Essential Elements for Improved Grid Usability

Dan Gunter, Keith Jackson, David Konerding, Jason Lee, Brian Tierney
Lawrence Berkeley National Laboratory

A major obstacle to Grid usability today is that Grid applications are prone to failures and performance bottlenecks. Writing "better" Grid software cannot really solve this problem: to a certain extent, this an inherent property of a large and multi-layered system. Thus, troubleshooting is an essential Grid component. However, the very layers of middleware designed to shield Grid users from the variation in underlying systems can also hide the feedback needed to diagnose and troubleshoot poor performance or failure. This paper proposes two new Grid monitoring elements, Grid workflow identifiers and consistent component lifecycle events, that will make Grid failures easier to diagnose and troubleshoot.

1.0 Introduction

One of the central challenges of Grid computing today is that, in practice, Grid applications are prone to frequent failures and performance bottlenecks. This is an unavoidable property of a large and multi-layered system. The large numbers of components involved make failure more likely, while the very layers of middleware designed to shield Grid users from variation in underlying systems also hide the feedback needed to diagnose and troubleshoot poor performance and failures.

For example, assume a simple Grid workflow has been submitted to a resource broker, which uses a reliable file transfer service to copy several files and then runs the job. Normally, this process takes 15 minutes to complete, but two hours have passed and the job has not yet completed. Determining what, if anything, is wrong is difficult. Is the job still running or did one of the software components crash? Is the network particularly congested? Is the CPU particularly loaded? Is there a disk problem? Is a software library containing a bug installed somewhere?

Current Grid monitoring systems can answer many of these questions if only one workflow is present, but they break down when Grid resources and components are used in parallel by multiple workflows. The reason for this difficulty is that below the highest level of middleware, e.g., the resource broker in the example above, there is no consistent mechanism for identifying which monitoring data is associated with a given workflow.

To solve this problem, we propose adding a new element to core Grid monitoring requirements. Previously, we have argued [17] that Grid monitoring and troubleshooting systems should have the following elements:

- Globally synchronized clocks (e.g., with NTP [23])
- End-to-end monitoring data (hosts, networks, middleware, application)
- Archiving (e.g., relational database)
- Standard data model (e.g., timestamp, name, values)
- Dynamic control of monitoring granularity

Two new elements should be added to this list: Grid workflow identifiers (GIDs) and consistent component lifecycle log events. A GID is a globally unique "key" that can track a Grid Job across administrative domains. The GIDs would be generated at the time the workflow was created, and

then transferred between layers of middleware and across administrative domains. Component lifecycle events are monitoring events that are logged at the start and end of every component's lifecycle.

1.1 Extended Use-Case

Assume the Grid workflow shown in Figure1. The workflow, which uses multiple input files and generates multiple output files, is submitted to a resource broker. The resource broker determines the best compute and storage resources to use for the workflow at this time based on some information from Ganglia [29] host monitoring and the PingER [6] network monitoring system. Data is staged using a replica manager, which uses a Reliable File Transfer service [28], which in turn uses GridFTP [1]. A job is submitted to the Globus Gatekeeper [10], which passes it to the Globus Job Manager, which authorizes the user using Akenti, which hands the job off to Portable Batch System (PBS) [3] scheduler, which runs the job. Output data files are then send to a High Performance Storage System (HPSS) [16] installation using the Storage Resource Manager (SRM) [31] middleware.

