

An Overview of Quill: A Passive Operational Data Logging System for Condor

Jiansheng Huang Ameet Kini Erik Paulson Christine Reilly
Eric Robinson Srinath Shankar Lakshmikant Shrinivas David DeWitt
Jeffrey Naughton
{jhuang, akini, epaulson, chrisr, erobinso, srinath, pachu, dewitt, naughton}@cs.wisc.edu

July 18, 2007

1 Introduction

Condor, a system for high throughput computing, schedules user-submitted jobs on idle workstations and dedicated clusters of machines. In this paper we present Quill, an add-on to Condor that improves the data collection, retention, and query capabilities of Condor. A running Condor system generates a large amount of operational data about the machines and jobs in the system. While Condor stores information about current jobs and partial information about completed jobs in text logs that are distributed throughout the system, Quill stores this information in a central database. Quill also has the capabilities to gather and store additional information about machines, the job-machine matching process, and file transfers. By storing the data about a Condor pool in a central relational database, Quill provides fast and flexible access to the operational and history data gathered from a Condor pool.

The Quill project is motivated by three observations. The first is that a database is a more appropriate tool for data management than are the set of text logs and scripts currently used by Condor. A reason why Condor was not initially developed to use a database for its data management is relational databases were in their infancy when the Condor project began more than twenty years ago. Now that databases have matured, they are suitable for managing Condor data. The second observation motivating this project is that, in certain cases, Condor users experience performance problems due to the method that Condor uses to query its data. By moving the data storage and query processing to a relational database, Condor is free to perform its job scheduling functions. Our third motivation is that there is much information in a Condor pool that Condor does not capture. Quill captures information about machines, the job-machine matching process, and file transfers. This information can be utilized by users and administrators to better manage running jobs, obtain information about completed jobs, and manage and analyze the operation of the pool.

The purpose of this paper is twofold: to detail the design and functioning of Quill, and to provide instructions for users of Quill. We begin in Section 2 with a brief overview of Condor in order to familiarize the reader with the portions of the Condor system that are involved in Quill. In Section 3 we provide details about the internal design and functioning of Quill. Section 4 provides instructions for the installation and use of Quill.

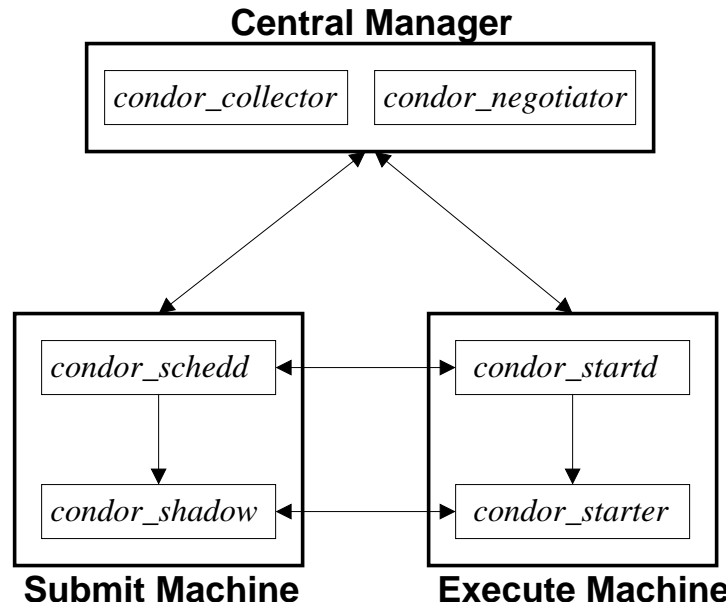


Figure 1: Entities of a Basic Condor Pool

2 A Brief Overview of Condor

Condor is a system for high throughput computing that schedules user-submitted jobs on idle workstations and dedicated clusters of machines. In this section we provide an overview of the portions of Condor that are important for understanding the following discussion of Quill. A thorough explanation of Condor can be found in the Condor Manual which is available on the Condor web site, <http://www.cs.wisc.edu/condor>.

A basic Condor pool consists of three types of machines: the central manager, submit machines, and execute machines. The pool has a single central manager machine, which is responsible for collecting information about the jobs from submit machines and the resources offered by execute machines. Users log into submit machines to submit jobs to the system. Execute machines offer their resources to run jobs in the system. These three entities of a basic Condor pool are illustrated in Figure 1. A single machine can act as both a submit and an execute machine. The central manager is usually run on a dedicated machine, but it is possible for a single machine to act as the central manager, a submit machine, and an execute machine.

Condor uses daemons and processes to perform the various tasks required by the system. The daemons and processes that relate to our discussion are shown in Figure 1 as the boxes within each type of machine. There are other daemons and processes in a Condor system but this discussion focuses only on those that are involved with Quill. A *condor_schedd* daemon on each submit machine accepts jobs from users and stores them in the job queue. The schedd sends advertisements for the jobs in its queue to the collector, and is responsible for claiming resources for its jobs. When the schedd claims a resource for a job it spawns a *condor_shadow* process to serve that job's request. The *condor_shadow* process is the resource manager for a job. A *condor_startd* daemon on each execute machine sends the collector advertisements for the machine resource and enforces the machine's policies for running jobs. When the resource is matched with and claimed by a

job, the `condor_startd` spawns a `condor_starter` process. The `condor_starter` process establishes the execution environment on the machine and monitors the job while it is running. The central manager consists of the `condor_collector` and `condor_negotiator` daemons. The `condor_collector` is responsible for collecting the information sent to it by the other daemons and serves requests from daemons for information specific to the pool. The `condor_negotiator` is responsible for running the match-making process that matches compatible jobs and resources.

The Condor system uses a mechanism called ClassAds to describe resources, jobs, and system information. A ClassAd is a set of uniquely named expressions where each expression has the form of an attribute-value pair. ClassAds are very flexible because they allow arbitrary expressions. Every execute machine in the pool advertises its resources by sending a ClassAd to the `condor_collector` daemon. The requirements of jobs are also sent to the `condor_collector` in the form of a ClassAd. The `condor_negotiator` daemon parses the job and resource ClassAds and matches jobs with corresponding resources. ClassAds are used by other processes in the system for communication and accounting purposes.

3 Internal Design of Quill

In this section we present the operational details of Quill and discuss the design challenges and decisions we faced. This section begins with a discussion of our design philosophy, much of which was shaped by the fact that Quill operates in cooperation with Condor. Then we discuss the three possible configurations of Quill. Next we detail the Quill and Dbmsd daemons, two new daemons that we added to Condor, and describe the other modifications we made to Condor. We then discuss our choice of Oracle and PostgreSQL as the database software options for Quill. Finally, we present the known limitations of Quill in the areas of scalability, concurrency, currency, possible data loss, and other issues.

