Online Workflow Management and Performance Analysis with Stampede

Dan Gunter*, Ewa Deelman‡, Taghrid Samak*, Christopher H. Brooks¶, Monte Goode*, Gideon Juve†, Gaurang Mehta‡, Priscilla Moraes§, Fabio Silva‡, Martin Swany§, Karan Vahi‡
*Lawrence Berkeley National Laboratory, Berkeley, CA, USA
†University of Southern California, Marina Del Rey, CA, USA
‡University of Southern California Information Science Institute, Marina Del Rey, CA, USA
§University of Delaware, Newark, DE, USA
¶University of San Francisco, San Francisco, CA, USA

***Abstract*—Scientific workflows are an enabler of complex scientific analyses. They provide both a portable representation and a foundation upon which results can be validated and shared. Large-scale scientific workflows are executed on equally complex parallel and distributed resources, where many things can fail. Application scientists need to track the status of their workflows in real time, detect execution anomalies automatically, and perform troubleshooting – without logging into remote nodes or searching through thousands of log files. As part of the NSF Stampede project, we have developed an infrastructure to answer these needs. The infrastructure captures application-level logs and resource information, normalizes these to standard representations, and stores these logs in a centralized general-purpose schema. Higher-level tools mine the logs in real time to determine current status, predict failures, and detect anomalous performance.**

I. Introduction

Scientific applications today make use of distributed resources to support computations such as campus resources, grids, clouds, or a mixture of them all. Scientific workflows provide a representation of complex analyses composed of heterogeneous models usually designed by a collaboration of several scientists. Workflow representations and associated middleware help scientists compose and manage the computation as well as automate the validation and sharing of results. Workflows are also a useful representation for managing the execution of large-scale computations.

The majority of execution environments is not immune to failures of resources such as execution hosts and networks. However, unlike in a tightly coupled cluster or enterprise network, application and resource status information is often challenging to collect and understand and analyze in these diverse environments – particularly in real time. Solving this problem is important because misbehaving resources can cause large, complex workflows to take many hours or days longer than necessary to complete. If problems could be identified early, workflow middleware such as the Pegasus Workflow Management System (Pegasus-WMS) [1] has the ability to re-route the tasks onto other resources.

Up to now, tools such as NetLogger [2] have been used to perform off-line log analysis. The Synthesized Tools for Archiving, Monitoring Performance and Enhanced DEbugging (Stampede) project aims to apply the current offline workflow log analysis capability to address reliability and performance problems for large, complex scientific workflows. Specifically, Stampede integrates NetLogger and Pegasus-WMS into a general-purpose framework for analyzing performance and failure states of *running* workflows.

This paper describes the Stampede framework: architecture, components and data models. Scalability of the framework is evaluated in the context of detailed logs from a number of real application workflows. A novel and relevant analysis of the modeled information for workflow failure prediction is presented. Results from analyzing the real application workflows demonstrate the real-time performance and effectiveness of this approach.

II. Applications

Stampede is currently used in the analyses of many scientific applications. The analysis presented here, has been performed on $1,329$ real workflow executions across six distinct applications: Broadband, CyberShake, Epigenome, LIGO, Montage, and Periodograms. The datasets for these applications are summarized in Table I. A brief description of each application follows.

TABLE I
Summary of datasets for each application

	Cumulative Count for Each Application			
Application	**Workflows**	**Jobs**	**Tasks**	**Edges**
Broadband	71	42,060	62,261	161,867
CyberShake	886	288,665	288,665	1,245,131
Epigenome	47	9,918	19,935	26437
LIGO	26	2,116	2,116	6,203
Montage	184	74,851	1,270,718	531,663
Periodograms	116	95,424	2,219,071	96,296
TOTAL	1,329	513,034	3,862,766	2,067,597

Broadband [3], [4] is a computational platform used by the Southern California Earthquake Center to simulate the impact of of an earthquake at one of the Southern California faultlines, to guide the building design procedures in a given area.

CyberShake [5], [6] is designed to perform physics-based probabilistic seismic hazard analysis calculations for geographic sites in the Southern California region. CyberShake workflows analyze over $415,000$ rupture variations, each representing a potential earthquake.

Epigenome workflows [7] are a data processing pipeline that performs various genome sequencing operations using the DNA sequence data generated by the Illumina-Solexa Genetic Analyzer system [8].

LIGO The Laser Interferometer Gravitational Wave Observatory (LIGO) is attempting to detect *gravitational waves* predicted by Einstein's theory of general relativity [9], [10]. The LIGO Inspiral Analysis Workflow is used to analyze the data obtained from the coalescing of compact binary systems such as binary neutron stars and black holes.

Montage was created by the NASA/IPAC Infrared Science Archive to generate custom mosaics of the sky [11]. Montage workflows re-project, rotate, and normalize levels in the input images to form the output mosaic.

Periodograms calculate the significance of different frequencies in time-series data to identify periodic signals [12]. The periodigram application designed by IPAC (Caltech) is being used today to search for exoplanets in the Kepler mission's data.

III. Performance Analysis System

A. Framework

The Stampede architecture is shown in Figure 1. The components of the architecture can be divided into three groups: the workflow execution engine, log collection components, and the archival and analysis components. The workflow engine and execution components are part of Pegasus-WMS, described in detail elsewhere [1].

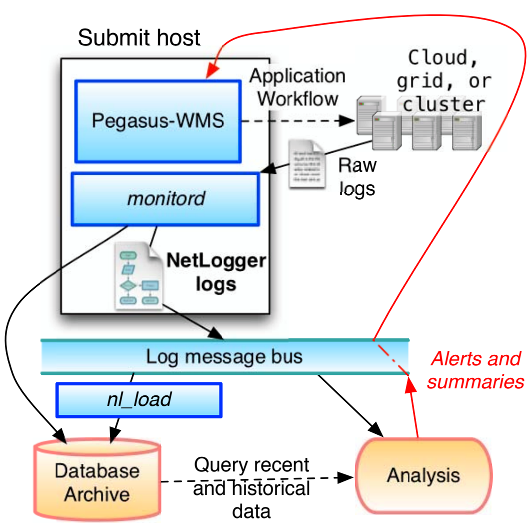


Fig. 1. Overview of the Stampede Architecture.

