and Section IV discusses current and future security models. Section V discusses early in adopting Sigiri into scientific workflow systems. We discuss related work in Section VI and conclude in Section VII with a discussion of ongoing and future work.

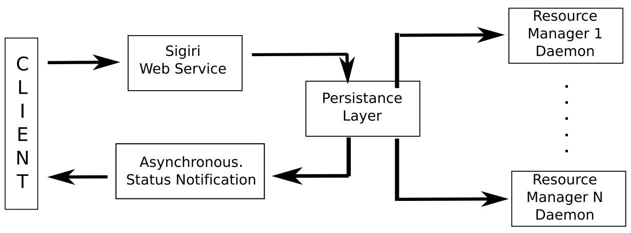
II. Architecture

The middleware tool is designed using a publish-subsribe[12] model so employs a decoupled architecture to improve the robustness and efficiency of the system. Job management is divided into decoupled functional blocks to provide component independence which, when coupled with asynchronous communication, improves the system scalability.

*This research is funded in part by the National Science Foundation under grants ATM 0331480, ACI-0338618l, OCI-0451237, OCI-0535258, and OCI-0504075, and a grant from Microsoft.

Figure 1 identifies the components and depicts their interactions.

- Sigiri Web Service
- Job Submission and Management
- Asynchronous Job Status Notification



C L I E N T
Sigiri Web Service
Persistance Layer
Asynchronous. Status Notification
Resource Manager 1 Daemon
Resource Manager N Daemon

Fig. 1. Sigiri Architecture Diagram

A. *Sigiri Web Service*

: A front-end service provides a web service single access point for platform independent clients, such as workflow clients, to submit and manage jobs to multiple large-scale systems. The asynchronous job submission request is queued and persisted and the request is responded to by means of a unique internal job identifier. The clients have an option of registering a callback URL or email to receive notification of job status changes. The Sigiri web service 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.

B. *Job Submission and Management*

: Referred to in Figure 1 by the Persistent Layer block and some number 'n' of Resource Manger Daemons, 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 script, 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.

The Job monitoring process continuously monitors the state of submitted jobs using a resource manager specific API to retrieve the information. Where supported, like IBM Load leveler[7], Sigiri uses the resource manager C API for registering callbacks during submission and the resource managers notify the registered end point with status changes. When call backs are not supported Sigiri polls the resource manager for status information taking into consideration quality of service metrics like estimated queue wait time, maximum wall time and scaling back accordingly.

C. *Asynchronous Job Status Notification*

: Sigiri clients can register a call back email address to receive notifications of job progression and status. On status changes notified by a resource manager or polled by Sigiri, the monitoring daemon verifies for any registered call back clients and notifies each. It also updates the persistence data for clients to poll.

The job acceptance and submission is decoupled 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.

III. Design Decisions

The design and implementation of the job management system is driven by the needs of large-scale scientific systems. Some of the key considerations include the following:

A. *Abstraction of heterogeneous resource managers*

: Resource managers on large-scale systems differ from each other for various reasons including vendor preference, licensing costs, platform dependence and performance. Resource managers accept different job script formats and have different interfaces, but all of them provide ways to submit jobs, monitor progress and many even support call back notifications. In our job management solution, all resource manager specifics are coded into Job Submission And Management module and provided to users as an extensible API. The interface is abstracted to work with different resource managers, and clients send one of the supported standard job description formats. A resource manager-specific plugin translates the job description to specific scripts, submits and manages the job.

Currently Sigiri supports IBM Load Leveler[7], PBS[8] and SLURM[9]; new resource managers can be added with minimal effort. The script parser can be extended to write the new scripts and JobManager class can be extended to implement the submit, cancel, and check status methods. And finally the new resource has to be registered with Sigiri system. The remainder of the job management system is generic and one installation of a server can manage multiple resources.

B. *Regulation of resource manager submissions*

: Each resource manager has a scalability limit on concurrent job submissions. The Sigiri job management components regulate job submissions by throttling and working within the operating limits of resource managers. Since Job Acceptance and Submission are decoupled, clients do not see higher latencies, and the immediate status notifications indicate that the job queuing within the Sigiri system. Initial performance evaluations reveal that the users will get the "job submitted" response within 125ms.

C. *Dynamic Workflow Support*

: Scientific workflows often need to be adaptive and dynamically reconfigurable [13] based on various factors like change in real-time input data, unanticipated output behavior, delays in batch queues or applications exceeding anticipated execution times. Adapting workflows to these environmental changes requires that the workflow system be able to change, cancel, and modify job submission requests. Sigiri exposes this needed functionality through the web service interface. Option features like verifying the status of resource allocations will help workflow clients make informed decisions and reconfigure workflows accordingly.

D. *Monitoring by Push or Pull Models*

: Each resource manager offers different approaches for querying job status information. As a bottom line, every resource manager provides support for polling for status, but certain resource managers, like Load Leveler provides a C API to register a callback. This push based notification model is very efficient as the resource manager can instantly notify status changes as opposed to delays in poll based approach. Sigiri tries to take advantage of push based notifications where available and falls back to poll based approach if the resource manager does not support push.