3.1 Passive Data Collection

Our design philosophy is largely shaped by the fact that Quill must operate in cooperation with the existing Condor system. Ensuring that Quill not effect the normal (i.e., non-Quill) operation of Condor largely drove our overall design. We took this design approach because the Condor team requested that Condor continue to operate if the Quill daemon or the database is not working properly. If the database or Quill is not functioning, Condor will still run jobs and function as it does without Quill, but it will lose the data collection and querying capabilities provided by Quill. This passive, Condor-centric, approach requires us to follow the Condor method of recording operational data by having the daemons write data to log files. The Quill daemon periodically probes these log files and creates the SQL to insert and update data in the database. The details on the operation of the daemons added by Quill are discussed in Section 3.3.

3.2 Various Configurations of Quill

There are three configurations possible for Quill. Each configuration builds upon the previous by collecting more data from Condor. The Condor administrator can choose the configuration for a pool based on the amount of information desired from Quill and the available storage space. In this section we provide a general discussion of the various configurations. Details of how Quill gathers information is provided in Section 3.3.1. For all of these configurations, the `condor_q` utility is

altered. The second and third configuration also use an altered `condor_history` utility. We discuss the alterations to `condor_q` and `condor_history` in Section 3.3.3.

3.2.1 Configuration 1: Current Job Information

The default configuration for Quill is to only gather information about current jobs. In this case Quill only runs on submit machines. Users can obtain job information by using SQL to query the database, through a web interface to the database, or by using the `condor_q` utility. The `condor_q` utility is the only Condor tool that is modified in this configuration; it is directed to the database to answer its queries. Examples of queries that can be answered by this configuration of Quill are:

- What users currently have jobs in the pool?
- For a given user, what jobs are currently in the pool and what is the state of each job?

3.2.2 Configuration 2: Job History Information

When the gathering of job history information is enabled, Quill will gather historical job information in addition to current job information. As with Configuration 1, the only machines running Quill are submit machines. With this configuration the `condor_q` and `condor_history` utilities are altered to direct their queries to the database. This configuration can answer all of the queries that can be answered by Configuration 1. Some examples of queries added by this configuration are:

- What users have ever run jobs in the pool?
- For a given user, what jobs were run within a specified date range?

3.2.3 Configuration 3: Extended Information

When the gathering of extended information is enabled, Quill will gather information beyond that associated with jobs. This additional information includes machines, matches, rejects, and file transfers. The information gathered by Configurations 1 and 2 continues to be gathered in this configuration, and the `condor_q` and `condor_history` utilities are directed to the database to answer their queries. The extended information can only be accessed by using SQL to query the database or through a web interface to the database. Some examples of queries that this configuration adds are:

- What is the current state of the machines in the pool?
- What files were transferred by a specific job, and what were the directions of the transfers?
- For each machine, how many jobs finished successfully? How many jobs exited with an error?

3.3 How Quill works

Quill adds two daemons to Condor: the Quill daemon and the Dbmsd daemon. Additionally, Quill requires database software in order to store the data gathered by the Quill daemon. An illustration of a Condor pool with Quill is shown in Figure 2. The top part of Figure 2 shows a high level view of the pool. From this perspective we see that all the machines in the pool are sending data to the