1) Log Collection: The log collection component begins with "raw" workflow logs. All the workflows analyzed here used Condor as the underlying job manager. During the execution of a workflow, Condor DAGMan [13] writes its log file (`dagman.out`) in near real-time. This file contains the status of each job, as well as the pre-execute and post-execute scripts. Some of the analyzed workflows augment this information with a job wrapper called Kickstart [14], which records system statistics such as resource usage and open files for each invocation within that job.

As these logs are staged back to the submission host by Pegasus-WMS and Condor, they populate directories on disk. Although this method incurs some additional latency as compared to streaming the data directly over the network, it has the advantage of leveraging the grid security mechanisms to guarantee that the monitoring data cannot be viewed by a third party. We have developed a program called *monitord* that continuously parses these log files. The *monitord* program is run automatically by Pegasus when a workflow is started, although it can also be run in a submit directory after a workflow's execution has finished.

The main job of *monitord* is to map the incoming workflow logs to the general data model described in Section III-B. The abstract model is made concrete either by emitting NetLogger log events (short text messages), or by populating tables in a relational database.

2) Log Message Bus: The NetLogger log events are placed on a message bus, which is used to decouple the consumers of the streaming workflow data from the many possible clients. For this function, we chose to use RabbitMQ [15] [16], a popular implementation of the standard Advanced Message Queueing Protocol (AMQP) [17] based on the Erlang Open Telecom Platform (Erlang/OTP) [18]. AMQP defines an efficient and flexible publish/subscribe interface that is independent of the data model. AMQP uses a central server, or *broker*, but the RabbitMQ implementation can be scaled linearly by distributing the broker over multiple physical nodes.

We use the hierarchical datatype of the NetLogger log message, called the *event* field, to route messages through using an AMQP *topic queue*. Topic queues allow clients to subscribe to messages matching a prefix of the message type, *e.g.*, to receive all "stampede.job" messages or just the subset starting with "stampede.job.mainjob". This capability provides a great deal of flexibility in gluing together analysis components, while maintaining good performance and keeping implementations simple.

3) Database Archive: The *monitord* program can insert data directly into an SQL database. Alternately, *monitord* can write to the message bus and a NetLogger component called *nl_load* can asynchronously insert them into a database. The former approach avoids copying the message, while the latter allows the message bus to impedance match the log message arrival rate with the database insertion rate.

Either way, the bulk of the work is performed by NetLogger modules that use the Python ORM, SQLAlchemy [19]. One advantage to using SQLAlchemy over SQL is the built-in support for a number of relational database products, including SQLite, MySQL and PostgreSQL. Once implemented, the same code can switch to any of these database products by simply changing the connection string. SQLAlchemy is sometimes slower than plain SQL, but the load and query results in Section III-C show that this overhead is not prohibitive.

4) Analysis Components: Analysis components can obtain data in two ways: they can query the database, or else subscribe to a stream of log messages from the message bus. The analysis presented here used SQL queries, which were sufficiently performant and simpler to implement natively in our chosen analysis language, *R* [20]. However, we have verified that the analysis does not need to perform any transformations of the data in order to operate directly on the live event stream. We have developed a prototype interface using Python AMQP libraries and the Python/R RPy2 [21] module, which pushes R dataframes to the analysis functions. Future work will use this interface where appropriate.

B. Data Model

We have developed an abstraction of workflows and mapped this to a canonical, general-purpose representation as a stream of log records. We have also implemented archival of these logs in a relational database, for efficient historical analysis. The current implementation uses the Pegasus-WMS workflow engine, but the representations are designed to be extensible to other workflow engines and potentially other distributed applications.

Each workflow model consists of two sub-models: an *abstract workflow* and an *executable workflow*. There is one abstract workflow, which describes the computations, data movement, and dependencies in a resource-independent way. Computational tasks describe the computations using a logical name and logical input/output data files. Tasks are arranged in a hierarchy that indicates logical dependencies between the inputs and outputs.

Each abstract workflow is mapped to one executable workflow based on a set of target resources. In this workflow, the logical tasks are mapped to concrete resources, executables are specified, etc. Individual tasks in the abstract workflow may be joined together or optimized away entirely. Additional tasks, not present in the abstract workflow, are added to create directories and manage data staging and registration. In Pegasus-WMS, multiple abstract tasks can be automatically *clustered* into a single executable job; clustering is often helpful in the case where individual tasks are of short duration and may incur comparatively high overheads during execution.

Clustering has an impact on failures, however. If for example, we have a cluster of 5 tasks, and one task fails, then the whole cluster fails. In the case of a non-clustered workflow, the other 4 tasks may have executed successfully. In practice, we try to strike a balance between performance and the cost of recovery from failures by not making the clusters "too large". This value is currently determined in a domain-specific fashion, but future work could derive cluster size from the observed failure probabilities.

1) Terminology: We will describe the details of this data model in the context of the implementation with Pegasus-WMS. Although the model allows for arbitrary graphs of node dependencies, Pegasus-WMS defines the following terminology to precisely differentiate between the various activities:

- Workflow: Container for an entire computation. Execution of a workflow is called a run.
- Abstract workflow graph (AW): Input graph of tasks and dependencies, independent of a given run on specific resources. We assume AW to be a directed acyclic graph.
- Executable workflow (EW): Result of mapping an AW to a specific set of resources. The cardinality of the AW task to EW job mapping is many-to-many. In Pegasus, this step is called planning.
- Sub-workflow: A workflow that is contained in another workflow.
- Task: Representation of a computation in the AW.
- Job: Node in the EW. May represent part of a task (e.g., a stage-in/out), one task, or many tasks.
- Job instance: Job scheduled or running by underlying system (e.g., DAGman). Due to retries, there may be multiple job instances per job.
- Invocation: One or more executables associated with a job instance. Invocations are the instantiation of tasks, whereas jobs are an intermediate abstraction for use by the planning and scheduling sub-systems.

These terms define components for both abstract and executable workflows.