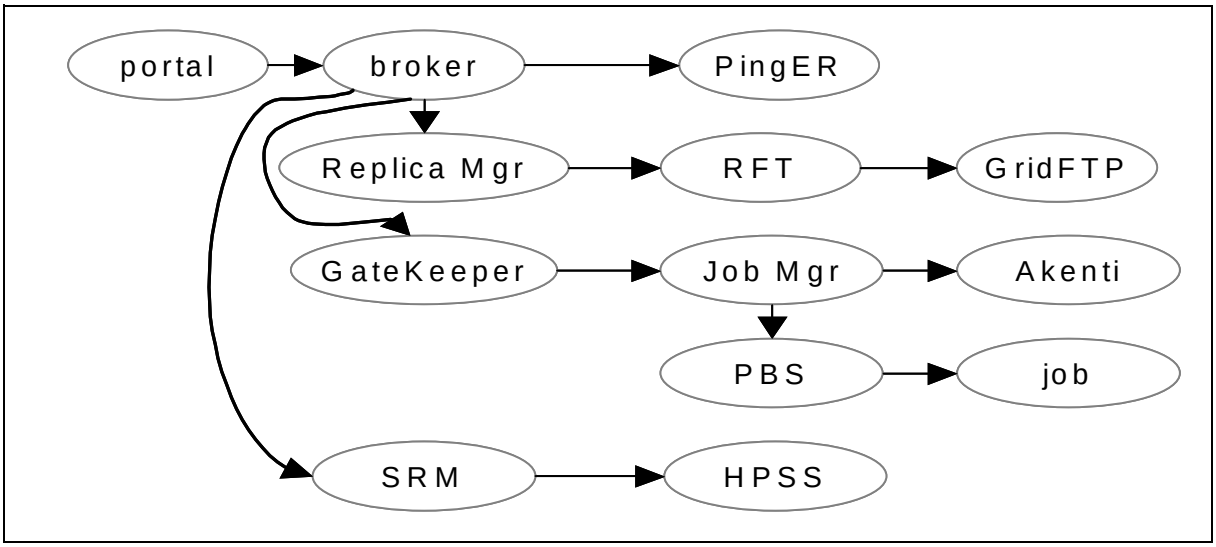

Figure 1: Example Grid Workflow

This workflow uses many of software components, any of which may potentially fail due to software, hardware, or network problems. At a minimum, the example above uses the following components: Grid portal, resource broker, Ganglia, PingER, replica manager, Reliable File Transfer service, GridFTP service, SRM service, Globus Gatekeeper, Globus job manager, Akenti, PBS, HPSS, in addition to whatever components are used by the job itself.

We will refer back to the above use-case as we discuss the importance of using GIDs.

2.0 Related work

In this section, we describe some related work in the area of Grid workflow monitoring.

2.1 Grid Workflow Engines

Although "workflow" is a familiar concept in Computer Science, implementations of workflow engines for the Grid are still in their infancy. One early example is the Condor [21] "Directed

Acyclic Graph Manager" (DAGMan), which is a meta-scheduler that sits on top of the standard Condor scheduler. The DAGMan submits jobs to Condor in an order represented by a directed acyclic graph (DAG) and processes the results. To monitor DAGs, Condor sends status and "standard error" output from a running DAG back to a user-specified log file. This monitoring information is not end-to-end: it does not include application, network, or host data, or monitoring data from jobs that are handed off to other schedulers such as the Globus JobManager. Because a DAG is itself a Condor job, it is assigned Condor "job cluster" identifier, but this identifier is not propagated outside of the Condor components. In order to get host and network status, Condor provides the Hawkeye [15] monitoring tool, but the integration of the Hawkeye sensor data with the built-in Condor job monitoring is a work in progress.

The Pegasus [8] workflow system provides an extra layer of abstraction on top of DAGMan. In Pegasus, users provide an "Abstract Workflow" that describes virtual data transformations. This is transformed into a "ConcreteWorkflow" that contains actual data locations. This concrete workflow is submitted to DAGMan for execution. The monitoring and identification of the workflow in Pegasus are essentially the same as for DAGMan.

All the preceding systems are built on pre-Web Services technologies. In the Web Services community, workflows are addressed in several specifications, including the Business Process Execution Language for Web Services (BPEL4WS) [3] specification from IBM , and the OASIS Web Services Coordination Framework specification [20]. The Web Services Resource Framework (WS-RF) [34] is a convergence of Web Services technologies with the Open Grid Services Architecture (OGSA) [11]. Many people are looking at using WS-RF for executing Grid workflows.

2.2 Grid Monitoring

There are a number of distributed monitoring projects that can provide the raw data needed for analysis of end-to-end performance. Cluster tools such as Ganglia [29] and Nagios [24] can scalably provide detailed host and network statistics. These data can be integrated into monitoring frameworks such as the European Data Grid's Relational GMA (R-GMA) [4], the Globus Monitoring and Discovery Service (MDS) [7], or Monitoring Agents using a Large Integrated Services Architecture (MonALISA) [26].