Condor Pool with Quill

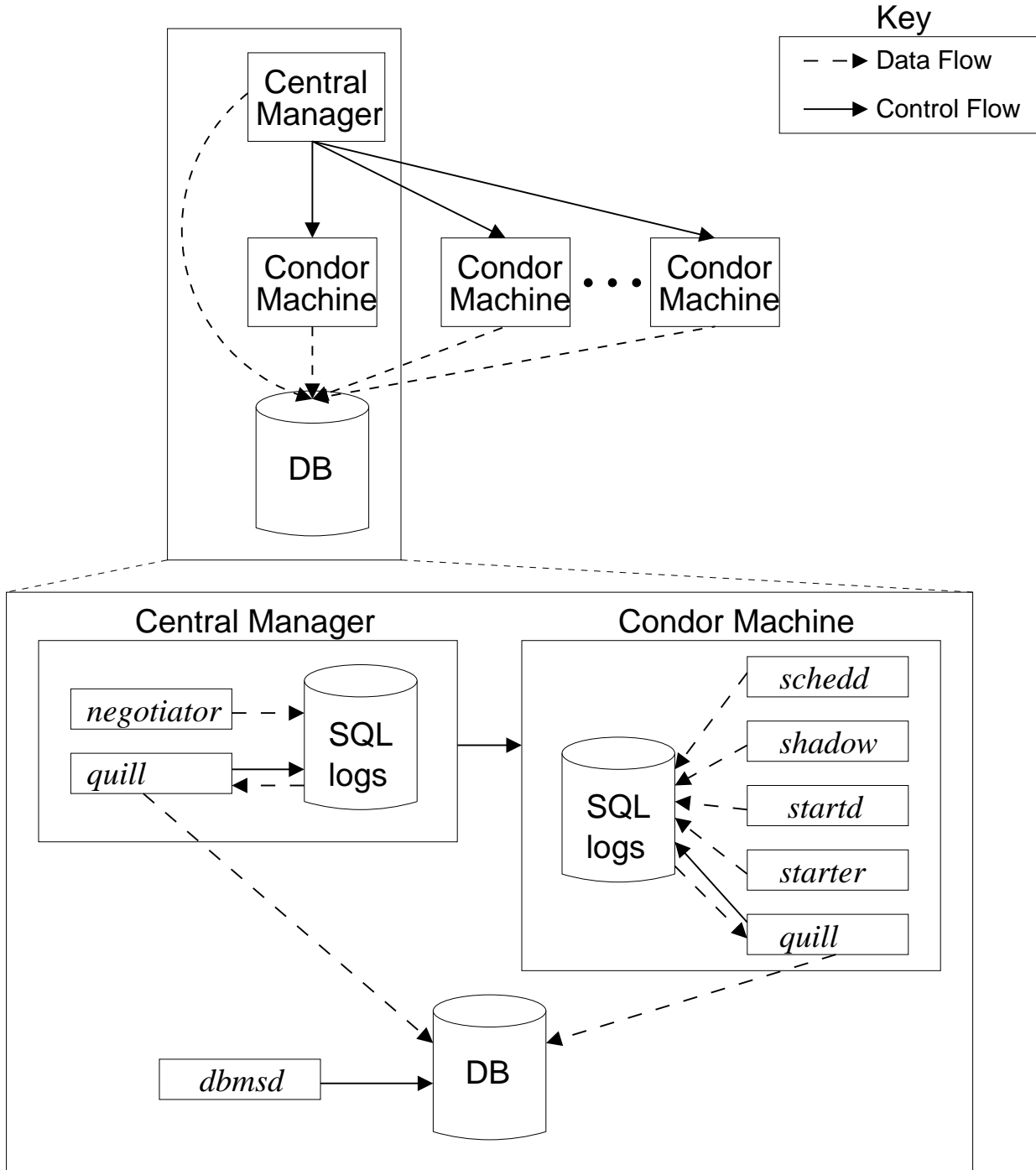


Figure 2: A Condor Pool with Quill

central database. The bottom part of Figure 2 is a more detailed view that focuses on the Central Manager machine and one typical Condor machine in the pool. From this zoomed in perspective we can see how Quill is working inside a machine.

In the following discussion we assume the Extended Information configuration of Quill (Section 3.2.3). In the case of the Current Job Information (Section 3.2.1) or Job History Information (Section 3.2.2) configurations, the Quill daemon will only reside on machines with a schedd and will only collect information from the schedd. With the Extended Information configuration, the Quill daemon resides on every machine in the pool and collects information from many of the daemons and processes.

3.3.1 How the Quill Daemon Gathers Information

The Quill daemon on each machine periodically wakes up and moves the data from the SQL log files to the database. The frequency of Quill runs is specified in the Condor configuration file. The data used by Quill is written to the SQL log files by Condor daemons and processes. The exact number and names of the SQL log files is configurable for each of the Condor daemons and processes that write Quill information. For example, all daemons and processes on a machine could write to a single Quill log, or each entity could have its own Quill log. The default configuration, which is the configuration assumed by this discussion, is that each machine has three additional log files: `sql.log`, `sql.log.copy`, and `thrown.log`. We altered the schedd daemon, startd daemon, negotiator daemon, shadow process, starter process, and file transfer mechanism to write information for Quill to the `sql.log` file. Note that any other Condor daemons or processes using the file transfer mechanism will also cause information to be written to the `sql.log` file. The `sql.log.copy` file is a copy of the `sql.log` file that is used to handle concurrency problems, as detailed in Section 3.6. The `thrown.log` file is used in the case that the `sql.log.copy` file exceeds its size limit, as detailed in Section 3.8.

The logging interface used by the Condor daemons is oblivious to the Quill database schema. This allows changes to be made to the schema without requiring corresponding changes to the Condor daemons. We achieve this schema independence by using the concept of an event as the unit of information. An event is a generic term that encompasses anything that happens in Condor. Some examples of events are: a job is run, there is an exception during a job run, a job is completed and moved to history, a match between a job and resource is made, and a machine reports its new state. By using this generic concept of an event, Quill is flexible enough to handle Condor events that were not predicted during the design of Quill. The Condor daemons write log records using one of two event functions. The `newEvent` function is used to log information about a new event. The `updateEvent` function is used to log updates to an existing event. The Quill daemon reads the event information from the log files, interprets the information, and inserts or updates the data in the database.

3.3.2 The Dbmsd Daemon

The Dbmsd resides on one machine in the pool, often the machine where the database is located. Periodically, the Dbmsd wakes up and connects to the database. The frequency with which the Dbmsd wakes up is set in the Condor configuration file. The Dbmsd performs two functions every time it runs: purge history information from the database, and estimate the size of the database. Additionally, if the PostgreSQL database is used, the Dbmsd recreates some of the indexes. The Dbmsd purges history information that is older than the threshold set in the Condor configuration