2) Log Events: The streaming representation uses NetLogger's Best Practices (BP) log format and model. In BP, each log record, called an "event", has a name, timestamp, severity level, and arbitrary additional metadata in the form of name/value pairs. NetLogger provides a simple ASCII representation that was designed to interoperate easily with UNIX *syslog* and its cousins *syslog-ng* and *rsyslog*, as well as command-line tools like *grep*.

In the BP model, what classical models of system state [22], [23] would call "activities" are each represented by two events: one for the start and one for the end of the activity. Relations between activities, *e.g.*, parent/child or start/end of a single activity, are recorded by including in each event unique identifiers for the activity and any related activities. For example, the `job.start` and `job.end` activities have identifiers for both the job and parent workflow (which may be a sub-workflow). Thus, a path can be traced from any activity up to its top-level workflow.

Additional metadata such as the username and job "name" are attached to the events. For example, a single job in a workflow would be represented in BP log events like the following:

```
ts=2010-02-20T23:09:13.0Z event=workflow.start level=Info
  wf.id=8bae72f2-31b9-45f4-bdd3-ce8032081a28
ts=2010-02-20T23:09:26.0Z event=job.mainjob.start level=Info
  condor_id=3309.0 job.id=1
  wf.id=8bae72f2-31b9-45f4-bdd3-ce8032081a28
  name=create_dir_montage_0_viz_glidein jobtype=compute
ts=2010-02-20T23:14:06.0Z event=job.mainjob.end level=Info
  job.id=1 wf.id=8bae72f2-31b9-45f4-bdd3-ce8032081a28
  remote_user=vahi site_name=viz_glidein status=0
...
ts=2010-02-20T24:34:06.0Z event=workflow.end level=Info
  wf.id=8bae72f2-31b9-45f4-bdd3-ce8032081a28 status=0
```

In the example, the workflow is identified with the UUID in `wf.id`. Within this scope, job identifiers, `job.id`, can then be small integers assigned in the order the jobs are submitted. Standard suffixes added to `event` indicate the start and end of an activity, and the `status` field provides a simple way of encoding success or failure.

3) Relational Schema: The Stampede database schema, shown in Figure 2, captures the attributes and relationships of the data model described earlier. Each box in the figure represents a database table, with the name underlined. The text in italics below the name gives the associated component using the standard terminology defined in III-B1, or a description of its function in the schema. Although the correspondence with the model is mostly straightforward, a few items merit further description. The history of states (planned, submitted, running, complete, etc.) experienced by workflows and jobs is recorded in the `workflow_state` and `jobstate` tables, respectively. The parent/child relationships for jobs and tasks are recorded in the `job_edge` and `task_edge` tables; note that this allows inference of the hierarchy in EW elements as well. The self-reference in the `workflow` table expresses the workflow/sub-workflow hierarchy. Auxiliary data, such as execution host statistics and file information, are not shown in the diagram.

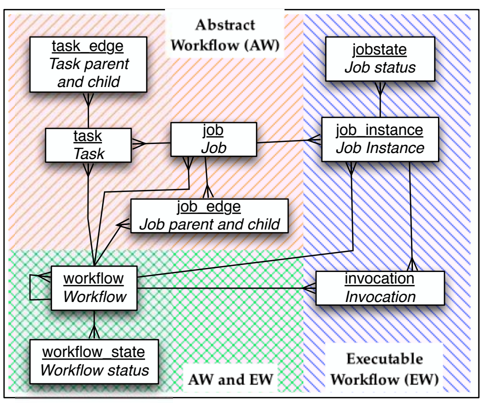


Fig. 2. Diagram of the Stampede relational schema. Shaded areas indicate which tables are used for Abstract Workflows (AW), Executable Workflows (EW), or both.

C. System Performance

The results below show that the system performance is sufficient to process and analyze data online. In most cases, performance exceeds the minimum requirement by orders of magnitude. All performance results were obtained on MySQL 5.1.49-3 running on a Debian 2.6.32-5-amd64 server. The server had two Intel Xeon X5650 processors (each with 6 cores) and 64GB of RAM.

For online analysis, the performance target is of course the rate of events arriving from a running workflow. This varied widely across and even within our datasets, as shown in the Arrival Rate columns in Table II. An average event is roughly 100 bytes of data in the NetLogger BP format, so the bandwidth is well within the capabilities of current WANs and disks.

TABLE II
SUMMARY OF DATA ARRIVAL RATES, IN EVENTS PER SECOND

Application	Arrival rate		Load rate	
	Mean	Max.	Mean	Min.
Broadband	10	52	453	366
CyberShake	77	95	444	417
Epigenome	6	118	412	199
LIGO	7	7	282	282
Montage	18	246	545	355
Periodograms	3	22	538	397

1) Data Load Performance: To measure load performance, the application datasets were loaded into MySQL using the *nl_load* program with the *stampede_loader* module. The minimum and mean load rate achieved for the same datasets is shown in the right-hand column of Table II. Even in the worst case of matching the minimum load rate against the maximum arrival rate, the load rate exceeds the event rate. On average the ratio of the two values is orders of magnitude. Although not shown, experiments also verified that the load rate did not depend significantly on the size of the workflow. Therefore, in general, a single database can keep pace with many concurrent workflows. Burstiness in event arrival rates will temporarily increase the latency between event arrival time and the time the data is added to the database, but this does not cause long-term disruptions of the system as the log message bus provides efficient buffering.

Although the time taken to load a workflow into the database is a linear function of the number of "events" in that workflow, the number of monitoring events generated by a Pegasus-WMS workflow is to some extent tunable. Recall that Pegasus-WMS has the ability to *cluster* multiple abstract jobs into a single executable job. This reduces the number of distinct jobs that must be submitted and managed by the Condor execution system, thus reducing overheads. A side-effect is to also reduce the number of `job.start` and `job.end` events, which speeds up the database loading process. Note that none of the relevant task details are lost by this process, as it simply reduces the number of jobs necessary to execute these tasks on the target resources.