At LBNL we have developed the NetLogger Toolkit [32], which provides non-intrusive instrumentation of distributed computing components. Using NetLogger, distributed application components are modified to produce time-stamped logs of "interesting" events at all the critical points of the distributed system. NetLogger uses a standard data model, allows for dynamic control of logging granularity, and can collect monitoring data in a relational database [14].

There are many more Grid monitoring tools. In fact, the main challenge for current Grid monitoring efforts is not a lack of functionality but rather a glut of implementations: there are many competing sensors, data models, formats, and protocols. We do not discuss how to "solve" this problem here, but we do believe that adoption of a standard GID across monitoring components will augment and help drive Grid monitoring interoperability efforts.

3.0 Grid Workflow Identifiers

In this section, we discuss the representation of GIDs and the interfaces needed for Grid Services to support them.

3.1 Creating identifiers

The GID will be chosen by the originator of the Grid job. Depending on the details of the job submission process, the "originator" could be a Web portal, Grid meta-scheduler, or other client

program. Because they are chosen independently in a distributed system, a primary requirement for GIDs is that they be globally and temporally unique. They must be globally unique so that no extra information is necessary to differentiate workflows running anywhere on the global Grid: even if the user that started the job has other information available, other users or site administrators, in general, will not. They must be unique for all time (or at least all times after GIDs come into use) so that workflows can be also be differentiated, for example, in archived monitoring data.

Procedures for creating unique identifiers are already well-documented under the moniker of a Universal Unique Identifier (UUID) [30]. There are actually several similar specifications for UUIDs, including ISO 11758 [18] and the OSF Distributed Computing Environment [9]. In all these specifications, UUIDs combine a name from a centrally administered namespace tied to the host (the network hardware address, DNS, etc.) and a high-resolution timestamp. On most operating systems, including Linux, Mac OS-X, and Windows, using the OSF DCE universal identifiers is as simple as running the program *uuidgen* or making API calls to the library *uuidlib.* The particular form of the identifier is not important, as long as it is reasonably small (e.g., less than 128 bytes) and is encoded in a readable format that is easily added to URI's, command-lines, configuration files, etc. For example, invoking *uuidgen* on a Linux system returns an acceptable encoding:

```
% uuidgen
c2717bd2-0b27-49a8-813a-ef196b9d5a5a
```

In summary, we propose that GIDs should be generated independently by the workflow originator, using the UUID standard.

3.2 Integration with Grid Services

In the use-case presented in Section 2, it is clear that the workflow encompasses many different Grid components. The graph of the workflow (Figur e1) also represents how GIDs must be propagated. Each edge in this graph also represents two transfers of the GID: one between Grid components, and a second transfer from the component to its logging facility. This presents a problem: What mechanisms should be used to transfer GIDs?

There are two types of approaches to take here. The first approach is to modify, component-by-component, the existing interfaces to support transfer of the GID. This is easier in the short term, and is the approach we took in example described below in Sect i on5.0

The second approach is to integrate GIDs in the ground floor of a general-purpose Grid framework such as the Open Grid Services Architecture (OGSA) [11] on which the Web Services Resource Framework (WS-RF) [34] is based. We think that this is by far the better solution in the long term, as it allows components to change their service definitions independently of the GID interface. Of course, it requires a level of agreement on how to leverage the standard services provided by the framework to provide GID propagation interfaces. For example, all Grid services could support a standard WSDL [5] portType for GIDs, or the WS-RF framework could support carrying the GID in a SOAP [12] header field. The choice of implementation technology is less important than the agreement to use some standard technology that spans Grid components.

3.3 Security considerations

By design, GIDs reveal information about a workflow. There is no doubt that this information could, under some circumstances, be a security threat. However, the control and data information transferred between the Grid components is also a potential security threat -- probably a far larger one. Therefore, need for additional security measures just for GIDs seems unlikely; in general, whatever measures already being taken to secure the control and/or data flows should also be

adequate for securing GIDs.