Comparison of condor_q in Traditional Condor and Condor with Quill

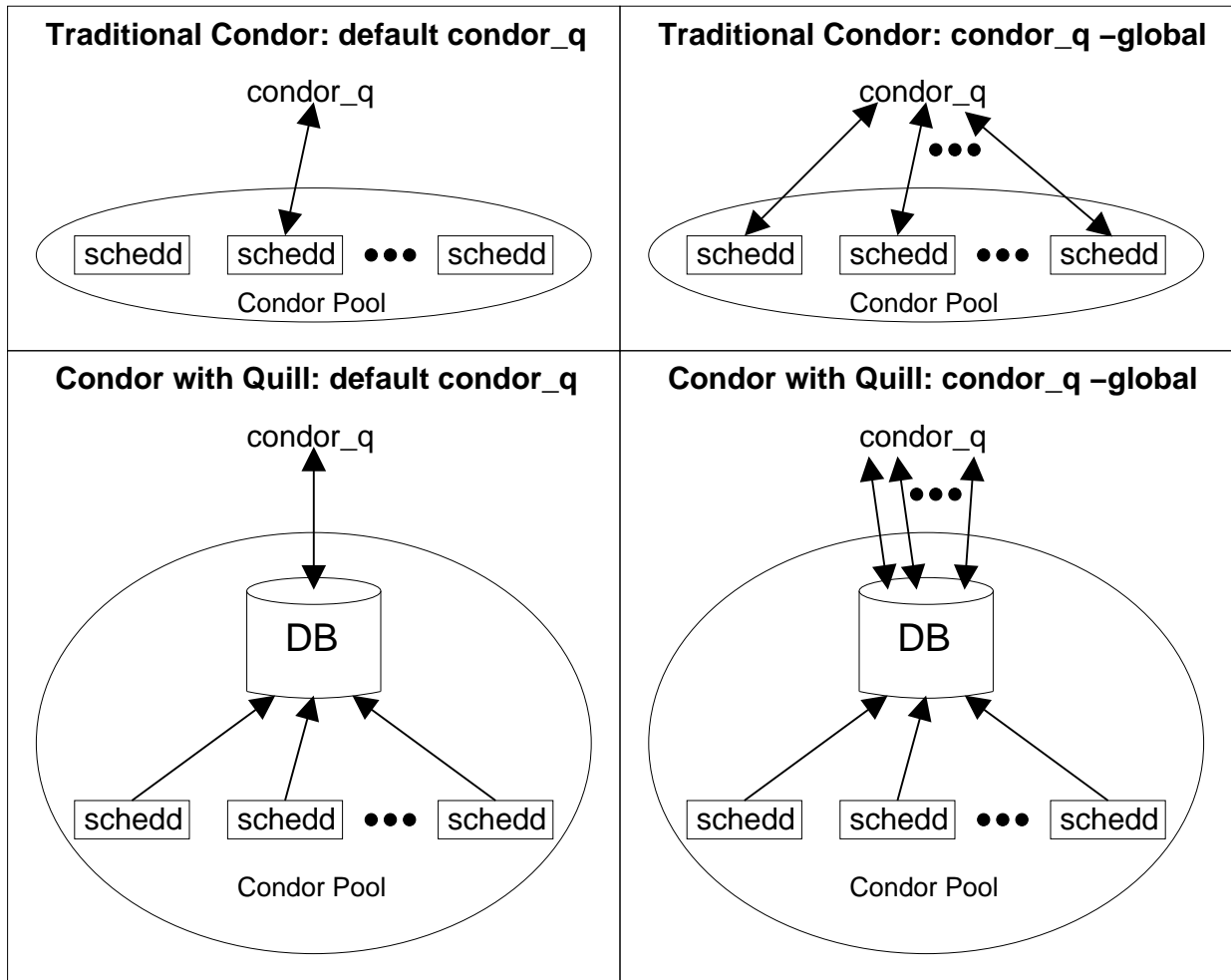


Figure 3: Operation of condor_q in Traditional and Quill Condor

file. The history tables are divided into three groups, with each group having its own purge threshold parameter. The Resource group consists of the information provided by the machine, master, and negotiator daemons. The Run Information group is composed of information on matches, rejects, and runs. The final group consists of information about job history. After the Dbmsd has truncated the tables, it estimates the size of the database by summing the sizes of all the tables. The Dbmsd emails the administrator if the database size exceeds the threshold set in the Condor configuration file. If the PostgreSQL database is used, the Dbmsd runs the PostgreSQL REINDEX command on tables where the operation is necessary.

3.3.3 Operation of condor_q and condor_history

In order to provide faster access to information and to alleviate the load on the schedd, Quill directs `condor_q` and `condor_history` to gather their information from the database. A comparison of how `condor_q` works in traditional Condor and in Condor with Quill is illustrated in Figure 3. In traditional Condor, these tools directly obtain the information from the appropriate log on the machine of the default or specified schedd. When the global option is specified with `condor_q`, the tool contacts all machines in the pool in order to gather their job queue information. With Quill, the job queue and job history information of each schedd is captured in the database. The `condor_q` and `condor_history` tools contact the database to obtain information. When the global option is specified with `condor_q`, the tool will send one query per schedd to the database.

3.4 Database Software

We currently support two database platforms for use with Quill: Oracle and PostgreSQL. The administrator of a Condor pool can decide which database to install for the pool based on the requirements and resources of the site. A different database platform could be used with Quill, as discussed at the end of this section.

Oracle is a commercial database platform. As detailed in the configuration instructions (Section 4.1.1), a number of parameters need to be properly set for Quill. These parameters involve cursor sharing, shared servers, and the memory pool size. It is especially important to have these parameters properly set when the Condor pool has more than 500 machines. Additionally, the administrator should monitor the instance performance and alert logs in order to detect any problems that may arise.

PostgreSQL is an open source database platform. Version 8.1 or later with the auto-vacuum feature enabled should be used with Quill. Previous versions of PostgreSQL presented problems with the database maintenance VACUUM procedure. These difficulties are alleviated by enabling the auto-vacuum feature found in Version 8.1 or later. Because Quill frequently updates the database, the VACUUM procedure is critical for maintaining good database performance. Prior to Version 8.1 the database administrator needed to determine when to vacuum each table. The auto-vacuum feature automatically determines when to vacuum each table and removes the burden of this maintenance procedure from the database administrator.

We have designed the Quill code such that it is possible to use a database other than Oracle or PostgreSQL. All of the database calls are wrapped in a virtual database class. Subclasses of the virtual database class implement the PostgreSQL and Oracle calls. In order to use a different database with Quill, a subclass of the virtual database class with the proper calls for that database must be created. Then the Quill daemon code must be altered to instantiate the proper database class.

3.5 Scalability

Quill faces two types of scalability challenges: first, the ability of the database to handle large clusters (more than a few hundred machines, depending on system resources); second, the problem of a constantly increasing amount of historical information if the system stores all information forever.

3.5.1 Scaling to Large Clusters

The challenges faced when scaling Quill up to large clusters are caused by an excessive number of connections and concurrent updates to the database. The methods for alleviating this problem depend on whether PostgreSQL or Oracle is used.

Because PostgreSQL does not support multithreaded servers, each connection from a Quill daemon requires a dedicated database background process for serving the SQL request. The high number of concurrent database processes utilized by a large cluster could cause thrashing on the database server. In order to alleviate this problem, we provide a Condor configuration parameter (`QUILL_MAINTAIN_DB_CONN`) that determines whether or not the Quill daemon maintains the database connection between two consecutive log processing calls. By setting this parameter to false, the database connection will not be maintained and the thrashing problem will be alleviated. Note that setting this parameter to false is only useful if the Condor pool is configured such that there is a low probability of many Quill processes concurrently updating the database. This proper configuration includes staggering the startup time of each node and setting the periodicity of Quill wake up such that few Quill processes have overlapping activity.

Oracle can support a large number of connections by using a fixed number of shared servers. However, the shared server mode raises an additional problem because session memory for each connection is allocated out of shared memory. In order to avoid using a large amount of shared memory, Quill must connect to and then disconnect from the database for each log processing call. This is achieved by setting the `QUILL_MAINTAIN_DB_CONN` parameter to false.

We used 100 physical machines to simulate a large Condor cluster that achieved stable performance. The cluster had 11 submit points, each running a schedd and a quill daemon. There were 950 execute nodes, with each node containing two 1 GHz Pentium3 processors and running a startd and a quill daemon. The central manager ran the collector, negotiator, dbmsd, and quill daemons. We ran Oracle on a Linux machine with four 3 GHz Xeon processors and 2 GB total memory, with 1 GB of memory used for Oracle shared memory. The database configuration parameters were set as shown in Section 4.1.1.

3.5.2 Continuously Increasing Historical Information

The second type of scalability issue is that we have a continuously increasing amount of historical information. Quill archives the data generated by Condor regarding job history, prior machine states, matchmaking, job runs, and files. Eventually the point will be reached where the amount of data exceeds the available storage. Also, the performance of certain queries may decrease as the size of tables in the database increases.

