Performance Evaluation of the GridSAM Job Submission and Monitoring System

William Lee, A. Stephen McGough, and John Darlington

London e-Science Centre, Imperial College London, South Kensington Campus, London SW7 2AZ, UK
Email: lesc-staff@doc.ic.ac.uk

Abstract. Existing Distributed Resource Managers (DRMs) lack support for a standard submission language and interoperable interface for describing and launching jobs. This paper presents a standards-based job submission system, GridSAM[1], which utilises Web Services and the Job Submission Description Language (JSDL). GridSAM provides a transparent and efficient bridge between users and existing DRM systems, such as *Condor*. We demonstrate, through a set of performance results, the small overhead imposed by the GridSAM submission pipeline while improving the overall system throughput and availability. The performance results are gathered from a variety of deployment configurations to exercise the wide-ranging support of GridSAM in terms of launching mechanism, clustering set up and persistence choice.

1 Introduction

There are many Distributed Resource Management (DRM) systems in existence for launching jobs efficiently onto computational resources. However, most systems adopt proprietary languages and interfaces to describe and interact with the job launching process. This leads to the requirement that a user needs to learn a large number of job description languages and deployment mechanisms to exploit a wide variety of Grid resources.

Web Services have been recognised as the preferred technology to build distributed services in the Grid context. It provides an interoperable approach to message-oriented machine-to-machine interaction. Commercial adoption of Web Services has catalysed the development of industrial-strength platforms for deploying high-performance Web Services. Moreover, the Grid community has gathered pace in recent years to define standards for core functionality of Grid Computing. This is required to achieve interoperability across organisational and technical boundaries in order to attain the vision of a global computational network. In particular, the Job Submission Description Language (JSDL) [2] currently under standardisation and the proposed Basic Execution Service (BES)[4] interface in the Global Grid Forum are essential standards to promote interoperability among DRMs.

In this paper we present our work on the GridSAM job submission and monitoring system, undertaken as part of the Open Middleware Infrastructure Institute Managed Programme[5], as one of the first systems adopting JSDL and Web Services for job description and interaction. GridSAM seeks to provide an efficient bridge between users who wish to submit jobs transparently onto the Grid supported by job launching mechanisms such as *Forking*, *Condor*, *Globus* and *Secure Shell*. The GridSAM *Job Management Library* encapsulates the mapping from JSDL to the submission actions required by the underlying DRM. The encapsulation allows the functions to be used through the GridSAM Web Service as well as utilising it as an independent library.

This paper is organised as follows: In Section 2, we introduce the architectural components of the GridSAM system and the adoption of the staged event-driven architecture (SEDA)[3] as the implementation principle. In Section 3, we define the metrics used for performance evaluation, present our experimental setup and discuss the role of each experiment. In Section 4, we present the collected results and a comparative analysis of the performance of various GridSAM deployment set-ups.

[1] Downloadable from http://www.lesc.ic.ac.uk/gridsam

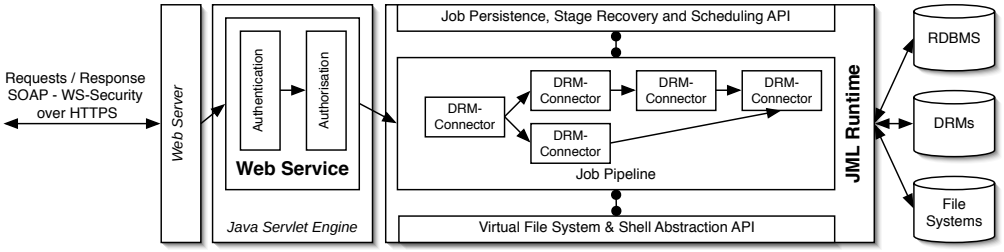

Fig. 1. GridSAM System Architecture

2 System Architecture

The objective of GridSAM is to let users execute applications through existing distributed resource managers transparently. Transparency is achieved through the use of a common job description language, JSDL, and a uniform networked access interface, Web Services. The core function of GridSAM is to translate the submission instructions specified in a JSDL document to a set of resource specific actions to stage, launch and monitor a job. This function is encapsulated in the GridSAM *Job Management Library (JML)*.