The improvement due to clustering turns out to be super-linear. In an experiment with different degrees of clustering on a Montage workflow, the measured effect closely fits the model: $L = 0.24 + 3 * E$, where L is the ratio of clustered to unclustered load times, and E is the ratio of clustered to un-clustered events. Thus clustering reduces the number of events and increases the load rate at the same time. This may be due to fewer updates across tables when there are fewer jobs per workflow, but verification of this hypothesis is outside the scope of this discussion.

2) Data Retrieval Performance: Part of the analysis presented in this work queries the relational database to retrieve its input data. Therefore, the database query performance is an important factor. To measure query performance, we used a representative set of queries, shown in Table III. These queries are broken into three categories: Counters, Timing, and Analysis. The Counters and Timing queries are a slightly modified subset of the queries given in [24], chosen for coverage of the dimensions of a workflow (state, job type, and host) and relevance to online displays such as dashboards. The Analysis queries are those used for the job failure analysis given in Section IV. The query type names used in Table III will be used in the discussion that follows.

TABLE III
REPRESENTATIVE QUERIES FOR PERFORMANCE ANALYSIS

Category	Name	Description
Counters	JobsTot	Total num. jobs in workflow
	JobsState	Num. jobs in a given state (run, queued, etc.)
	JobsType	Num. jobs of a given type
	JobsHost	Num. jobs on a given host
Timing	TimeTot	Total runtime of workflow
	TimeState	Time in a given state (run, queued, etc.)
	TimeType	Time by jobs of a given type
	TimeHost	Time running on a given host
Analysis	JobDelay	Durations of jobs from start to end, and delay between start and execution.
	WfSumm	Workflow duration and number of jobs total, failed, successful, and restarted.
	HostSumm	Number of jobs, successful job count and time, and failed jobs count and time for a given host.

We experimentally evaluated the performance of these queries on each of the application databases.

The results are shown in Figure 3. The median number (from the 10 runs) of queries per minute is shown for each combination of application and query type. A dashed black line indicates the median real-time data arrival rate, which was calculated by simply dividing the number of inserted rows by the workflow wallclock duration. Note that this is not the same as the event rate shown in Table II because the `edge` data, while log events, are not used by these queries.

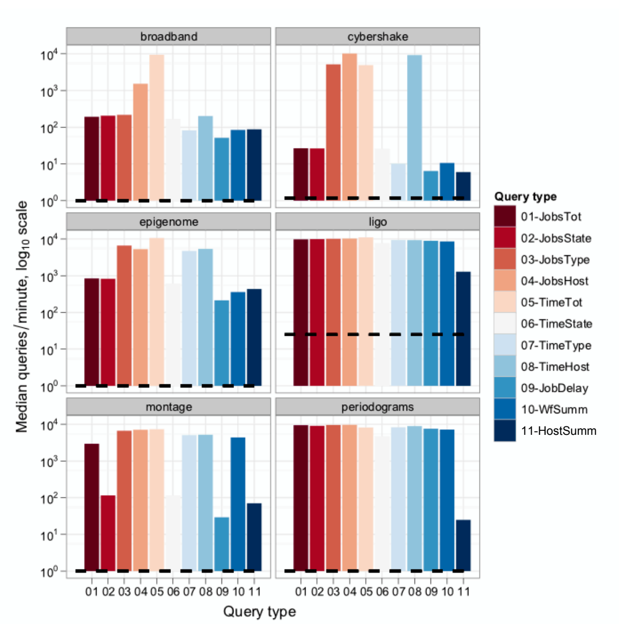


Fig. 3. Median number of queries per minute, for each application and query type. The dashed black line shows the median arrival rate, in database rows per minute.

From the figure, it is obvious that a query can be performed on a window of incoming database rows in a fraction of the time it takes for them to arrive from the application. Put another way, many workflows could be queried in parallel, in real time, on the same database server. Taken together with Table II, these results show that the maximum number of workflows that can be accommodated will be limited by the data load rates.

IV. Workflow Analysis

This section describes the analysis capabilities that we are developing with the Stampede framework: algorithms for workflow failure prediction, probabilistic anomaly detection, and a web-based dashboard to visually analyze and monitor workflows.

A. *Workflow Failure Prediction*

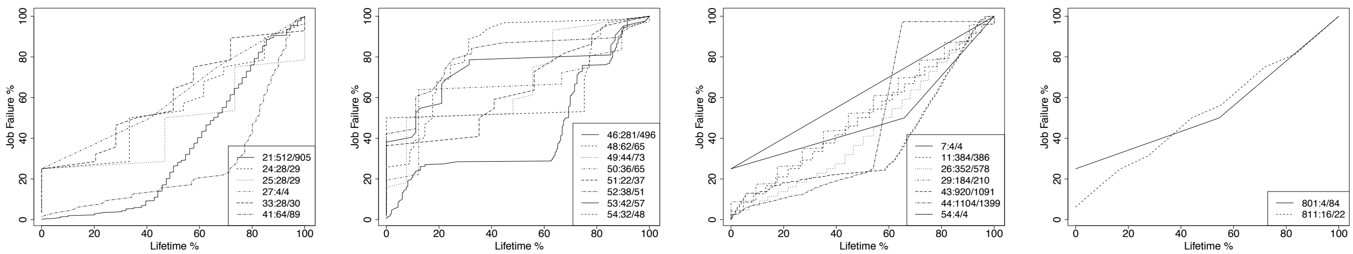
Complex scientific workflows often experience failures due to temporary or localized problems with resources. For example, a compute node may have a bad disk, or a network file server may be overloaded and time out. To cope with these situations Pegasus-WMS has the ability to retry jobs within a workflow multiple times. However, these "soft" failures may also be the symptoms of a deeper problem with the workflow, one that will eventually cause the entire workflow to slow or stop. The goal of workflow failure prediction is to automatically determine whether a given pattern of individual failures is a minor glitch or indicative of a deeper problem.

A summary of workflow datasets was given earlier in Table I. For this analysis, a slightly different set of data was used. This dataset had $1,332$ workflow instances with $288,668$ jobs, $577,330$ tasks, and $1,245,845$ edges (in the workflow graph). This is also a sufficiently large dataset for studying workflow behavior.