As discussed in Section 3.3.2, Quill provides a mechanism for truncating the tables that contain historical information. The history tables are divided into three groups, with each group having its own time period for truncation. The truncation time periods are set in the Condor configuration file. The amount of historical information stored in the database depends on the configuration of Quill used by the pool:

- Current Job Information (Section 3.2.1): No purging is necessary because no historical data is saved.
- Job History Information (Section 3.2.2): The database stores historical information about job classads so the job classad history purge interval should be set.

- Extended Information (Section 3.2.3): The database stores historical information about jobs, resources and runs. All three purge intervals (job classad history, resource classad history, and job run history) should be set.

In order to estimate the growth rate of the database, we submitted a large number of 20 minute jobs to our test pool. Our pool has 100 dual processor machines. We installed 10 startds on each execute machine to simulate a pool of 950 machines. Therefore, our simulation used 1900 virtual machines. The growth rates for the three categories of tables that store historical information are:

- Job classad history tables (jobs_horizontal_history, and jobs_vertical_history): 5 KB per job; we completed 46,842 jobs in one day, resulting in a growth rate of 224 MB per day.
- Resource classad history tables (daemons_horizontal_history, daemons_vertical_history, machines_horizontal_history, machines_vertical_history, maintenance_log): 823 MB per day.
- Job run history tables (events, files, fileusages, matches, rejects, runs, and transfers): 128 MB per day.

Based on the default time periods for purging the tables, the Quill historical data for Configuration 3 requires 824 gigabytes. The default time periods for purging the tables are: ten years for job classad history tables; one week for resource classad history tables; and one week for job run history tables. It should be noted that the maximum size for the variable size attributes is much longer than the actual size of these attributes in our pool. The tables could rapidly grow large if the variable size attributes frequently hold very long values.

We have started to explore other strategies for compacting the tables. One alternative to deleting data is to move older data to an archive, but that raises the question of how the archived data would be involved in queries. Another approach we have examined are compression strategies that “roll-up” the data by aggregation. This roll-up strategy also raises the question of how to query the compressed data. We have also observed that for certain types of Condor data we can reduce the storage requirements by recognizing the meaning of data and storing it appropriately. For example, the machines in a Condor pool periodically report their current state, including attributes such as processor type, amount of memory, and operating system. Instead of storing the entirety of every machine report we can store the attribute name and value with the range of time where that value is valid. This approach will reduce the storage requirements because the value for an attribute such as operating system is only stored once per machine instead of being stored every time the machine reports its state. We plan to continue to examine these strategies as part of our future work.

3.6 Concurrency

Quill handles the two concurrency issues regarding its log files: the first issue is concurrent writes to the sql.log when the daemons share a log; the second issue is concurrent Condor daemon writing and Quill daemon reading of the sql.log file.

The first concurrency issue is solved by having the Condor daemons obtain a write lock before writing to the sql.log file. In most cases the write lock solution is adequate because the individual daemons do not hold the write lock for very long. Two cases where it may be beneficial for daemons to write to separate log files are: when many daemons are running on one machine; or when a daemon, such as a very active schedd, frequently writes a lot of data.

We address the second concurrency issue by preventing a long Quill read operation from blocking the Condor daemon write operations. The Quill daemon obtains a write lock the `sql.log` file, copies `sql.log` to `sql.log.copy`, truncates the `sql.log` file, and releases the write lock. Then the Quill daemon obtains a read lock on the `sql.log.copy` file, reads the file, truncates the file, and releases the read lock. We expect the copy and truncate operation on the `sql.log` file to block Condor daemons for an acceptable amount of time.

3.7 Currency

Because Quill is a passive part of Condor, the Quill database may lag behind the current state of the Condor pool. For example, in our installation it can take a few minutes for information about newly submitted jobs to reach the database. The first currency issue is that Condor must write the information to the Quill log file before Quill is able to obtain that information. For example, the schedd takes some time to write to the log when a large number of jobs are submitted at once. Second, currency of data in Quill depends both on how frequently the Quill daemon runs and the size of the current update. We address the second currency issue by recording the most recent time that each Quill daemon reported to the database. This data can be queried to provide information about the currency of the information from various data sources.

3.8 Possibilities for Data Loss

The Quill SQL log files have a maximum size of 2 gigabytes in order to avoid the overflow of a file in any platform supported by Condor. Because these logs have a limited size, there is a slight possibility that Quill will miss data generated by Condor. Data loss can occur in two situations: if the Quill daemon is not operating properly or if the Quill daemon is unable to connect to the database.

In the unlikely scenario that the Quill daemon is down for a long time, the SQL log files could exceed the size limit because the Quill daemon is not available to truncate the logs. The Condor daemons will not write to the log files when these files exceed the size limit. The data generated by the Condor daemons after the log files reach the size limit will be lost.

Another unlikely scenario is that the Quill daemon is unable to connect to the database for a long time. In this case, the copies of log files (e.g., `sql.log.copy`) could exceed the size limit if the Quill daemon does not move the data to the database and truncate the file frequently enough. When the log files exceed the size limit, the Quill daemon truncates the files and inserts a record to `thrown.log` with the name and size of each log that was truncated and the timestamp of when the log was truncated. We assume that before `thrown.log` exceeds its size limit the problem with the database connection will be resolved or at least that the system administrator will notice that there are new entries in `thrown.log`.

In order to estimate the growth rate of the Quill logs, we ran our test pool for one day without connecting to the database. The largest log was 53 MB after one day. Given the maximum log size of 2 gigabytes, this log could grow for 37 days without losing any information. The pool administrator will most likely realize the problem with the Quill daemon or database connection long before any data is lost. The details of our log growth test are below.

Our test pool has 100 dual processor machines. We installed 10 startds on each execute machine to simulate a pool of 950 machines. Therefore, our simulation used 1900 virtual machines. We submitted a large number of 20 minute jobs to our test pool and ran the pool for one day without

connecting to the database. Each scheduler machine used two logs. One log was dedicated to the schedd which logs information about the schedd ads, files, file usages, and job history. Of our 10 schedulers, the largest schedd log was 53 MB. This log could grow for 37 days before reaching the maximum size. The other log on the scheduler machines is the log for all other daemons on those machines and contains information about the master ad, job runs, and file transfers. Of our 10 schedulers, the largest log was 24 MB. This log could grow for 83 days before reaching the maximum size. Our pool used a dedicated central manager machine that ran the master, collector, and negotiator daemons. This machine used one log containing information about the master ad, negotiator ad, matches, and rejects. After one day, the size of this log was 4 MB. This log could grow for 500 days before reaching the maximum size. The single log on the execute machines contains information about the startd and master ads. On one of our execute machines the log was 2 MB after one day. This log could grow for 1000 days before reaching the maximum size.