The role of the *JML* is to orchestrate the execution of a set of *DRMConnector*s - reusable component encapsulating job management actions. These components are composed by the deployer into *a network of stages* resembling a job launching pipeline. The *JML* runtime alleviates system engineers from programming common tasks, such as persistence, failure recovery and concurrency management by exposing these through the *JobManager* API.

The GridSAM Web Service makes available the *JML* through a Web Service interface. It is implemented as a Java JAX-RPC-compliant Web Service deployable in any Java Servlet compliant container. This opens up the choice of deployment platforms depending on the scalability requirements. The Web Service interface makes use of HTTPS transport security and the OMII WS-Security framework to protect message exchange as well as authenticating and authorising users. The Web Service interface demonstrates the use of the *JML* as a networked multi-user service. It is envisaged that the *JML* can be embedded in other frameworks (e.g. portal, Grid applications) offering different modes of interaction.

2.1 Submission pipeline as a network of stages

The GridSAM pipeline is constructed as a *network of stages* connected by event queues. This design is inspired by the *staged event-driven architecture* (SEDA) [3]. Instead of treating each job submission request as a single submission action, it is decomposed into robust stages that may be individually conditioned to load by thread-holding or filtering its event queue. A number of exemplar systems (e.g. *Haboob* web server) have demonstrated the use of this principle to deliver robustness over huge variations in load.

Figure 2 depicts a pipeline that launches jobs onto a *Condor* pool. Each stage in the pipeline is an implementation of the *DRMConnector* event handler interface. *DRMConnector* instances encapsulate a specific functionality that is triggered by an incoming event (e.g. stage-in event) associated with a job. Once the *DRMConnector* has completed its operation, it may enqueue events onto another stage. It effectively passes control to the next stage in the pipeline asynchronously. Long-running stages that perform blocking operations (e.g. reading files) can potentially be broken down further into sub-stages by using non-blocking I/O libraries.

The explicit control boundary, introduced by the queue between stages, improves overall parallelism in the system. The simple message-oriented event-based interface allows system engineers to focus on the DRM-specific logic, rather than the

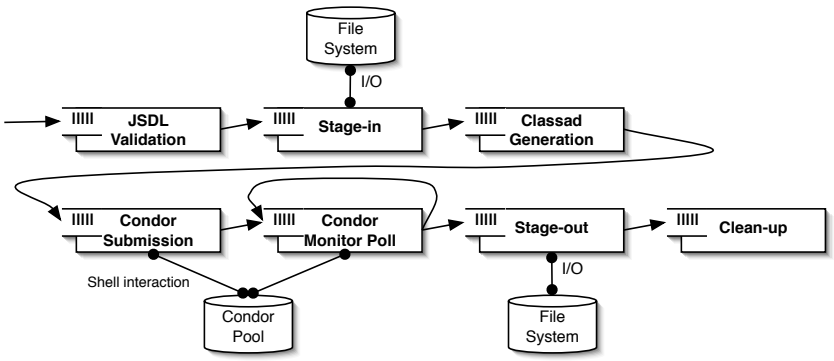

Fig. 2. Submission pipeline for Condor job in GridSAM

details of concurrency and resource management. Moreover, the event-based interface echoes the *Command* design pattern[1] that encourages component reuse and action encapsulation. For example, the *Forking* and *Secure Shell* pipeline share most of the pipeline components apart from the launching stage. Representation of state is completely encapsulated in the incoming event message, the *DRMConnector* can be distributed easily across a cluster without complex state management.

2.2 Fault recovery

The adoption of an event-based architecture allows individual stages to be restarted upon failure by persisting the event queues and the information associated with each job instance. The GridSAM *JML* provides the *JobInstanceStore* API for persisting per-job information that needs to be carried between stages and inspected for monitoring purpose; GridSAM uses the *Hibernate* [6] toolkit to provide transactional object-to-relational mapping. *DRMConnector* implementations are agnostic to the underlying persistence mechanism (e.g. in-memory replication, RDBMS persistence). *JobInstance* objects are stored in JDBC compliant RDBMS databases along with the event queues by default.

When the *JML* is initialised the event queues and the scheduler are reinstated. A previously failed job pipeline will be restarted from the beginning of the failed stage instead of the beginning of the pipeline. A stage can perform the necessary recovery action by undoing the failed resource-specific actions or perform the idempotent operation again.