As a first step in exploring this dataset, we look at job failures for each workflow over time. This gives an overall timeline for workflow performance in terms of failure characteristics. To normalize the results for different absolute workflow sizes, we use the statistic of the percentage of failed jobs, $\frac{failed\ jobs_t}{failed\ jobs_{total}} \times 100$, calculated at fixed intervals, t, over the lifetime of each workflow. Failures are easily inferred from the data: they are represented in the BP logs by a `job.end` event with a negative value for the `status`, which becomes a state of `JOB_FAILURE` in the `jobstate` table in the database. Job failure characteristics can affect the total duration of the workflow, and the number of restarts.

Figure 4 shows the pattern of failed jobs for selected workflow runs from each application. On the X axis is the percentage of the total duration of the workflow, and on Y the percentage of the total number of failed jobs. The legend shows the exact number of failed and total jobs for each run in the format `run-number:failed/total`. For the *Epigenome*, *Montage* and *Broadband* applications, we only show workflows for which more than 50% of the jobs failed. Through the total of 881 runs of *Cybershake*, only 2 workflows contained at least one job failure. *LIGO* workflows are not shown because they did not suffer any job failures.

From the figure, it is clear that failure patterns vary across applications. The *Montage* runs have three distinct phases:



(a) Epigenome (b) Montage (c) Broadband (d) Cybershake

Fig. 4. Job failure percentage at different time steps in the lifetime of the workflow. Each line represents a single run. The legend shows the wf_id, and the actual number of failures/total jobs.

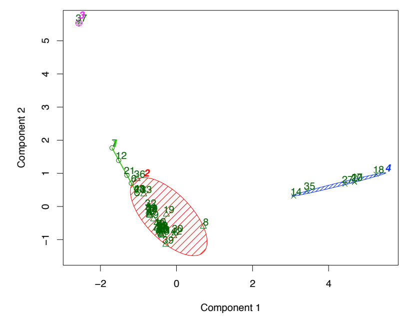
a sequence of initial failures, followed by a period with no failures, then a sudden increase of failures near the end. For these runs, what may appear as a temporary initial episode of failures is actually indicative of a long-range failure pattern. For *CyberShake* and *Broadband*, most runs show a steady increase in failure percentage from the beginning to end. This means that a few initial failures may be an indicator of many more failures for that run. For *Epigenome*, the failure patterns are more varied. Sometimes there are two phases of slow and quicker steady increases in failures, and sometimes there is a “staircase” pattern of alternating failure and non-failure periods.

It is also evident from Figure 4 that there are distinct patterns of job failures within each application. For example, in *Broadband* workflow runs `11`, `29`, and `26` all have a fine-grained staircase pattern of job failures; whereas the other runs have job failures at only a few points in time. The existence of these patterns suggests that jobs can be effectively clustered within each application.

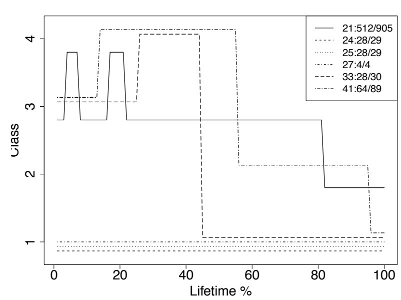
We have developed clustering algorithms that can effectively group workflows according to different patterns of failure. More details on the methods are available from [25], but in brief: we used the efficient *k-means* clustering algorithm [26], [27], with randomized initial centers and the (standard) Euclidean distance metric. The algorithm takes as input four parameters: workflow duration, failed and successful job durations, and percentages of failed jobs. The result of this algorithm is a set of clusters, each of which are then compared to job failure rates and classified as belonging (or not) to the class of “high failure workflows” (HFWs).

As an example of this algorithm, Figure 5 shows clusters for *Epigenome* workflows. The graph shows the projection in two dimensions (first 2 principal components) of the *k-means* clusters determined from historical data. The centers of these clusters are saved, and the cluster with the most failures is labeled the HFW class – in this case, Class 1, the tight blue cluster on the right-hand side of Figure 5(a). Then, for each time step in a given workflow run, the workflow is classified as belonging in the cluster whose center is “nearest”. Figure 5(b) shows the changing classifications of the *Epigenome* workflows over time.

In most cases, workflows that experience many failures will converge to the HFW class. In the example, this is



(a) Clusters



(b) Classification

Fig. 5. Epigenome Online Analysis

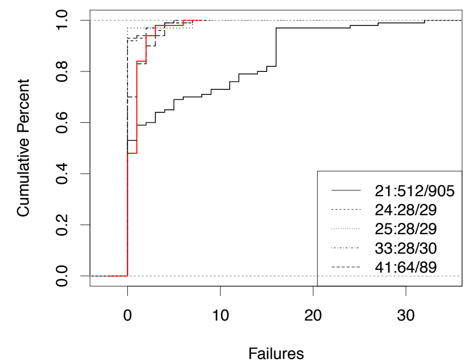
represented by the selected workflows converging to Class 1 at the bottom of Figure 5(b). It is also evident that, in most cases, once a workflow belongs to the HFW class, it is unlikely to revert. Therefore, classification as HFW provides a very clear indication that a workflow will continue to experience job failures, and that in-depth root cause analysis and/or corrective action are warranted. This is of course particularly true for workflows classified early in their execution cycle. Though only *Epigenome* is shown, similar results were obtained for other applications.

B. *Anomaly Detection*

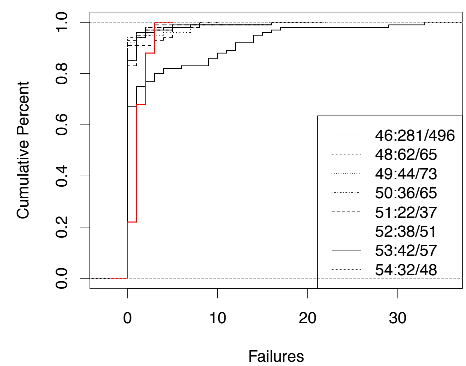
The second analysis example is also concerned with workflow job failures. Rather than trying to predict future workflow behavior, the goal is to detect periods in which anomalous numbers of failures occur. The methods used are probabilistic: modeling the past distributions of job failures and comparing them with new data from running workflows. Each application is considered independently. The random variable to consider here is the number of failed jobs during a certain time window. As before, to facilitate comparing behaviors of individual workflow runs, the lifetime of a workflow is represented as a percentage of the total. The analysis time windows are, for now, arbitrarily chosen to be $1/100$ of the total runtime.