4.0 Standard Component Lifecycle Events

In order to trace the progress of a distributed job, it is very important not to have any “holes” in the monitoring data. Unlike debugging a process on a single host, it is often difficult to pause and restart a Grid job, or to query it for information that it is not already pre-configured to log. Therefore, it is important that all Grid components are configured to perform at least a minimal level of logging. Specifically, every Grid component should log at least one monitoring event when it starts and one just before it ends, whether it failed or succeeded. This methodology, in coordination with GIDs, allows troubleshooting systems to see which components are associated with the workflow, and to determine which of those components completed successfully, failed, or are still executing (and for how long).

At a minimum, each monitoring event, including the “start” and “stop” events, should have a timestamp, name of the event, source and possibly destination host, and a GID. For example, the Globus GateKeeper could log these monitoring events:

```
DATE      : 2004-02-02T22:21:35.753024
EVENT_NAME: gateKeeper.start
HOST      : 131.243.2.22
GID       : b21746a9-6cb8-4256-93e1-1a3106f8d575

<... GateKeeper runs ...>

DATE      : 2004-02-02T22:21:35.839715
EVENT_NAME: gateKeeper.end
HOST      : 131.243.2.22
GID       : b21746a9-6cb8-4256-93e1-1a3106f8d575
```

Although Grid performance analysis benefits from much more detailed logging, the component lifecycle events alone provide an overview of the relative time spent in various Grid components. This is illustrated in the graphs of example results shown in the next section.

5.0 Sample Results

At the IEEE Supercomputing Conference in November 2003 (SC2003), we demonstrated tracing a workflow for a distributed biochemistry computation called AMBER [27]. The following components were instrumented with NetLogger:

- pyGlobus: pyglobusrun, pyglobus-url-copy
- Globus GateKeeper
- Globus JobManager
- Akenti (access control policy library)
- Amber

Start and end events, with GIDs, were logged for each component. From these monitoring events, we could visualize the file staging, remote execution, and access-control steps of the job. The demonstration GUI used the status and error codes from the NetLogger messages to draw a "lifeline" (line connecting successive events on theY-axis vs. time on the X-axis) of events that indicate to the user the progression of the file staging and job execution components.

5.1 Methodology

For this demonstration, as could be expected, our methods of inserting the GID into the logging were ad-hoc and specific to the components we were using. Our task was eased somewhat by our use of pyGlobus [19], which provides Python language wrappers for Globus. Using the Python version of NetLogger with pyGlobus made it very easy for us to create instrumented versions of the pyglobus-url-copy and pyglobusrun programs.

First, we modified the pyglobusrun command to add a GID to the Resource Specification Language (RSL) of each job being submitted. Next we had to modify the Globus GateKeeper, which has no knowledge of GIDs, to read the GID from the RSL of the submitted job. Once the GateKeeper had the GID, we took advantage of the fact that both Akenti and the AMBER application were a child processes of the GateKeeper, and simply passed the GID to them via Unix environment variable. The GridFTP client was also built using pyGlobus, so transferring the GID to the Reliable File Transfer Service (RFT) could be done within Python. This propagation of the GID is shown in Figur e2.

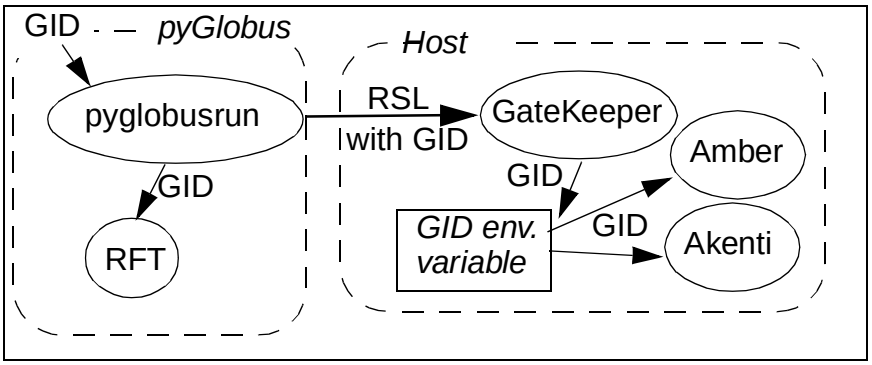

Figure 2: GID propagation in the SC2003 demo

The GID is passed to each component via a NetLogger-specific URL syntax that sets the destination for NetLogger messages. Thus, a typical RSL string looked like this (with the GID highlighted):

```
&(executable=/usr/local/bin/amber)(jobType=single)
 (environment=
   (NETLOGGER_ON "yes")
   (NETLOGGER_DEST "x-net-
log://B146.231.SC03.ORG:14830?level=2&const_GID=b21746a9-6cb
8-4256-93e1-1a3106f8d575"))
```