2.3 Concurrency management

Each stage in the pipeline is served by a pool of threads that consume events from the stage queue and invoke the stage-specific *DRMConnector*. GridSAM builds on the *Quartz* framework [7] to schedule stages and allocate threads. Welsh et al [3] described in the original SEDA proposal the role of an application controller to dynamically self-tune resource management parameters based on run-time demands and performance targets. For example, the number of threads allocated to a stage can be determined automatically without *a priori* knowledge of job arrival rate and perceived concurrency demands. Although *Quartz* currently lacks the dynamic load adaptation support, it provides advanced date-based scheduling, fault recovery and clustering support that is unavailable in other frameworks. This can be accommodated in the future with the extensible *JML* framework using the *Java Management Extension* as an instrumentation tool.

3 Experimental Setup

Several experiments have been devised to evaluate the performance of the GridSAM Web Service. The experiments exercise several pipeline configurations, which target the following job launching mechanisms:

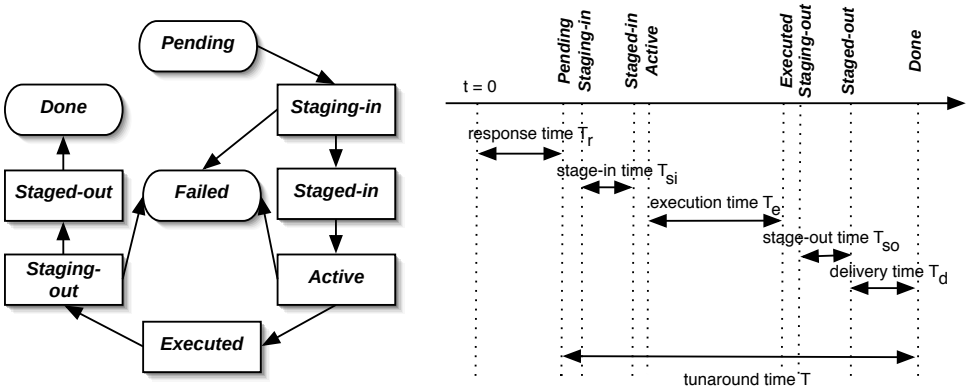

Fig. 3. Job state transition and timing parameters

– Condor
– Secure Shell
– Local Forking

The goal of the experiments is not to analyse the performance of the underlying launching mechanisms because of their incomparable nature, but the overhead incurred by using GridSAM to orchestrate the staging, launching and monitoring processes under varying load condition.

3.1 Metrics

The following performance measures were investigated in our study:

1. **Response time** is the time taken for the job request to be registered with the GridSAM service. This metric measures the performance of the Web Service security and network protocols in use. It also measures the waiting time of a user posting a job request, which is important for an acceptable user experience.
2. **Throughput** is defined as the number of jobs completed in a unit of time. Since this number depends on how many jobs are taken into account, we consider *throughput* to be a function of the number of jobs, k, and define it as k divided by the amount of time necessary to complete k jobs. The *total throughput* is therefore the special case of throughput where k equals to the total number of jobs submitted during the experiment. This definition is taken from El-Ghazawi et. al. to systematically evaluate throughput of job submission systems [8].
3. **Average turn-around time** is the time from a job being accepted by the GridSAM Web Service till completion (i.e. the job has reached the *done* state), averaged over all jobs submitted in an experiment.

3.2 Testbed

The testbed consists of 3 PCs running SUSE Linux (2.8GHz hyper-threaded, 1GB RAM). One of the PCs serves as the submission host, while the others are running the GridSAM Web Service. The testbed also includes a *Condor* pool with 347 PCs. All the machines in the testbed are connected to a 100Mbit/s network. At the start of each experiment involving the *Condor* pool, there are on average 250 unclaimed machines available in the pool.