3.9 Additional Issues

Condor has a number of special features and configurations that Quill does not specifically gather data about. Some of these configurations may result in odd or incomplete information in the Quill database. It should be noted that the Quill daemon will continue to operate properly in these cases. For example, Condor has a feature called flocking where the jobs from Condor Pool A are run on machines in Condor Pool B. Flocking results in incomplete information in the Quill database. The database for Pool A will not have information about the machines in Pool B, and the database for Pool B will not have information about the jobs that flocked from Pool A. The Condor Glide-In feature may also result in the database containing incomplete information. We have not implemented any special handling for DAGMan applications, but we have written queries that reconstruct a DAG from the job history information stored by Quill. Also, we have not tested jobs in the Condor universes that allow special functionality, such as PVI, MPI, and Globus. It might be possible to write queries that gather some information about the special functionalities provided by these universes, but we have not explored such queries.

3.10 Quill before Condor version 6.9.3

A different implementation of Quill was used prior to Condor version 6.9.3. The old version of Quill was completely non-invasive to Condor and the information it could obtain was limited to job queue and job history information. Additionally each schedd used its own database. The Quill daemon sniffed the job queue log and inserted information about current jobs into the database. History information was inferred from the job queue information. As with the current version, the `condor_q` and `condor_history` utilities were directed to obtain information from the database. The old version of Quill had a higher potential to lose historical information than the current version.

As is detailed in Section 4.2, it is possible to migrate history information from the old version of Quill to the current version of Quill. It should be noted that the first step of data migration must be preformed before upgrading to the new version of Condor.

4 User Guide

Quill is intended to be a replacement for and extension of the job and machine monitoring tools provided by Condor, such as `condor_q`, `condor_analyze`, `condor_status`, and `condor_stats`. A user

can use Quill to monitor the progress of jobs in Condor and retrieve information about historical jobs. The system administrator can use Quill for examining the current state of the Condor pool and for retrieving history information to use in system debugging and accounting applications. The Quill database can also be leveraged to provide periodic reports about a Condor pool to the system administrator. Because Quill stores the Condor data in a relational database, users and administrators can easily write SQL queries to retrieve the information they desire. In this section we discuss how to install and configure Quill and the software it requires. We then present the relational schema of the Quill database.

4.1 Installation and Configuration of Quill

In order to use Quill, a Condor administrator must install a release of Condor that includes Quill, install a database system, and configure Condor and the database system for Quill. As discussed in Section 3.4, we support Oracle or PostgreSQL as the database system for Quill. In this section we describe how to configure each of the database systems and how to configure Condor for Quill.

4.1.1 Configuring Oracle

For best performance we recommend using different physical disks for the redo log, undo log, temporary table space, and data files. Additionally we do not recommend using RAID5 for the Quill database as it has a negative impact on performance.

Follow the steps below to configure Oracle for Quill.

1. Start the Oracle server.
2. Create a user named quillwriter with a password.
3. Create a user named quillreader with a password.
4. Start up the listener process so the database can be accessed remotely.
5. Set up the database schema:
 - Start the sqlplus program as user quillwriter.
 - Install the schema common to all database types using the following command:
@/path/to/common_createddl.sql
 - Install the schema that is specific to Oracle using the following command:
@/path/to/oracle_createddl.sql
6. Set the following Oracle parameters:
 - Some SQL statements in Quill do not use bind variables. In order to improve cursor sharing and therefore improve system scalability, set the cursor sharing parameter to similar:
cursor_sharing=SIMILAR

- Auto-memory management does not work perfectly in Oracle 10. If an Oracle error related to memory shortage is encountered in a shared or large Condor pool, the Oracle pool size parameters should be increased. The current size of these parameters can be found in the v\$sgainfo table. For example:


```
shared_pool_size=300M
large_pool_size=200M
```
- If Quill is deployed to more than 500 nodes we advise using Oracle's shared server. For example:


```
shared_servers=100
max_shared_servers=100
```
- The following parameters should also be set. The values below are the settings used in our test cluster.


```
dispatchers='(PROTOCOL=TCP)(DISPATCHERS=50)'
sga_target=1GB
sessions=1500
```

4.1.2 Configuring PostgreSQL

As detailed in Section 3.4 we strongly recommend using PostgreSQL Version 8.1 or later because of the auto-vacuum feature. Follow the steps below to configure PostgreSQL Version 8.1 or later for Quill.

1. Configure to accept TCP/IP connections. For PostgreSQL version 8, use the listen_addresses variable in postgresql.conf file as a guide. For example, listen_addresses = '*' means listen on any IP interface.
2. Configure automatic vacuuming. Ensure that these variables with these defaults are commented in and/or set properly in the postgresql.conf configuration file:
 - Turn on/off automatic vacuuming:


```
autovacuum = on
```
 - Time between autovacuum runs, in seconds:


```
autovacuum_naptime = 60
```
 - Minimum number of tuple updates before vacuum:


```
autovacuum_vacuum_threshold = 1000
```
 - Minimum number of tuple updates before analyze:


```
autovacuum_analyze_threshold = 500
```
 - Fraction of relation size to add to autovacuum_vacuum_threshold:


```
autovacuum_vacuum_scale_factor = 0.4
```
 - Fraction of relation size to add to autovacuum_analyze_threshold:


```
autovacuum_analyze_scale_factor = 0.2
```
 - Setting the autovacuum cost delay to -1 means the vacuum_cost_delay value will be used:


```
autovacuum_vacuum_cost_delay = -1
```
 - Setting the autovacuum cost limit to -1 means the vacuum_cost_limit will be used:


```
autovacuum_vacuum_cost_limit = -1
```

3. Start the PostgreSQL server.
4. Create a user named quillwriter with a password. Allow quillwriter to create databases.
5. Create a user named quillreader with a password.
6. Create a database for Quill with quillwriter as the owner.
7. Set the CIDR-ADDRESS so that the Condor machines can connect to the database using password authentication. To do this, add the following line to the pg_hba.conf file (usually located in the data directory within the directory where PostgreSQL is installed).

Format of line:

```
TYPE DATABASE USER CIDR-ADDRESS METHOD
```

For example:

```
host quilldb all 128.105.0.0 255.255.0.0 md5
```

8. Set up the database schema:
 - If the pl/pgsql language has not been installed in the Quill database, then issue the following command:


```
createlang plpgsql [database name]
```
 - Start the psql program with the Quill database as user quillwriter.
 - Install the schema common to all database types using the following command:


```
\i /path/to/common_createddl.sql
```
 - Install the schema specific to PostgreSQL using the following command:


```
\i /path/to/pgsql_createddl.sql
```

4.1.3 Configuring Condor

The following parameters in the Condor configuration file must be set in order to run Quill.