The NetLogger instrumentation module uses the "NETLOGGER_DEST" enviroment variable to control a number of NetLogger options. One of these options is to add a GID to every message if const_GID is found. This feature allowed us to add GIDs to all NetLogger-instrumented components without changing the instrumentation itself or re-compiling the component. For more details on NetLogger URLs, see [25].

5.2 Analysis

To illustrate the usefulness of GIDs in correlating the results, we show in Figure 3 two versions of the same monitoring data. In these graphs, there are six separate AMBER jobs, of varying length.

Each job stages its data, submits to the Globus GateKeeper, gets authorized by Akenti, runs the AMBER application, then returns the results.

In the first graph, the GID is ignored and only time correlation is performed. In the second, the GID is used to connect successive events into a lifeline. The improvement is obvious, but note particularly how difficult it would be to correlate the monitoring events for the entire life of the first job, or during the overlaps of the third and fourth jobs between 75 to 100 seconds (circled in the figure).

The graph also shows the importance of consistent component lifecycle events. During this set of runs, two of the jobs were killed prematurely. This shows up on the graph as a failure to transfer the (non-existent) result data, i.e. missing events in the "transfer results" section of the lifeline. Normally, failures of this sort are clear from the job's return value, but in this case a false value of "success" was being returned. Despite this false positive, consistent "start" and "end" events for every step of the job still alert the user to an error in these application runs.

Not discussed in this paper is NetLogger's activation service, which provides the ability to activate more detailed instrumentation and debugging information in a running process. This allows one to "drill down" to find the source of problems or bottlenecks. This is desribed in more detail in [13].

5.3 Job-Independent Monitoring Data

Clearly there are some types of monitoring data that are not associated with a particular job, and for which the concept of GID is meaningless. For example, host monitoring data (CPU, disk, memory, etc.) or network monitoring data may not be associated with a specific Grid job. Sometimes the impact of other processes on the system is crucial to understanding your job's performance. To correlate this type of data with job performance, we must use time stamps. This is why it is important have synchronized clocks on all Grid hosts, and to collect monitoring data using precision time stamps. A graph showing this type of analysis is shown in Figur e4. In this graph, it is clear that increased CPU utilization, shown at the bottom of the graph, correlates with decreased performance for the AMBER job.

While time correlation is sometimes the only means of associating workflows with other monitoring data, it is good to remember that time correlation alone cannot distinguish which of several parallel activities is associated with the change in resource state. Because, as discussed above, GIDs add this valuable ability, we should add them to the monitoring data, and use them in analyses, wherever possible.

6.0 Open Issues

A GID could be designed to provide some additional information about the workflow itself, for example to indicate the parent-child relationship among workflow nodes or contain metadata about the originator of the workflow. We are wary of this approach because any metadata embedded in the GID may complicate its representation or interpretation. Instead, we believe that this functionality should be layered on top of the GID, e.g. by using the GID as a key to locate and update the metadata in a central repository. As we gain more experience with Grid workflows, this intuition may prove to be wrong; either way, though, we know of no compelling reason to justify complicating GIDs, at this early stage, with workflow-specific metadata.

7.0 Conclusion

Troubleshooting a workflow in current Grid environments is difficult. By default, many components produce no monitoring data and, even when they do, the monitoring data is difficult to correlate with the data from other components in the same workflow. We believe that an important first step to solving this problem is to build into the Grid infrastructure standard interfaces to Grid workflow identifiers (GIDs), and to add to every Grid component standardized lifecycle monitoring events. In this paper, we have