3.3 Experiments

Each experiment simulates M users concurrently submitting N jobs each. For each simulated user, a benchmark application (**Dhrystone 2**) is submitted one at a time in pseudo-random time intervals. Each experiment was repeated for different job launching mechanisms and different average

Number	Benchmark Job	Average CPU time per Job	Average Time Intervals Between Submissions	No. of Concurrent Users	No. of Jobs per User	Comments
1	Dhrystone 2	9.99s	0s, 10s, 20s	5, 15, 30	15	Single host deployment
2	Dhrystone 2	9.99s	0s, 10s, 20s	5, 15, 30	15	Cluster deployment on 2 hosts
3	Dhrystone 2	9.99s	0s	30	15	Server killed and restarted on the 100th job

Table 1. Parameters of the experiments

job submission rates (see Table 1). The same experiment is repeated at least twice using the same parameters to reduce the effects of random events. In the stress-test case (i.e. zero interval between submissions), the simulated users are allowed to submit jobs consecutively without waiting. Experiment 1 is also repeated with several different secured transport configurations in order to evaluate the response time:

- **HTTP, WS-Security** - Signed messages on unencrypted channel
- **HTTPS, WS-Security** - Signed messages on encrypted channel
- **HTTPS, Mutual Authentication** - Unsigned messages on mutually authenticated and encrypted channel

Experiment 2 investigates the effect of clustering two GridSAM hosts on the job turnaround time. GridSAM exploits the clustering feature in Quartz to allow stages to be executed across virtual machines.

The last experiment aims to quantify fault recovery of GridSAM after a system failure. After the 100th job has been submitted to the GridSAM service, the service process is terminated abruptly with a KILL signal. The service is then restored and the status of the submitted jobs are recorded once they have reached the terminal state.

3.4 Measurement collection

All the measurements were obtained in the same way using system utilities and analysis of server-side log files. The Network Time Protocol (NTP) was used to synchronize clocks on all the hosts involved in the experiments. It provides accurate synchronisation in the range of milliseconds, therefore timestamps in log files reflect accurate timings of when events took place across the testbed.

In order to simulate concurrent load, the *Apache JMeter* utility was used. This utility allows captured SOAP messages to be replayed and response time recorded. It supports advance features for functional and performance validation. Job events are recorded in the server-side log files that were later post-processed using a statistical analysis package.

4 Experimental Results

In this paper, we will focus on two performance metrics, namely *average response time* and *job turnaround time* under various load conditions. The *average response time* denotes the time a user would have to wait for a job to be accepted by the GridSAM service. In Experiment 1 (see Figure 4[a]), the *average response rate* is well under 1 second under light load (i.e. less than 15 jobs/minute). The Use of WS-Security dominated the time taken for a job request to be handled. Less than a quarter of the time is used by the GridSAM Web Service to enqueue the request into the *JML*, the rest was spent in verifying and signing incoming and outgoing messages respectively. Although the use of HTTPS as the transport mechanism places little overhead in comparison with unencrypted HTTP, HTTPS with mutual authentication has shown dramatic decrease in response

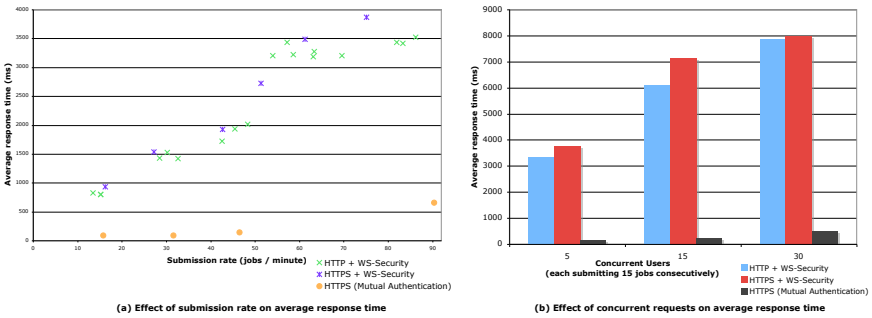

Fig. 4. Average response time for different secured transport configuration

time without lowering the security protection provided by the two mechanisms. From this analysis, HTTPS with mutual authentication is recommended if scalability is a concern. However if the message is intended to be routed to other SOAP consumers who need to validate the origin of the message, HTTPS with WS-Security would be the better choice.

The experiments have also shown that the response time scales linearly against the submission rate irrespective of the underlying job launching mechanisms. This is likely to be attributed to the *SEDA* architecture in the *JML*. The Web Service front-end simply enqueues the request into the persistence store without waiting for any of the lengthy job stages to start. It is also expected that the web server will queue incoming messages and eventually drop TCP/IP requests once the network is overwhelmed with requests. The *Apache Tomcat* web service was configured with 150 threads to handle incoming HTTP(S) requests. During the experiment especially in the case of HTTPS with mutual authentication under stressed load, the server achieved submission rate of 2236 jobs per minute with all requests accepted successfully. Figure 4[b] depicts the average response time observed during the stress test.