Figure 6 shows the empirical cumulative distribution function (ECDFs) for job failures from workflow runs of *Epigenome* and *Montage*. On the X axis is the total number of failures, and on the Y axis is the proportion of time windows experiencing that number of failures or less. Visual inspection of the distributions shows that failure trends are similar for most workflow runs. Further detailed analysis of individual workflows from both applications showed that the total number of jobs is high for some workflows due to repeated job failures, such as workflow 21 for *Epigenome* and workflow 46 for *Montage*. These workflows, which correspond to the two right-shifted lines in the ECDFs, are anomalies we would like to detect.

In order to distinguish between the anomalies and other workflows, we model the failure distribution over time as a Poisson process, which has distribution given by $P(x) = \frac{e^{-\lambda}\lambda^x}{x!}$. A random subset of historical workflows with a low number of total failures are used to determine λ, the expected number of failures per time period. The red line in Figure 6 shows the fitted distribution using the mean job failure from one normal workflow run. The fitted distribution is very close to the "normal" workflows. This result differs from failure analysis presented in [28], where the log normal distribution was a better fit for failures. In our analysis, a Poisson model is more appropriate since we are dealing with a more localized dataset. Each workflow runs within relatively a small period of time (minutes or even hours), compared to days or years in [28]. The characteristics of the underlying resources are less likely to change in this time range, making a constant failure rate, the Poisson λ parameter, a more reasonable assumption. For much longer workflows an alternate distribution or coupling of Poisson with change-point analysis would be appropriate.



(a) Epigenome



(b) Montage

Fig. 6. ECDFs for runs of Epigenome and Montage. Fitted Poisson distribution is shown in red.

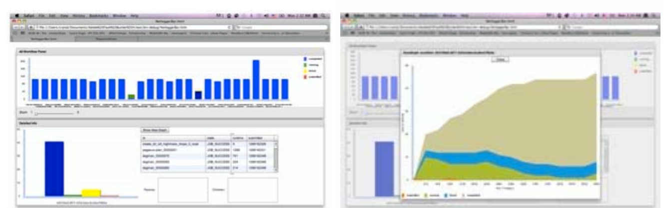


Fig. 7. Barchart and areachart views of the Stampede dashboard.

C. *Workflow Status Dashboard*

We have also provided a set of web-based tools to allow Stampede users to examine runs visually. A simple HTTP-based API was built on top of the Stampede database, thereby providing a flexible, architecture-neutral way for web clients to access data either in an online or offline format. Two screenshots of the dashboard are shown in Figure 7.

The left screenshot shows the barchart view. In this view, users can see the elapsed time for each workflow within a run, divided into the time its jobs spent in the submitted, running, completed, and failed states. By clicking on a particular workflow, the user brings up a more detailed display in the bottom panel in which the details of each job can be seen, such as its name, state, running time, submit time, parents, and children. This allows a user to easily zoom in on particular aspects of a run to identify failures and bottlenecks.

The right screenshot shows the areachart view. In this view, the user can see the performance of a run over time, with the fraction of jobs in each state indicated by strata in the chart. By clicking in a region of this chart, the user can then get a more detailed snapshot of each job's state at that point in time, allowing the user to get a clear understanding of the dynamics of the run and potentially identify bottlenecks.

V. Related work

A large number of systems have been developed for monitoring and performance analysis in grid environments [29], [30]. These systems focus on monitoring grid jobs and infrastructure, and have limited support for data relevant to workflow applications, such as workflow dependencies, and task clustering data.

Many workflow systems have some form of integrated monitoring capability. Visual workflow systems such as Kepler [31], Triana [32], and Taverna [33] support runtime monitoring through a graphical user interface. The SWAMP [34] workflow system provides monitoring information through a web interface. The monitoring features of these systems are focused primarily on reporting, not on online performance analysis and prediction. P-GRADE [35] is one notable exception that supports integrated performance analysis and debugging. Our focus is on a decoupled interface that allows both *export* and *import* of performance information, e.g., to and from perfSONAR [36].

Several workflow systems have been integrated with external monitoring tools and middleware. A few different monitoring and analysis tools have been developed for the ASKALON system [37], including an online, service-oriented system based on SCALEA-G [38], [39], and the Perftool system [40], which provides a graphical user interface for inspecting and analyzing the ASKALON logging database at runtime. Kepler has been used with the MidMon monitoring system [41]. Our approach captures more details about workflow execution than these previous systems and has been used successfully with larger workflows.

In addition to performance analysis and debugging, the information collected by Stampede could be used to answer *provenance* queries regarding the origin and history of data items. Provenance is focused on tracing the lineage of data in order to support repeatability and ensure data quality in scientific computations. Provenance in the context of scientific workflows has been studied extensively, and many workflow systems support provenance collection and analysis [42]. Support for provenance can be built into a workflow management system, or it can be provided through external, service-oriented provenance systems such as Karma [43]. Kepler [44] and Triana have native support for provenance collection and retrieval. Triana has been integrated with external services such as those provided by the EU provenance project [45]. And Pegasus has been integrated with the PASOA provenance system [46], [47]. These external provenance services use a similar architecture and implementation as Stampede, and collect similar data. Although the approach described in this paper can be used for provenance collection, the primary focus of the Stampede project is to process information about workflow execution for the purposes of performance analysis and anomaly detection.

Failure prediction using event correlation has been studied in [48]. Temporal and spatial correlations are proposed using multiple features (system information, utlization, packet count, etc). The analysis was performed on a large dataset spanning a one-year period. The large available dataset makes using supervised learning, but might not fit general online learning when low level grid conditions are unknown.