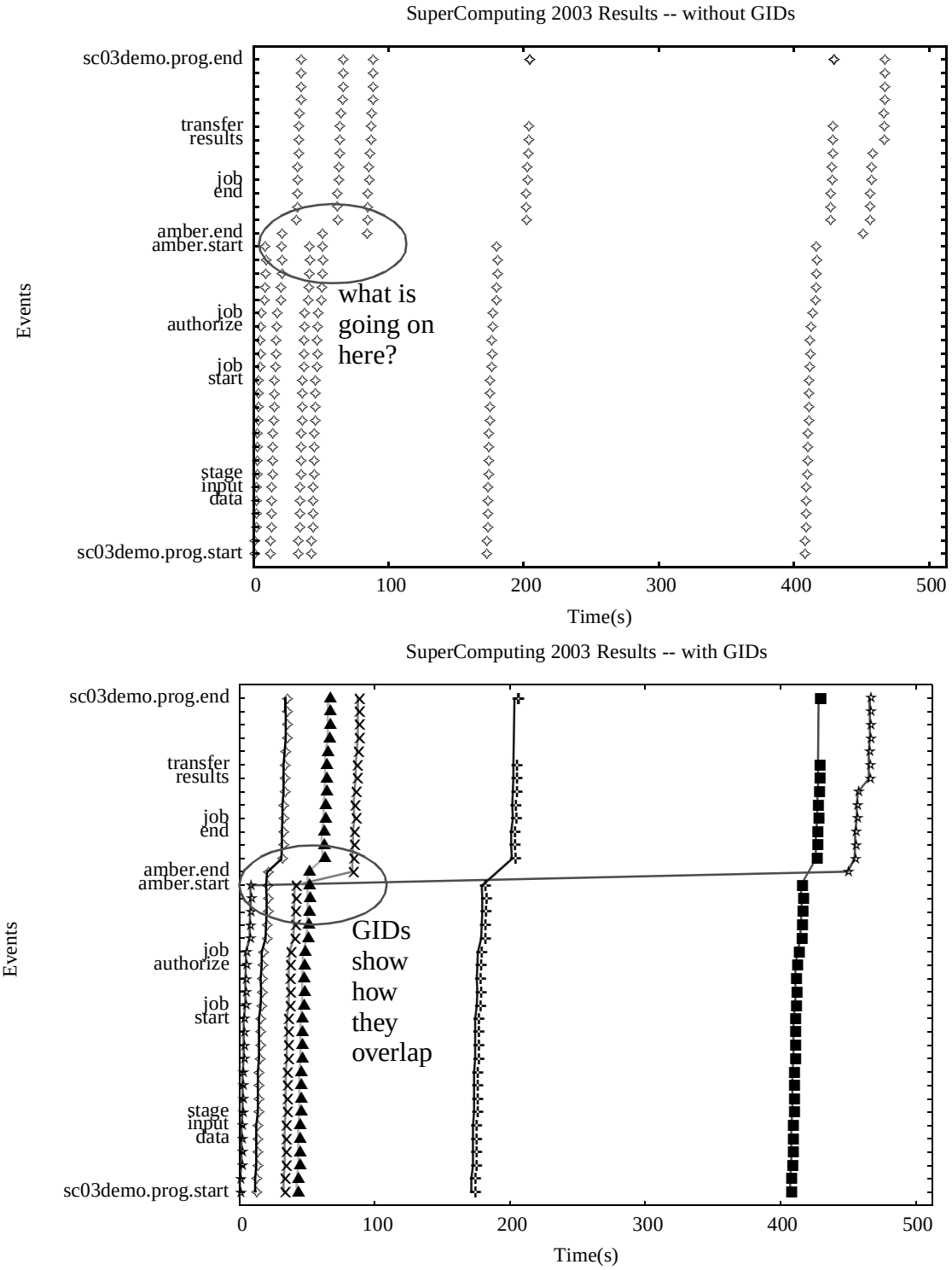

Figure 3: Amber Workflow (SC2003 demonstration)

outlined the technologies needed to create GIDs and integrate them with Grid monitoring services. We have also described the requirements for lifecycle monitoring events. Using results from our Supercomputing demonstration, we have shown how these two elements, working