GridSAM currently supports *Forking*, *Condor*, *Secure Shell* and *Globus 2.4.3* as job launching mechanisms. In this paper, only the first three have been examined. The aim of this analysis is not to compare their performance in optimising usage of local resources, but the overhead induced by adopting GridSAM as the transparent job management system.

Figure 4 shows that the *average turnaround time* is relatively constant when the submission rate is less than 50 jobs a minute. The *Forking* pipeline takes on average 14.24 seconds to complete the job. Since the benchmark application takes on average 9.99 seconds to complete on the testbed, GridSAM incurs around 4 seconds of overhead to the short-running job. The overhead includes working directory preparation, potential file staging, JSDL translation and persistence for recovery. Since the job pipeline only spends time managing the launched application the overhead is unaffected by the total run-time of the executable.

In addition, Figure 4 demonstrates substantial increase in *average turnaround time* in the cases of *Forking* and *Secure Shell* when the submission rate exceeds 50 jobs/minute. This is partly due to the thread-pool settings in the *JML* defaulting to 50 threads shared among all stages. The thread-pool becomes saturated as the submission rate increases. This is particularly severe in *Forking* and *Secure Shell* because the launching stage is synchronously monitoring the forked application, therefore the threads are mostly occupied when more than 50 benchmark jobs are executing simultaneously.

In the case of *Condor*, although the *average turnaround time* is around two times higher than the other mechanisms, the saturation point is much higher and the rate of increase is less severe. Since the job is scheduled and forked by the *Condor* system, GridSAM is only serving a

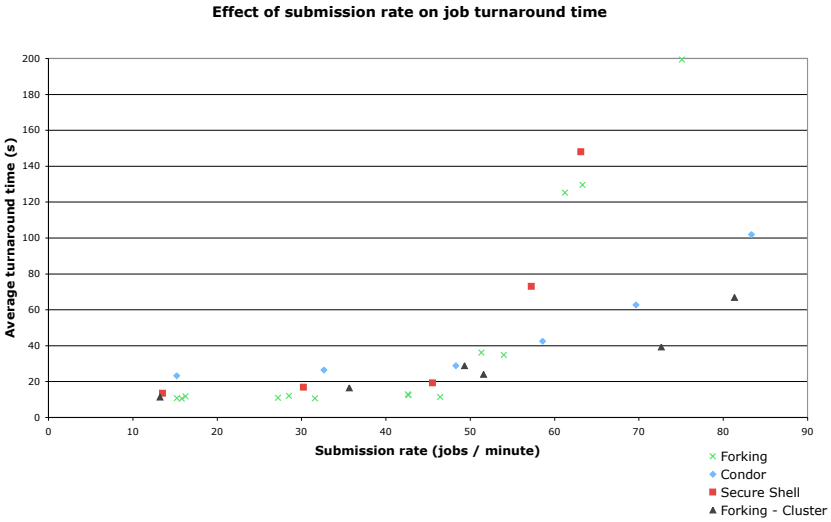

Fig. 5. Average turnaround time for different job launching mechanisms

management role by translating a JSDL document into a Condor *Classad*, then placing and monitoring it. These are short operations interacting with the *Condor* system tools. Job status is monitored through asynchronous notification from the *Condor* logging sub-system. Furthermore, since the jobs are executed remotely to the GridSAM host, the system load and the cost of context-switching is much lower than the previously described mechanisms.

In Experiment 2, the experimental clustering feature in GridSAM was enabled to allow pipeline stages to load-balance across two hosts. This feature has altered various default settings in the *JML*, in particular a networked *MySQL* database is used instead of the embedded *Hypersonic* database as well as a *Network File System* that was used for job data spooling. In Figure 4, the effect of enabling clustering for the *Forking* job launching mode is shown. It incurs on average 1 second extra overhead compared to the single-host *Forking* mode. This can be attributed to the cost in the network communication to the RDBMS and the distributed synchronisation mechanism (i.e. database row locking) used by *Quartz* to schedule requests across hosts. Nonetheless, the clustering set-up can cater for a much higher submission rate with a smoother degradation of *turnaround time*.