1. Enable Quill. In order for the Quill or Dbmsd daemons to run they must be listed in the DAEMON_LIST parameter (Step 3, below):


```
QUILL_ENABLED = TRUE
```
2. For machines running the Quill daemon, specify the locations for the daemons to write their Quill logs. By default all daemons write to the sql.log file. If many daemons are running in the same installation, specify one log per daemon in order to improve performance. Each daemon can be configured to write to its own file by setting the [DAEMON NAME]_SQLLOG parameter to the log file. For example:


```
SCHEDD_SQLLOG = $(LOG)/schedd_sql.log
```
3. Add Quill and/or Dbmsd to the daemon list. The Quill daemon must be listed on all machines from which data is to be collected. For the Job History Information (Section 3.2.2) and Extended Information (Section 3.2.3) configurations of Quill, the Dbmsd daemon must be run on one machine in the pool (often the machine where the database resides). For example:


```
DAEMON_LIST = MASTER, SCHEDD, QUILL, DBMSD
```

4. Specify the locations of the binaries for the Quill and/or Dbmsd daemons. Due to the Oracle library issue we currently start these daemons with wrappers that first add the path to the Oracle client library to LD_LIBRARY_PATH then start the daemon. For example:


```
DBMSD = $(SBIN)/condor_dbmsd-wrapper
QUILL = $(SBIN)/condor_quillpp-wrapper
```
5. On each machine where the Quill daemon is running, specify the desired level of Quill:
 - For the Current Job Information configuration (Section 3.2.1), disable all SQL logs:


```
QUILL_USE_SQL_LOG=false
```
 - For the Job History Information configuration (Section 3.2.2), enable the SQL log on the schedd:


```
QUILL_USE_SQL_LOG=false
SCHEDD.QUILL_USE_SQL_LOG=true
```
 - For the Extended Information configuration (Section 3.2.3), enable the SQL logs on all daemons:


```
QUILL_USE_SQL_LOG=true
```
 - For a custom configuration, enable the SQL logs only on the desired daemons. The [DAEMON NAME].QUILL_USE_SQL_LOG parameter has priority over the QUILL_USE_SQL_LOG parameter. For example, to gather only job history and machine history, enable the SQL logs on the schedd and startd:


```
QUILL_USE_SQL_LOG=false
SCHEDD.QUILL_USE_SQL_LOG=true
STARTD.QUILL_USE_SQL_LOG=true
```
6. For machines running the Quill daemon, specify the number of seconds to wait for the Quill daemon to reply before it is killed by the master. This parameter needs to be set to a large number so that Quill will not be killed while it is processing a long SQL log. For example:


```
QUILL_NOT_RESPONDING_TIMEOUT = 9000
```
7. For machines running the Quill daemon, specify the number of seconds between two log polling calls. For example:


```
QUILL_POLLING_PERIOD=10
```
8. Set the database access information:


```
QUILL_DB_USER = quillwriter
QUILL_DB_NAME = [database name]
QUILL_DB_IP_ADDR = [machine address:port number]
QUILL_DB_QUERY_PASSWORD = [password for quillreader]
QUILL_DB_TYPE = [PGSQL or ORACLE]
```
9. If the Dbmsd daemon is running on the machine, set its parameters:
 - Interval between consecutive purging calls in seconds:


```
DATABASE_PURGE_INTERVAL = 86400
```

- Interval between consecutive database reindexing operations in seconds (for PostgreSQL only):
`DATABASE_REINDEX_INTERVAL = 86400`
 - Number of days before purging resource classad history (resource history includes machine and daemon classads, the maintenance_log, and submitters_horizontal_history):
`QUILL_RESOURCE_HISTORY_DURATION = 7`
 - Number of days before purging job run history (run history includes runs, events, matches, rejects, files, fileusages, and transfers):
`QUILL_RUN_HISTORY_DURATION = 7`
 - Number of days before purging job classad history (job history includes jobs_vertical_history, jobs_horizontal_history, throws, and error_sqllogs):
`QUILL_JOB_HISTORY_DURATION = 3650`
 - After every purge the Dbmsd checks the size of the database. If the database size exceeds the value set by this parameter, an email will be sent to the pool administrator. Size limit in gigabytes for the Quill database:
`QUILL_DB_SIZE_LIMIT = 20`
 - Number of seconds the master waits for Dbmsd daemon to reply before killing it:
`DBMSD_NOT_RESPONDING_TIMEOUT = 9000`
10. This parameter specifies the file where the schedd writes its name. Quill reads this file to obtain the schedd name:
`SCHEDD_DAEMON_AD_FILE = $(LOG)/.schedd_classad`
 11. The name of the Quill daemon:
`QUILL_NAME = [name of quill daemon]`
 12. For pools with more than 100 machines, this parameter should be set to false in order to minimize the number of concurrent connections at any point in time. Note that the default value is true.
`QUILL_MAINTAIN_DB_CONN = false`
 13. Add the file .pg_pass (discussed in the next section) to the VALID_SPOOL_FILES variable since condor_preen must be told to not delete this file.

4.1.4 System Configuration

Configure the system for Quill by following the steps below.

1. Add the path to the Oracle client libraries to LD_LIBRARY_PATH, for example:
`setenv LD_LIBRARY_PATH /path/to/OracleClientLibraries`
2. Create a file named .pgpass in the spool directory located in the directory where Condor is installed. This file specifies the location of the Quill database and the password for quillwriter in the following format:
`[machine address]:[port number]:[database name]:quillwriter:[password]`
For example:
`machine.cs.wisc.edu:1521:XE:quillwriter:password`

4.2 Migrating Data From Quill before Condor version 6.9.3

The process of migrating history information from the older version of Quill to the current version involves dumping the data from the old database to a file, then loading the file into the new database. Note that the dump process must be performed before upgrading to Condor version 6.9.3 or later. We provide tools for both the dump and load processes: the `condor_dump_history` tool and the `condor_load_history` tool, respectively. The `condor_dump_history` tool reads the history information from the old Quill database and writes the information to standard output in the Condor history format.

The `condor_load_history` tool reads a Condor history file and inserts the information into the current version of the Quill database. Any Condor history file can be passed to the `condor_load_history` tool, not only those files created by the `condor_dump_history` tool.