Streaming data processing systems like TelegraphCQ [49] provide a general solution to the problem of online processing of streaming data, whereas Stampede is more focused on troubleshooting for distributed workflows. Commercial products like Splunk [50] provide data for IT analytics across various system layers, but are not built to efficiently extract the dependency graph needed for workflow analysis. Data analytics tools from industry, such as the Percolator system from Google [51] also addresses some of the issues with online ingestion of data. These approaches aren't directly comparable, but the techniques could be applied to Stampede in the future.

A comprehensive offline analysis for 9 years of LANL failure logs was presented in [28]. The analysis produced probability models for failures in different systems with respect to various factors including root cause, time between failures and time to repair. This offline analysis gives guidelines for possible probability models that can be extended to suit our online analysis.

VI. Conclusion

Applications keep scaling up to ever larger systems and the execution environment is also growing in complexity. Although cloud technologies are simplifying application deployment, information about the application execution, including failure behavior, remains hard to understand and analyze in real time. In this paper, we showed an approach to the problem, where we developed an execution information capture and analysis system which is capable of streaming and/or storing workflow performance information in real time.

We demonstrated the system in the context of a number of real application workflows, which were executed in distributed environments that included campus, grid, and cloud resources. We showed that the system and associated analysis capabilities are performant enough to provide results in real-time. Our job failure analysis confirmed that the longer a workflow runs, the more failure-prone it becomes. The analysis also showed that jobs that are likely to fail may be executing for a long time and then fail; therefore early recognition of "doomed" workflows is important for overall efficiency.

Future work will apply the information about failure workflows to restart and recover these workflows without waiting for them to fail entirely. This work will require the addition of adaptation capabilities to the Pegasus-WMS system, and additional analysis to track the effectiveness of these adaptations. In addition, knowledge of which workflows are likely to fail can be used to condition measurement granularity. This is part of the general technique called *adaptive monitoring*[52], a necessary research direction in this age of information deluge.

Acknowledgment

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract DE-AC02-05CH11231. Additional support was provided by NSF grant OCI-0943705.

REFERENCES

[1] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. Berriman, J. Good *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[2] D. Gunter and B. Tierney, "Netlogger: A toolkit for distributed system performance tuning and debugging," in *Integrated Network Management, IFIP/IEEE Eighth International Symposium on Integrated Network Management (IM 2003)*, ser. IFIP Conference Proceedings, vol. 246. Kluwer, 2003, pp. 97–100.

[3] R. Graves and A. Pitarka, "Broadband ground-motion simulation using a hybrid approach," *Bulletin of the Seismological Society of America*, vol. 100, no. 5A, p. 2095, 2010.

[4] "Broadband working group." [Online]. Available: http://scec.usc.edu/research/cme/groups/broadband

[5] P. Maechling, E. Deelman, L. Zhao, R. Graves, G. Mehta, N. Gupta, J. Mehringer, C. Kesselman, S. Callaghan, D. Okaya, H. Francoeur, V. Gupta, Y. Cui, K. Vahi, T. Jordan, and E. Field, "Scec cybershake workflows – automating probabilistic seismic hazard analysis calculations," in *Workflows for e-Sciences*, I. Taylor, E. Deelman, D. Gannon, and M. Shield, Eds. Springer, 2006.

[6] S. Callaghan, P. Maechling, E. Deelman, K. Vahi, G. Mehta, G. Juve, K. Milner, R. Graves, E. Field, D. Okaya, D. Gunter, K. Beattie, and T. Jordan, "Reducing time-to-solution using distributed high-throughput mega-workflows - experiences from scec cybershake," in *Proceedings of the 2008 Fourth IEEE International Conference on eScience*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 151–158. [Online]. Available: http://portal.acm.org/citation.cfm?id=1488725.1488889

[7] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. Berman, and P. Maechling, "Scientific workflow applications on Amazon EC2," in *E-Science Workshops, 2009 5th IEEE International Conference on*. IEEE, 2010, pp. 59–66.

[8] "USC Epigenome Center." [Online]. Available: epigenome.usc.edu

[9] "LIGO Project." [Online]. Available: www.ligo.caltech.edu

[10] D. Brown, P. Brady, A. Dietz, J. Cao, B. Johnson, and J. McNabb, "A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis," in *Worflows for e-Sciences*, I. Taylor, E. Deelman, D. Gannon, and M. Shield, Eds. Springer, 2006.

[11] G. Berriman, E. Deelman, J. Good, J. Jacob, D. Katz, C. Kesselman, A. Laity, T. Prince, G. Singh, and M.-H. Su, "Montage: A grid enabled engine for delivering custom science-grade mosaics on demand," in *SPIE Conference 5487: Astronomical Telescopes*, 2004.

[12] "Periodograms." [Online]. Available: www.ipac.caltech.edu

[13] "Dagman." [Online]. Available: www.cs.wisc.edu/condor/dagman

[14] J. Vockler, G. Mehta, Y. Zhao, E. Deelman, and M. Wilde, "Kickstarting Remote Applications," in *International Workshop on Grid Computing Environments*, no. 0, 2007.

[15] "RabbitMQ." [Online]. Available: www.rabbitmq.com

[16] J. O'Hara, "Toward a commodity enterprise middleware," *Queue*, vol. 5, pp. 48–55, May 2007. [Online]. Available: http://doi.acm.org/10.1145/1255421.1255424

[17] "Advanced message queuing protocol." [Online]. Available: www.amqp.org

[18] "Erlang Open Telecom Protocol." [Online]. Available: http://www.erlang.se/

[19] "SQLAlchemy." [Online]. Available: www.sqlalchemy.org

[20] "R." [Online]. Available: www.r-project.org

[21] "Rpy2." [Online]. Available: rpy.sourceforge.net

[22] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of computer programming*, vol. 8, no. 3, pp. 231–274, 1987.

[23] C. Hoare, "Communicating sequential processes," *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, 1978.

