

Essential Grid Workflow Monitoring Elements

Submitted to GCA'05

Daniel K. Gunter, Keith R. Jackson, David E. Konerding, Jason R. Lee and Brian L. Tierney
Distributed Systems Department
Lawrence Berkeley National Laboratory
{dkgunter, krjackson, dekonering, jrlee, bltierney}@lbl.gov
1 Cyclotron Rd, MS 50B-2239, Berkeley, CA 94103, USA

Abstract

Troubleshooting Grid workflows is difficult. A typical workflow involves a large number of components – networks, middleware, hosts, etc. – that can fail. Even when monitoring data from all these components is accessible, it is hard to tell whether failures and anomalies in these components are related to a given workflow. For the Grid to be truly usable, much of this uncertainty must be eliminated. We propose two new Grid monitoring elements, Grid workflow identifiers and consistent component lifecycle events, that will make Grid troubleshooting easier, and thus make Grids more usable, by simplifying the correlation of Grid monitoring data with a particular Grid workflow.

keywords: Grid performance, Grid troubleshooting, Grid Services

1 Introduction

One of the central challenges of Grid computing today is that Grid applications are prone to frequent failures and performance bottlenecks. The real causes of failure are often hidden by intervening layers of application, middleware, and operating systems. 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. In today's Grid, it is difficult to determine what, if anything, went wrong. 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?

In the simple case where the resources and middleware are only servicing one workflow, current Grid monitoring

systems can answer these questions by correlating the workflow performance with the timestamps on the associated monitoring data. But the whole point of a Grid is that resources and middleware are shared by multiple workflows. In this case, workflows will interleave their usage of middleware, hosts and networks. At the highest level of middleware, e.g., the resource broker in the example above, there may be an identifier that can track the workflow. But once the workflow leaves that layer, there is very little beyond rough time correlation to help identify which monitoring data is associated with which workflow.

With enough monitoring data, and enough time spent in analysis, troubleshooting is still possible. For instance, network packet traces can reveal the traffic patterns relevant to a given application (assuming ports and hosts are known); multiple runs of the same workflow can allow better guesses at which performance anomalies are correlated, and which are due to other workflows; etc. But this is tedious, non-reusable, work.

Fortunately, with the addition of two simple elements, we can improve the troubleshooting process across the board. Previously, we have argued [?] that Grid monitoring and troubleshooting systems should have the following elements:

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

The two new elements we propose are Grid workflow identifiers (GIDs) and consistent component lifecycle events. A GID is a globally unique *key* that can track a Grid

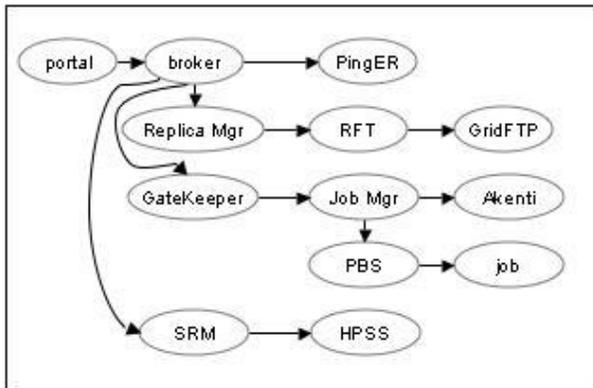


Figure 1. Example Grid Workflow.

workflow across components. 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 mark the start and end of every component’s lifecycle in a consistent and useful way; in particular they include the GID to identify the workflow to which this lifecycle belongs.

1.1 Extended Use-Case

Assume the Grid workflow shown in Figure ???. The workflow, which uses multiple input files and generates multiple output files, is submitted via a portal 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 [?] host monitoring and the PingER [?] network monitoring system. Data is staged using a replica manager, which uses a Reliable File Transfer service [?], which in turn uses GridFTP [?]. A job is submitted to the Globus Gatekeeper [?], which passes it to the Globus Job Manager, which authorizes the user using Akenti, which hands the job off to Portable Batch System (PBS) [?] scheduler, which runs the job. Output data files are then send to a High Performance Storage System (HPSS) [?] installation using the Storage Resource Manager (SRM) [?] middleware.