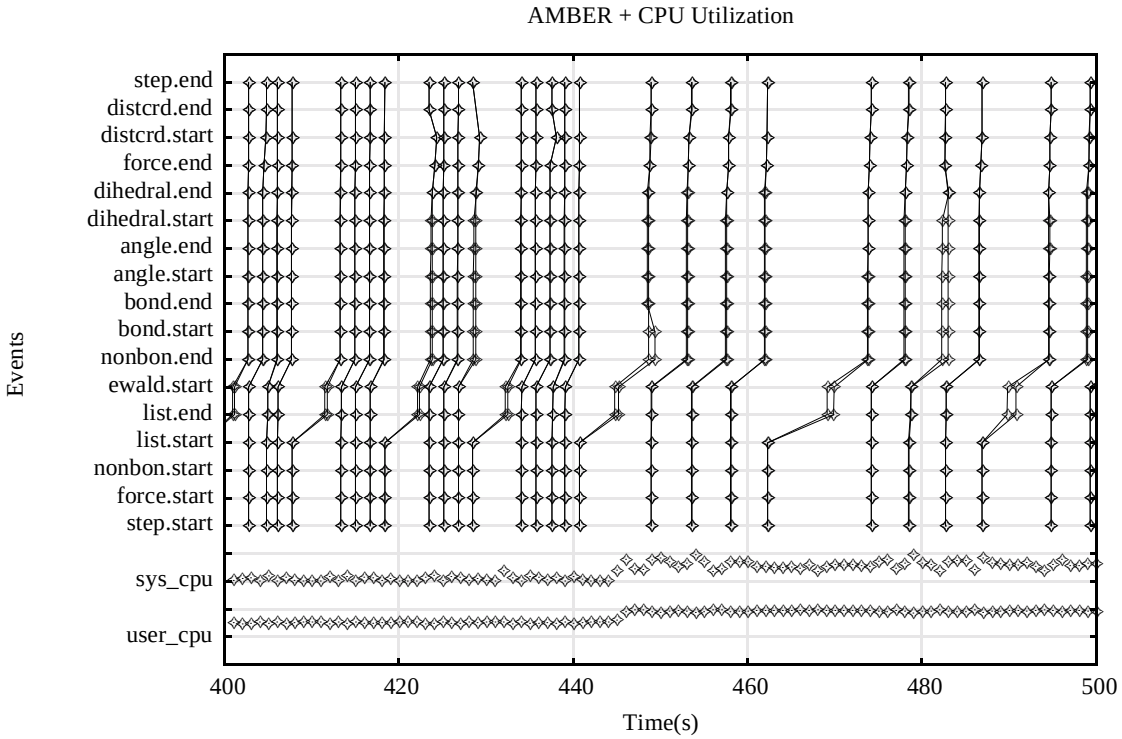

Figure 4: Amber Instrumentation with CPU monitoring

together, can form the basis for understanding the behavior of a Grid workflow.

8.0 References

[1] W. Allcock, Bester, J., Bresnahan, J., Chervenak, A., Foster, I., et.al. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing.IEEE Mass Storage Conference, 2001

[2] BPEL4WS. Business Process Execution Language for Web Services. http://www.ibm.com/developerworks/library/ws-bpel/

[3] Bayucan, A. et al.. Portable Batch System: External Reference Specification. Technical Report, MRJ Technology Solutions, November 1999

[4] R. Byrom et. al., R-GMA: A Relational Grid Information and Monitoring System, Proceedings of the Cracow '02 Grid Workshop, January 2003. Web: http://edms.cern.ch/file/368364/1/rgma.pdf

[5] R. Chinnichi, M. Gudgin, J. Moreau, J. Schlimmer, S. Weerawarana, eds. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. http://www.w3.org/TR/wsdl20/

[6] R. Les Cottrell and Connie Logg. PingER History and Methodology. Developing Countries Access to Scientific Knowledge: Quantifying the Digital Divide, ICTP Trieste, October 2003; also SLAC-PUB-10187.