E. *Ensuring status changes are not missed*

: Scientific workflow tasks are very long running and cannot be risked to miss a completed job due to communication or any other middleware failures. Resource managers typically maintain job status in memory for only a brief period of time after a job is complete, after which the job details are pushed into a historical repository. Secondly, almost every resource manager kills any pending jobs after the specified wall-clock time is reached.

When polling for jobs, there are scenarios where a job status can be missed when the resource manager may remove job status between polls by the job management system. We have additionally seen resource manager demons getting overwhelmed by current load and become inaccessible transiently. To ensure that workflows continue to make progress for any of these reasons or other unexpected communication failures, Sigiri tracks each job based on its execution metrics. When status is not notified for a specified period, it reverts to alternative approaches of acquiring job status. For example, if the job active notification is received and no completion of failure is received after the maximum wall clock time, then history data is queried to find more information about the job and user is notified of the same. Similarly when Sigiri has subscribed to receive notifications from push based resource managers and transient network failures cause status to be missed, Sigiri will also poll the resource manager and its history files directly ensuring no status is missed.

IV. Security

Job management systems should enforce strong security measures to avoid any unauthorized access or malicious attacks to large-scale systems. The Sigiri web service mandates SSL mutual authentication and the client and server certificates are issued and periodically audited through a controlled process. The database connections are tightly controlled as well. Sigiri does not do any further job deletion and all secured connections to the job management system are from a secured workflow stack. These services interact with one direct hop and we argue that transport level security and/or WS-Security is sufficient for securing the web service interface. We currently use grid certificate based mutual authentication, based on SSL, but because of the flexibility of the underlying Web services stack, employing WS-Security based security would be straight forward. Further security improvements are planned as discussed in Section VII.

V. Success Stories

Sigiri is in use as the job manager to support Linked Environments for Atmospheric Discovery (LEAD) [14] cyberinfrastructure. LEAD is a large-scale e-Science system which enables researchers and educators to assimilate real-time observational weather data and configure and run weather forecasts on demand. In support of LEAD workflow submissions, Sigiri is managing jobs to Indiana University's Big Red, a 30.7 Teraflop, 3000 IBM JS21 cores Massively Parallel Processing system managed by Load Leveler. Sigiri has also been tested on National Center for Supercomputing Application's Abe, a 89.47 Teraflop, 9600 Intel 64 cores cluster managed by PBS Resource Manager. The easy installation of Sigiri has motivated adoption and management by a part-time administrator who maintains a 16 core Intel cluster at Howard University and non-grid enabled Indiana University's Quarry, a 896 Intel Xeon cores cluster managed by PBS being incorporated into LEAD workflows.

National grid resources like TeraGrid serve the majority of the e-Science production workflow needs. There are a significant number of research applications which have to be co-located with the data and as the latter undergoes constant changes, scientists prefer to execute them on local academic research clusters. For example, LEAD collaborators at Howard University intend to study upper air data obtained during an intensive field campaign and process that data to display standard meteorological charts used to develop forecasts for operational decision-making. The charts are rendered on different temporal and spatial scales. Additionally the cataloged and stored data will be processed retroactively to develop analysis of the thermodynamic and dynamic conditions that prevailed during the campaign. Through Sigiri, the LEAD workflow engine is able to utilize academic clusters like Howard University's cluster. This feature is anticipated to increase the adaption of more such research applications through scientific workflows.

VI. Related Work

The primary objective of the Sigiri job management system is to enable workflow-driven 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 server side component 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) [11], Condor-G [15], and GridWay [16], all of which provide uniform job management with web service interfaces, but are tightly integrated with complex middleware to address a broad range of problems. Sigiri, on the other hand, provides standalone functionality which in itself has attracted academic system administrators to incorporate their resources into workflow systems. Specifically, systems like Gram provide a unified job interface for the resource managers integrated in to grids. Sigri addresses a key items not currently supported in Gram. These include the following:

- few researchers have access to grid-enabled resources. From the example given in Section V, there are small scale clusters that scientists need to integrate to run their workflows. Each of these systems have different resource manager implementations implemented and the system administrators are reluctant to install complex systems on these clusters.
- missing job statuses has been a known problem in the LEAD[14] system and more broadly. Missing statuses requires users to re-submit the jobs, increasing the total execution time and also reducing overall workflow throughput. Sigiri provides robustness in obtaining job statuses by providing multiple ways of getting information of submitted jobs and this is expected to minimize this important problem.

CREAM [17] 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's support for popular job specifications like Job Submission Description Language (JSDL)[2] and The Globus Resource Specification Language (RSL)[3]. Furthermore Sigiri directly interacts with resource managers so assumes no grid or meta scheduling middleware.