This workflow uses many 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: a Grid portal, resource broker, Ganglia, PingER, replica manager, Reliable File Transfer service, GridFTP service, SRM service, Globus Gatekeeper, Globus job manager, Akenti, PBS, and HPSS. The application itself may also use external components.

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

2 Related Work

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

2.1 Global Clock Synchronization with NTP

Much has been written about the theory of global clock synchronization (e.g., [?]). In distributed systems, the dominant implementation is the Network Time Protocol (NTP). Globally synchronizing clocks with the Network Time Protocol (NTP) is, by now, common practice. It is configured by default in most flavors of Linux; in fact, NTP is arguably the oldest continuously used protocol on the Internet. Although its accuracies of “low tens of milliseconds on WANs, submilliseconds on LANs” [?] are not ideal, they are often sufficient for Grid troubleshooting. And, anecdotally, our experience is that the networks used for Grid computing tend to have accuracies even better than tens of milliseconds, more often in the range of two to five milliseconds.

2.2 Grid Workflow Engines

Although *workflow* is a familiar concept in Computer Science, implementations of workflow engines for the Grid are still in the early stages. One of the more widely used engines is the Condor [?], *Directed Acyclic Graph Manager* (DAGMan), which is a meta-scheduler that interfaces with 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 Condor components. In order to get host and network status, Condor provides the Hawkeye [?] monitoring tool, but the integration of the Hawkeye sensor data with the built-in Condor job monitoring is a work in progress.

The Pegasus [?] 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 *Concrete Workflow* 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.

The preceding systems are not built on Web Services technologies. However, Web Services is becoming an important part of Grid functionality. In the Web Services community, workflows are addressed in several specifications, including the Business Process Execution Language for Web Services (BPEL4WS) [?] specification from IBM, and the OASIS Web Services Coordination Framework specification [?]. The Web Services Resource Framework (WS-RF) [?] is a convergence of Web Services technologies with the Open Grid Services Architecture (OGSA) [?]. Several groups anticipate using WS-RF for executing Grid workflows.

2.3 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 [?] and Nagios [?] can scalably provide detailed host and network statistics. This data can be integrated into monitoring frameworks such as the European Data Grid's Relational GMA (R-GMA) [?], the Globus Monitoring and Discovery Service (MDS), or Monitoring Agents using a Large Integrated Services Architecture (MonALISA) [?].

At LBNL we have developed the NetLogger Toolkit [?], 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.

There are many more Grid monitoring tools. In fact, the main challenge for current Grid monitoring efforts is interoperability between 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 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 should be chosen at the root, or originator, of the workflow, so that it can propagate to all components. The originator could be a Web portal, Grid meta-scheduler,

or other client program. A primary requirement for GIDs is that they be unique within the scope of the workflow. On the Grid, this usually means that they must be unique in space and time, particularly if the results are to be archived or shared. Although distinct identifiers in a local scope could be mapped to globally unique ones post-hoc, this mapping process is a potential source of errors that can be avoided by simply creating a globally unique identifier to begin with.

Creating a globally unique identifier is not hard. Procedures for creating a Universal Unique Identifier (UUID) [?] are specified in ISO 11758 [?] and in the documentation for the OSF Distributed Computing Environment [?]. In both specifications, UUIDs combine a name from some centrally administered namespace tied to the host (the network hardware address, DNS, etc.) with 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`.

3.2 Integration with Grid Services

Propagating GIDs from the workflow originator to all the workflow components is the most technically challenging part of using GIDs. Referring back to the Grid workflow from Section 2, each edge in the workflow graph (Figure 1) can be seen as two transfers of the GID: one between Grid components, and a second transfer from the component to its logging facility. Some mechanism must be added to existing software for this to work. We see two basic approaches to this problem. 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 to perform the experiment described in Section 5.0