[7] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman. Grid Information Services for Distributed Resource Sharing. Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

[8] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Workflow Management in GriPhyN. Grid Resource Management, Kluwer, 2003. http://www.isi.edu/~deelman/pegasus.htm

[9] Distributed Computing Environment (DCE), http://www.opengroup.org/dce/

[10] I. Foster, I. and C. Kesselman. Globus: A Toolkit-Based Grid Architecture. In Foster, I. and Kesselman, C. eds. The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann,1999, 259-278. http://www.globus.org/

[11] I. Foster, C. Kesselman, J. Nick, S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[12] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, H. Nielsen. SOAP Version 1.2 Part 1: Messaging Framework. http://www.w3c.org/TR/2003/REC-soap12-part1-20030624/

[13] D Gunter, B. Tierney, C. E. Tull, V. Virmani. On-Demand Grid Application Tuning and Debugging with the NetLogger Activation Service. 4th International Workshop on Grid Computing (Grid2003), LBNL-52991

[14] D. Gunter, B. Tierney, K. Jackson, J. Lee, M. Stoufer. Dynamic Monitoring of High-Performance Distributed Applications. Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing, HPDC-11, July 2002.

[15] Hawkeye: A Monitoring and Management Tool for Distributed Systems. http://www.cs.wisc.edu/condor/hawkeye/

[16] HPSS: http://www4.clearlake.ibm.com/hpss/

[17] J. Hollingsworth and B. Tierney. Instrumentation and Monitoring in The Grid, Volume 2. Morgan Kaufman, 2003.

[18] ISO/IEC 11578:1996 Information technology -- Open Systems Interconnection -- Remote Procedure Call,. http://www.iso.ch/cate/d2229.html

[19] K. Jackson. pyGlobus: a Python Interface to theGlobus Toolkit. Concurrency and Computation:Practice and Experience, 14 (13-15), 2002, pp.1075-1084.

[20] M. Little, E. Newcomer, eds. Web Services Coordination Framework (WS-CF). OASIS WS-CAF TC Public Documents, July 2003. http://www.oasis-open.org/committees/download.php/4345/WSCF.pdf

[21] M. Litzkow, M. Livny, M. Mutka. Condor --- A hunter of idle workstations. In Proceedings of the Eighth Conference on Distributed Computing Systems, San Jose, California, June 1988

[22] W. Matthews, L. Cottrell. The Pinger Project: Active Internet Performance Monitoring For The HENP Community. SLAC-REPRINT-2000-008, May 2000. 7pp., Published in IEEE Commun.Mag.38:130-136,2000

[23] D. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. IETF RFC 1305. http://www.ietf.org/rfc/rfc1305.txt

[24] Nagios, http://www.nagios.org

[25] NetLogger URLs: http://dsd.lbl.gov/NetLogger/current/NetLogger_URL.html

[26] H.B. Newman, I.C. Legrand, P.Galvez, R. Voicu, C. Cirstoiu. MonALISA: A Distributed Monitoring Service Architecture. CHEP 2003, La Jolla, California, March 2003

[27] D.A. Pearlman, D.A. Case, J.W. Caldwell, W.R. Ross, T.E. Cheatham, III, S. DeBolt, D. Ferguson, G. Seibel and P. Kollman. AMBER, a computer program for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to elucidate the structures and energies of molecules. Comp. Phys. Commun. 91, pp. 1-41 (1995)

[28] Reliable File Transfer Service: http://www-unix.mcs.anl.gov/~madduri/main.html

[29] F. Sacerdoti, M. Katz, M.. Massie, D. Culler. Wide Area Cluster Monitoring with Ganglia. Proceedings of the IEEE Cluster 2003 Conference, Hong Kong. http://ganglia.sourceforge.net

[30] R. Salz, P.Leach. UUIDs and GUIDs. http://hegel.ittc.ukans.edu/topics/internet/internet-drafts/draft-l/draft-leach-uuids-guids-01.txt.

[31] A. Shoshani, A. Sim, and J. Gu. Storage resource managers: Middleware components for grid storage. In Proceedings of the Nineteenth IEEE Symposium on Mass Storage Systems, 2002.

[32] B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, D. Gunter. The NetLogger Methodology for High Performance Distributed Systems Performance Analysis. Proceeding of IEEE High Performance Distributed Computing, July 1998, LBNL-42611. http://www-didc.lbl.gov/NetLogger/

[33] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman. Grid Service Specification. Open Grid Service Infrastructure WG, Global Grid Forum, Draft 2, 7/17/2002. http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-grid-service-29_2003-04-05.pdf

[34] WS-Resource Framework: http://www-fp.globus.org/wsrf/default.asp