[24] H.-L. Truong, S. Dustdar, and T. Fahringer, "Performance metrics and ontologies for grid workflows," *Future Generation Computer Systems*, vol. 23, no. 6, pp. 760 – 772, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/B6V06-4MWXPP7-1/2/e5a7c8d9748a2b9e0b5c8ad3bc37dd30

[25] T. Samak, D. Gunter, E. Deelman, G. Juve, G. Mehta, F. Silva, and K. Vahi, "Online Fault and Anomaly Detection for Large-Scale Scientific Workflows," in *13th IEEE International Conference on High Performance Computing and Communications (HPCC-2011)*, IEEE. Banff, Alberta, Canada: IEEE Computer Society, Sep. 2011.

[26] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, L. M. L. Cam and J. Neyman, Eds., vol. 1. University of California Press, 1967, pp. 281–297.

[27] J. A. Hartigan and M. A. Wong, "A K-means clustering algorithm," *Applied Statistics*, vol. 28, pp. 100–108, 1979.

[28] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," in *Proceedings of the International Conference on Dependable Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 249–258. [Online]. Available: http://portal.acm.org/citation.cfm?id=1135532.1135705

[29] S. Zanikolas and R. Sakellariou, "A taxonomy of grid monitoring systems," *Future Generation Computer Systems*, vol. 21, no. 1, pp. 163–188, Jan. 2005.

[30] M. Gerndt, Z. Balaton, G. Gombs, P. Kacsuk, Z. Nmeth, N. Podhorszki, H. Truong, T. Fahringer, E. Laure, M. Bubak, and T. Margalef, "Performance tools for the grid: State of the art and future," in *APART White Paper*, 2004.

[31] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludscher, and S. Mock, "Kepler: An extensible system for design and execution of scientific workflows," in *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, 2004, pp. 423–424.

[32] I. Taylor, M. Shields, I. Wang, and A. Harrison, "Visual grid workflow in Triana," *Journal of Grid Computing*, vol. 3, no. 34, pp. 153–169, 2005.

[33] T. Oinn *et al.*, "Taverna: lessons in creating a workflow environment for the life sciences," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1067–1100, 2006.

[34] Q. Wu, M. Zhu, X. Lu, P. Brown, Y. Lin, Y. Gu, F. Cao, and M. Reuter, "Automation and management of scientific workflows in distributed network environments," in *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum (IPDPSW)*, 2010, pp. 1–8.

[35] P. Kacsuk, G. Dzsa, J. Kovcs, R. Lovas, N. Podhorszki, Z. Balaton, and G. Gombs, "P-GRADE: a grid programming environment," *Journal of Grid Computing*, vol. 1, no. 2, pp. 171–197, 2003.

[36] A. Hanemann, J. Boote, E. Boyd, J. Durand, L. Kudarimoti, R. Lapacz, M. Swany, S. Trocha, and J. Zurawski, "PerfSONAR: A service oriented architecture for multi-domain network monitoring," in *In Proceedings of the Third International Conference on Service Oriented Computing (ICSOC 2005)*, ser. ACM Sigsoft and Sigweb, December 2005, pp. 241–254.

[37] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek, "Askalon: a grid application development and computing environment," in *The 6th IEEE/ACM International Workshop on Grid Computing*, 2005.

[38] H. Truong and S. Dustdar, "Dynamic instrumentation, performance monitoring and analysis of grid scientific workflows," *Journal of Grid Computing*, vol. 3, no. 1-2, pp. 1–18, 2005.

[39] P. Brunner, H. Truong, and T. Fahringer, "Performance monitoring and visualization of grid scientific workflows in ASKALON," in *High Performance Computing and Communications*, ser. Lecture Notes in Computer Science, 2006, vol. 4208, pp. 170–179.

[40] S. Ostermann, K. Plankensteiner, R. Prodan, T. Fahringer, and A. Iosup, "Workflow monitoring and analysis tool for ASKALON," in *Grid and Services Evolution*, 2009.

[41] S. M. S. da Cruz, F. N. da Silva, L. M. R. G. Jr., M. C. R. Cavalcanti, M. L. M. Campos, and M. Mattoso, "A lightweight middleware monitor for distributed scientific workflows," in *IEEE International Symposium on Cluster Computing and the Grid*, 2008, pp. 693–698.

[42] L. Moreau *et al.*, "Special issue: The first provenance challenge," *Concurrency and Computation: Practice & Experience*, vol. 20, no. 5, 2008.

[43] Y. Simmhan, B. Plale, D. Gannon, and S. Marru, "Performance evaluation of the karma provenance framework for scientific workflows," in *In Proceedings of the International Provenance and Annotation Workshop (IPAW)*, 2006.

[44] I. Altintas, O. Barney, and E. Jaeger-frank, "Provenance collection support in the kepler scientific workflow system," in *In Proceedings of the International Provenance and Annotation Workshop (IPAW)*, 2006.

[45] L. Chen, V. Tan, F. Xu, A. Biller, P. Groth, S. Miles, J. Ibbotson, M. Luck, and L. Moreau, "A proof of concept: Provenance in a service oriented architecture," in *Proceedings of the UK OST e-Science Second All Hands Meeting*, 2005.

[46] S. Miles, E. Deelman, P. Groth, K. Vahi, G. Mehta, and L. Moreau, "Connecting scientific data to scientific experiments with provenance," in *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, 2007, pp. 179–186.

[47] S. Miles, P. Groth, E. Deelman, K. Vahi, G. Mehta, and L. Moreau, "Provenance: The bridge between experiments and data," *Computing in Science and Engineering*, vol. 10, no. 3, pp. 38–46, 2008.

[48] S. Fu and C.-Z. Xu, "Exploring event correlation for failure prediction in coalitions of clusters," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, ser. SC '07. New York, NY, USA: ACM, 2007, pp. 41:1–41:12. [Online]. Available: http://doi.acm.org/10.1145/1362622.1362678

[49] S. Chandrasekaran *et al.*, "TelegraphCQ: Continuous dataflow processing for an uncertain world," 2003.

[50] "Splunk." [Online]. Available: www.splunk.com

[51] D. Peng and F. Dabek, "Large-scale incremental processing using distributed transactions and notifications," in *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*, 2010.

[52] M. Munawar and P. Ward, "Adaptive monitoring in enterprise software systems," *SysML*, June 2006.