The second approach is to integrate GIDs at the ground floor of a general-purpose Grid framework such as the Open Grid Services Architecture (OGSA) [?], on which the Web Services Resource Framework (WS-RF) [?] is based. We think that this solution has advantages in the long term, because it allows components to change their service definitions independently of the GID interface. Also, it does not require an ongoing programming effort by all the users of Grid software. On the other hand, this approach requires a standardization effort. The choice of implementation technology is secondary, but two possibilities are that all Grid services could support a standard WSDL [?] portType for GIDs, or the WS-RF framework could support carrying the GID in a SOAP [?] header field. As an example of the latter, an element called *GID* in an officially sanctioned namespace, here assigned the prefix *gridns* could be added to the SOAP header:

```

<SOAP-ENV:Envelope>
  <SOAP-ENV:Header>
    <gridns:GID SOAP-ENV:role=
      `http://www.ggf.org/Workflow">
        dc62ebb8-15f6-429e-9241-
          f935f96f6964
    </gridns:GID>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    (message contents)
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

The *SOAP-ENV:role* attribute indicates the namespace URI of the message receivers (either the ultimate receiver or intermediaries along the way), in SOAP parlance called actors, that should look at this header. Actors that do not know about the namespace *http://www.ggf.org/Workflow* will simply ignore the GID header. Finally, it should be noted that support for GIDs is not an all-or-nothing proposition; support in some components may allow inference of the GIDs in other components in the same workflow by mapping shared context to a GID. For example, consider a parallel GridFTP transfer. The start and end of the transfer are marked with a GID, URL being transferred, and ports set up for the transfer. Then low-level monitoring from the GridFTP transfer logs can be mapped to the same GID by matching the URL, port, and time range for the transfer start/end events against the URL and port for transferred files. Making this inference requires extra analysis code, but it is a general solution: the same code can aid the analysis of all workflows using those components.

4 Standard Component Lifecycle Events

Tracing the progress of Grid workflow is difficult if there are *holes* in the monitoring logs where the workflow leaves one component and reappears several components down the line. Filling these *holes* after the workflow has already started is difficult. Even if there are well-documented control APIs, a workflow, unlike a process on a single host, is tricky to pause, restart, or query 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, indicating failure or success. This technique (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.

At a minimum, the *start* and *stop* events, should have a timestamp, name of the event, host (or host pair for network

events), and a GID. The *stop* event should also include a status code. For example, the Globus GateKeeper might log these monitoring events:

```

Timestamp : 2004-02-02T22:21:35.753024
Event Name : gateKeeper.start
Host : 131.243.2.22
GID : b21746a9-6cb8-4256-93e1-1a310...
.. GateKeeper execution ..
Timestamp : 2004-02-02T22:21:35.839715
Event Name : gateKeeper.end
Host : 131.243.2.22
GID : b21746a9-6cb8-4256-93e1-1a310...
Status : Success

```

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

5 Experimental Results

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

- 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 the Y-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 Experimental Procedure

For this demonstration our methods of inserting the GID into the logging were ad-hoc and specific to the components we were using. This task was simplified by our use of pyGlobus [?], which provides high-level Python language

wrappers for Globus Toolkit components. We quickly integrated NetLogger's Python API with pyGlobus 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 (see below). The GridFTP client was also built using pyGlobus, so transferring the GID to the Reliable File Transfer Service (RFT) could be done within Python.

The GID is communicated to the NetLogger instrumentation in each component through an environment variable, NETLOGGER_DEST, that contains a URL of the form *log destination?const_GID=value*. This sets the file or network destination for the monitoring events, and also adds the name/value pair GID=value to all events. Using this feature, we added GIDs to all NetLogger-instrumented components without changing the instrumentation itself or re-compiling the components. For more details on NetLogger URLs, see [?].

5.2 Analysis of Results

To illustrate how GIDs help to correlate the results, we show, in Figure 2, two versions of the same monitoring data. Both graphs have 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, and 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. Note how 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), are clarified with the addition of GIDs.

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 pro-

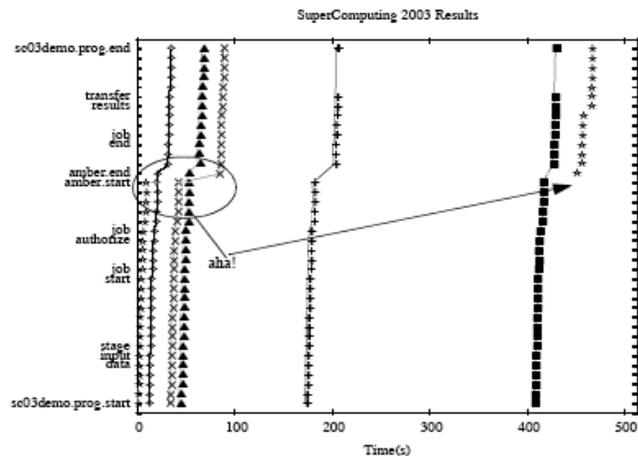
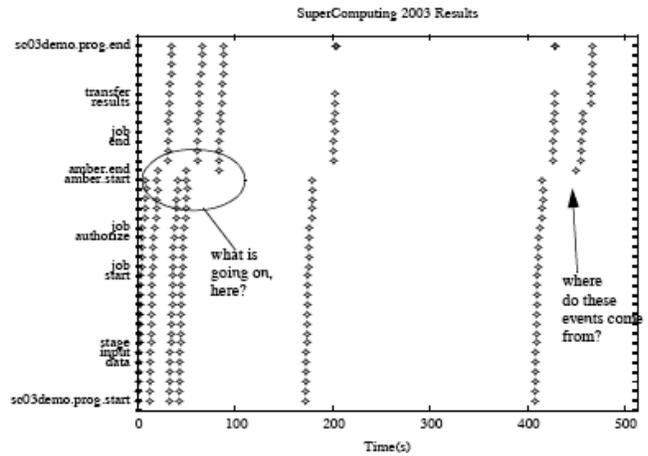


Figure 2. Amber Workflow.

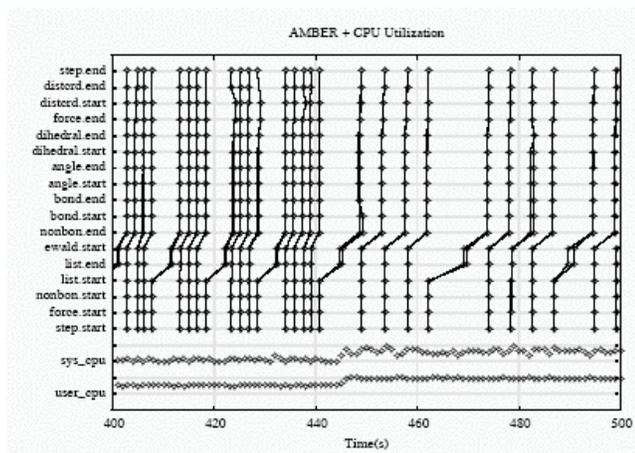


Figure 3. Amber Instrumentation with CPU monitoring.

cess. This allows one to *drill down* to find the source of problems or bottlenecks. This is described in more detail in [?].

5.3 Workflow-Independent Monitoring Data

Some monitoring data cannot be directly associated with a particular workflow. For example, host monitoring data (CPU, disk, memory, etc.) or network monitoring data can encompass, at a given point in time, any number of workflows. Sometimes, as noted in Section 3.2, with other types of context we can infer the GID. But this is not always the case. As a fallback, we must use time stamps. A graph showing this type of analysis is shown in Figure 3. 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. If we could categorize the system and user CPU data by process (PID), and then associate that with a GID, we could be more certain that the change in CPU was indeed correlated with the application workflow's performance. Despite the apparent correlation in Figure 3, the results are actually quite tentative: it is possible that host CPU and AMBER performance change at the same time by coincidence, or due to a third factor (another workflow, a rogue vi process, etc.). Verifying this analysis requires either detailed logs from the entire system or an unambiguous identifier like a GID.

6 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; but currently we know of no compelling reason to justify complicating GIDs with workflow-specific metadata.

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

Acknowledgment

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, Mathematical, Information, and Computational Sciences Division under U.S. Department of Energy Contract No. DE-AC03-76SF00098. This is report no. LBNL-57060.

References

- [1] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Secure, efficient data transport and replica management for high-performance data-intensive computing, 2001.