VII. Future Directions

Sigiri will continue to evolve while preserving the decoupled, lightweight, reliable, scalable job management system inter-operating with existing popular job specifications. In our experience, system administrators play a key role in the interaction of scientific codes with large-scale systems, so providing a service that eases the administration will continue to be a top level requirement. This paper introduces Sigri and its design features and early success. In addition to a comprehensive evaluation of Sigri's major design claims, we are investigating additional directions for development, several of which are identified by users who have had early experience with Sigri:

1) Dash Board: : Long running scientific workflows mandate the need of ability to monitor individual task job progress and also outputs. We plan to implement a Web dashboard to monitor the progress and status of jobs and also aggregate reports of jobs and and their states.

2) Failure Recovery: : Failures are inevitable in any distributed system. Sigiri persists all the statuses and currently can recover from various component failures. Furthermore we intend to add automatic anomaly detecting rules to monitor events, emails or human triggers and recover from any of the underlying resource managers or hardware. Also, dynamic workflows [13] need intensive human or automated system intervention which Sigiri will facilitate.

3) Quality of Service: : Sigiri currently provides support of specifying additional quality of service parameters like anticipated job queue time, maximum wall time, which are used to monitor jobs more closely and when deviations occur notify clients. This additional granular monitoring will be further enhanced and improved to ensure no jobs are left orphan due to any failures.

4) Parametric Workflow Support: : Parametric workflows are constructed by a function of high throughput input generation, by systematically varying a set of input values or executing with a set of input files. These workflows result in a large number of jobs per workflow and submitting and managing these jobs individually will add additional overhead on workflow systems. Specifications like JSDL allow parameter sweep extensions to specify these multiple jobs and bundle into one request. Sigiri will support these parametric extensions. We are further evaluating the need of glide-in support where a wider, bigger job will be submitted to the underlying resource manager and Sigiri can start up a personal resource manager and manage the acquired resources while executing multiple smaller jobs within it.

5) Security: : For resources which already support grid middleware, we plan to leverage the Grid Security Infrastructure and support authentication and authorization of grid certificates and proxies.

6) Cloud Computing Support: : With the emerging cloud computing, inter-operating with grid, stand alone private clusters and cloud provisioned resources is challenging for any large-scale escience environment requiring to submit jobs. We are evaluating Sigiri along with few other components to provide the needed abstraction for these diverse interfaces to ease the burden on workflow systems.

7) Integrating Other Job Management Interfaces: : We will continue to integrate and support other job management systems and will incorporate those on requirement basis.

Acknowledgments

The authors wish like to extend their deep appreciation and thanks to all the members of Extreme Lab, Mike Lowe, Jenett Tillotson of Indiana University and Victor Hu, IBM for their intellectual discussions and support.

References

[1] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International Jounral of Supercomputer Applications*, vol. 15, no. 3, 2001. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.24.9069

[2] A. Anjomshoaa, F. Brisard, R. L. Cook, D. K. Fellows, A. Ly, S. McGough, and D. Pulsipher, *Job Submission Description Language (JSDL) Specification v0.3*, Global Grid Forum, 2004.

[3] "The globus resource specification language (rsl), specification," http://www.globus.org/toolkit/docs/3.2/gram/ws/developer/mjs_ rsl_ schema.html.

[4] "Amazon elastic computing cloud," http://aws.amazon.com/ec2/.

[5] C. Catlett, "The philosophy of TeraGrid: building an open, extensible, distributed TeraScale facility," *Cluster Computing and the Grid 2nd IEEE/ACM International Symposium CCGRID2002*, pp. 5–5, 2002.

[6] O. E. Board, "The open science grid," *Journal of Physics: Conference Series 78:012057*, 2007.

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

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

[9] A. Yoo, M. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," *Lecture Notes in Computer Science*, pp. 44–60, 2003.

[10] W. G. S. Microsystems), "Sun grid engine: Towards creating a compute power grid," in *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2001, p. 35.

[11] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," *Network And Parallel Computing: IFIP International Conference, NPC 2005, Beijing, China, November 30-December 3, 2005: Proceedings*, 2005.

[12] P. T. Eugster, P. A. Felber, R. Guerraoui, and A. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys (CSUR)*, vol. 3, pp. 114–131, 2003.

[13] M. Caeiro-Rodriguez, T. Priol, and Z. Németh, "Dynamicity in scientific workflows," Institute on Grid Information, Resource and Workflow Monitoring Services , CoreGRID - Network of Excellence, Tech. Rep. TR-0162, August 2008. [Online]. Available: http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0162.pdf

[14] K. Droegemeier, V. Chandrasekar, R. Clark, D. Gannon, S. Graves, E. Joseph, M. Ramamurthy, R. Wilhelmson, K. Brewster, B. Domenico *et al.*, "Linked environments for atmospheric discovery (LEAD): A cyberinfrastructure for mesoscale meteorology research and education," *20th Conf. on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology*, 2004.

[15] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, 2002.

[16] E. Huedo, R. Montero, and I. Llorente, "The GridWay framework for adaptive scheduling and execution on Grids," *Scalable Computing: Practice and Experience*, vol. 6, no. 3, pp. 1–8, 2005.

[17] P. A. et al, "Cream: A simple, grid-accessible, job management system for local computational resources," in *XV International Conference on Computing in High Energy and Nuclear Physics*, 2006.