In Experiment 1, the number of failed jobs under normal circumstances have been recorded in Table 2. GridSAM successfully accepts all job requests although a small number fail to be launched especially in the case of *Secure Shell*. All the job failures in *Forking* and *Secure Shell* are caused by the over-consumption of system handles and remote SSH servers refusing new connections. During the experiments, failures occurred in the clusters at some saturation points, subsequent submissions are launched gracefully when system resources are freed up by the terminated jobs.

In Experiment 3, we demonstrate the robustness of GridSAM against system failure. To simulate system failure under stressed load, the GridSAM server is abruptly killed after the hundredth job has been accepted. Results of Experiment 3 have shown that GridSAM recovers all jobs eventually after restart. This is accredited to the *Quartz* framework for robustly persisting stage queue information in the RDBMS database. It is crucial for the RDBMS database to be hosted and replicated on high availability resources. The *JDBC* database abstraction in Java has provided the flexibility in scaling the underlying database implementation and deployment without any functional alteration.

Test Run	No. of Failed Submissions	No. of Failed Jobs
Forking - 5 users - 15 jobs each	0, 0, 0	0, 0, 0
Forking - 15 users - 15 jobs each	0, 0, 0	2, 0, 0
Forking - 30 users - 15 jobs each	0, 0, 0	5, 0, 0
Secure Shell - 5 users - 15 jobs each	0, 0, 0	1, 0, 0
Secure Shell - 15 users - 15 jobs each	0, 0, 0	7, 2, 0
Secure Shell - 30 users - 15 jobs each	0, 0, 0	33, 4, 0
Condor - 5 users - 15 jobs each	0, 0, 0	0, 0, 0
Condor- 15 users - 15 jobs each	0, 0, 0	0, 0, 0
Condor - 30 users - 15 jobs each	0, 0, 0	0, 0, 0

Table 2. Job submission and launching failures at 0s, 10s and 20s submission intervals

5 Conclusions

GridSAM provides an efficient bridge between users who wish to submit jobs transparently onto the Grid and existing local DRM systems. This paper has evaluated GridSAM in terms of the *response rate* and *turnaround time* against various submission rates and deployment settings. File staging performance has not been examined in this study. In future studies, we would like to investigate the effects of altering the thread-pool size and the portfolio of benchmark jobs. In this evaluation, the GridSAM *Job Management Library* and the Web Service interface have shown to impose little overhead relative to the overall runtime of the job. As the submission rate increases, GridSAM sustains high turnaround time until a high saturation point. The clustering feature in GridSAM has demonstrated an efficient means to remedy this situation. The separation of the Web Service interface and the staged event architecture for the job submission pipeline allows GridSAM to exploit widely used techniques to improve scalability. HTTP(S) traffic can be handled by Web Farms and TCP/IP load-balancers. This feature already exists on the *Apache Tomcat* server used by GridSAM. The decomposition of the job pipeline into multiple stages improves asynchronicity and parallelism. While some of the SEDA principles have been adopted in the job submission pipeline, we envisage greater control over scheduling and dynamic adaptation of resource usage could further improve performance and robustness.

6 Acknowledgements

We would like to thank the Open Middleware Infrastructure Institute Managed Programme who have funded the "*GridSAM Simple Web Service for Job Submission and Monitoring*" project.

References

1. E. Gamma and R. Helm and R. Johnson and J. Vlissides. Design patterns: Abstraction and reuse of object-oriented design. In *Lecture Notes in Computer Science*, volume 707, pages 406–431. Springer, 1993.
2. Job Submission Description Language Working Group. https://forge.gridforum.org/projects/jsdl-wg.
3. M. Welsh and D. Culler and E. Brewer. Seda: An architecture for well-connected scalable internet services. In *Eighteenth Symposium on Operating Systems Principles (SOSP-18)*, October 2001.
4. Open Grid Services Architecture - Basic Execution Service. https://forge.gridforum.org/projects/ogsa-bes-wg.
5. OMII Project. Open Middleware Infrastructure Institute. http://www.omii.ac.uk.
6. The Hibernate Project. Hibernate. http://www.hibernate.org.
7. Quartz Enterprise Scheduler. Quartz enterprise scheduler. http://quartzscheduler.org.
8. T. El-Ghazawi and K. Gaj and N. Alexandridis and F. Vroman and N. Nguyen and J. Radzikowski and P. Samipagdi and S.A. Suboh. A performance study of job management systems. pages 1229–1246, 2004.