- [2] A. Bayucan and all. Portable Batch System: External Reference Specification. Technical Report, MRJ Technology Solutions, November 1999.
- [3] Business Process Execution Language for Web Services. <http://www.ibm.com/developerworks/library/ws-bpel/>.

- [4] R. Byrom and all. R-gma: A relational grid information and monitoring system. In *Proceedings of the Cracow '02 Grid Workshop, January 2003*. Web: <http://edms.cern.ch/file/368364/1/rgma.pdf>, 2003.
- [5] R. Chinnichi, M. Gudgin, J. Moreau, J. Schlimmer, and S. Weerawarana. Web Services Description Language (WSDL), Version 2.0 Part 1: Core Language. <http://www.w3.org/TR/wsdl20/>.
- [6] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing, 2001.
- [7] Distributed Computing Environment (DCE). <http://www.opengroup.org/dce/>.
- [8] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Workflow management in grifhyn. In *Grid Resource Management, Kluwer, 2003*. <http://www.isi.edu/deelman/pegasus.htm>, 1998.
- [9] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. Published online at <http://www.globus.org/research/papers/ogsa.pdf>, January 2002.
- [10] Ganglia. <http://ganglia.sourceforge.net/>.
- [11] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and H. Nielsen. SOAP Version 1.2 Part 1: Messaging Framework. <http://www.w3c.org/TR/2003/REC-soap12-part1-20030624/>.
- [12] D. Gunter, B. Tierney, K. Jackson, J. Lee, and M. Stoufer. Dynamic monitoring of high-performance distributed applications. In *11th IEEE Symposium on High Performance Distributed Computing*, 2002.
- [13] Hawkeye: A Monitoring and Management Tool for Distributed Systems. <http://www.cs.wisc.edu/condor/hawkeye/>.
- [14] J. Hollingsworth and B. Tierney. Instrumentation and monitoring. In *The Grid, Volume 2. Morgan Kaufman*, 2003.
- [15] HPSS. <http://www4.clearlake.ibm.com/hpss/>.
- [16] ISO/IEC 11578:1996 Information technology – Open Systems Interconnection – Remote Procedure Call. <http://www.iso.ch/cate/d2229.html>.
- [17] K. Jackson. pyglobus: a python interface to the globus toolkit. In *Concurrency and Computation: Practice and Experience, 14 (13-15), 2002, pp.1075-1084*, 2002.
- [18] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.
- [19] e. M. Little, E. Newcomer. Web services coordination framework (ws-cf). In *OASIS WS-CAF TC Public Documents, July 2003*. <http://www.oasis-open.org/committees/download.php/4345/WSCF.pdf>, 1998.
- [20] W. Matthews and R. L. Cottrell. The pinger project: Active internet performance monitoring. *IEEE Communications Magazine*, pages 130–137, May 2000.
- [21] D. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. IETF RFC 1305. <http://www.ietf.org/rfc/rfc1305.txt>.
- [22] Nagios. <http://www.nagios.org>.
- [23] NetLogger User Manual. <http://dsd.lbl.gov/NetLogger/manuals.html>.
- [24] H. Newman, I. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu. Monalisa: A distributed monitoring service architecture. In *CHEP 2003, La Jolla, California*, March 2003.
- [25] Network Time Protocol General Overview. <http://www.eecis.udel.edu/~mills/database/brief/overview/overview.pdf>.
- [26] B. Patt-Shamir and S. Rajsbaum. A theory of clock synchronization.. In *Proc. of 26th Symp. on Theory of Computing, May 1994.*, 1994.
- [27] D. Pearlman, D. Case, J. Caldwell, W. Ross, T. C. III, S. DeBolt, D. Ferguson, G. Seibel, and P. Kollma. Amber, a computer program for applying olecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to elucidate the structures and energies of molecules. In *Comp. Phys. Commun. 91, pp. 1-41*, 1995.
- [28] Globus Reliable File Transfer Service. <http://www-unix.globus.org/toolkit/docs/3.2/rft/index.html>.
- [29] R. Salz and P. Leach. UUIDs and GUIDs. <http://www.ietf.org/internet-drafts/draft-mealling-uuid-urn-05.txt>.
- [30] A. Shoshani, A. Sim, and J. Gu. Storage resource managers: Middleware components for grid storage. In *In Proceedings of the Nineteenth IEEE Symposium on Mass Storage Systems*, 2002.
- [31] WS-Resource Framework. <http://www-fp.globus.org/wsrft/default.asp>.