The steps for migrating data are:

1. On the old version of Condor (prior to version 6.9.3), run the `condor_dump_history` tool and save its output to a file. Note that problems will arise if the amount of data dumped exceeds the maximum file size of the operating system. An example of dumping history data is:

```
condor_dump_history -name [quill-name] > [quill-file]
```
2. Upgrade to Condor version 6.9.3 or later, install database software and configure the database for Quill.
3. Run the `condor_load_history` tool, specifying the file where the history information was saved. Optionally, also specify the schedd name and jobqueue birthdate so that these database fields will have the proper values. For example:

```
condor_load_history [quill-file] -name [schedd-name] [jobqueue-birthdate]
```

4.3 Schema

The Quill schema consists of nine categories of tables: jobs, machines, files, matchmaking, daemon classads, runtime, administrative, system, and web interface. Figure 4 shows the categories of tables in the Quill schema, including the tables within each category and the connections between various categories.

A major challenge we faced in developing a relational schema to represent Condor is that, by design, the Condor ClassAd mechanism has a flexible schema. We addressed the challenge of a flexible schema by splitting the table into two tables, one with a horizontal schema and one with a vertical schema. A horizontal schema is the standard schema used by relational databases. It has one row per entity with many attributes per row. In a vertical schema, an entity is split into multiple rows with one row per attribute. Each row of a vertical schema consists of the entity identifier, the attribute name, and the value for that attribute. We use the horizontal schema for the attributes that are frequently occurring. The vertical table is used for infrequently occurring attributes and those attributes that are created by Condor users or administrators. In Figure 4 we represent the horizontal and vertical pairs as a single table, but in the following discussion we will indicate which tables are composed of a pair of horizontal and vertical tables. Appendix A contains the entire schema of the Quill database.

The set of tables that contain information about jobs are: `clusterads`, `procads`, and `jobs_history`. The `clusterads`, `procads`, and `jobs_history` tables are implemented as a horizontal and vertical

Overview of Quill Schema

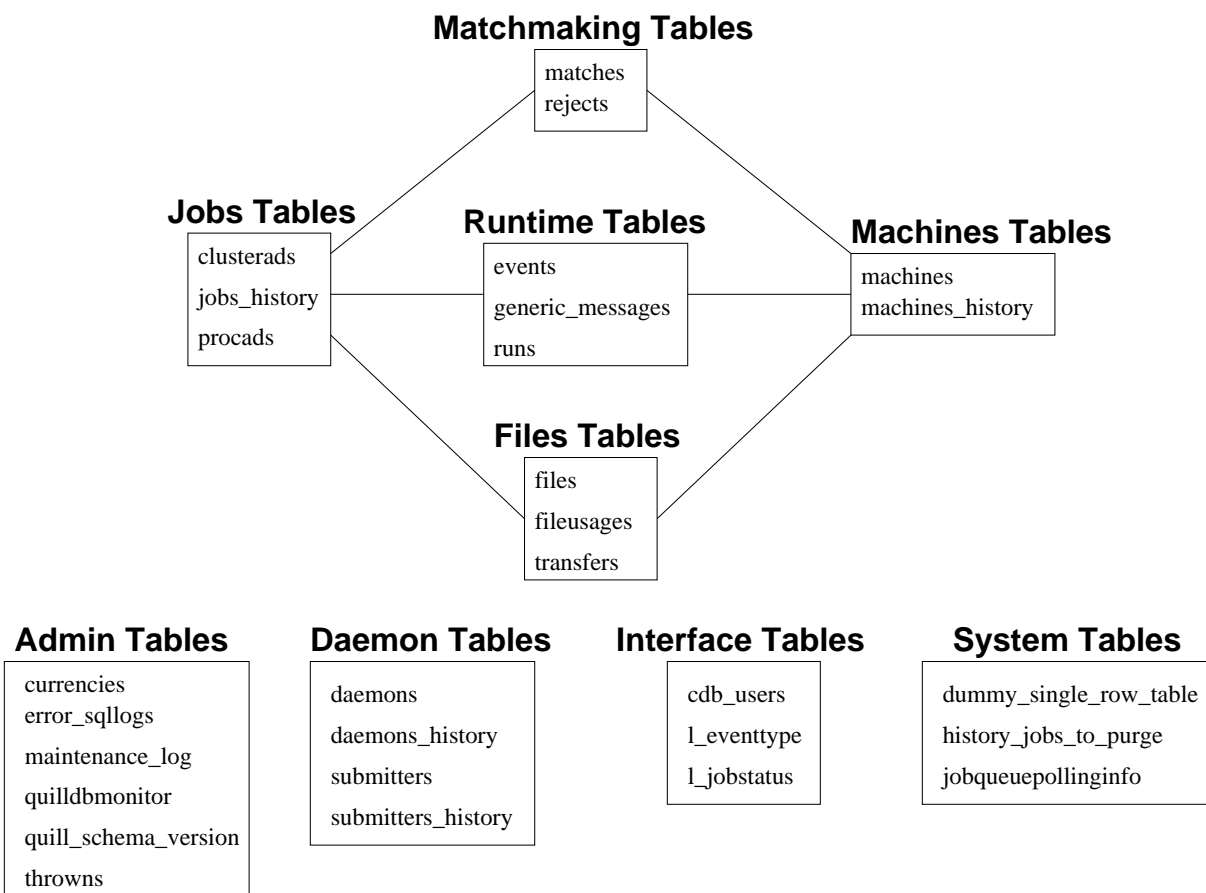


Figure 4: Tables in the Quill Schema

pair because they store data from flexible ClassAds. The clusterads and procads tables contain information about jobs that are currently in the system. The current job information is separated into these two tables because we are following the Condor model of having clusters of similar jobs with each individual job represented as a procedure (proc) within the cluster. The jobs_history table stores information about jobs that have exited the system.

The set of tables that contain information about machines are machines and machines_history. Both tables are implemented as a horizontal and vertical pair. The machines table contains information about the current state of machines in the pool. The machines_history table contains information about the state of machines in the pool prior to the current state.

The set of tables that contain file information are: files, fileusages, and transfers. The files table contains information about individual files that were used by jobs in the Condor pool. The fileusages table associates individual files with the job that used the file. The transfers table contains information about file transfers that occurred in the Condor pool.

The set of tables that contain information about matchmaking are matches and rejects. The matches table shows the job-machine matches that were made in each matchmaking cycle. The rejects table contains information about jobs that were rejected during a matchmaking cycle.

The set of tables that contain information about daemon classads are daemons, daemons_history, submitters, and submitters_history. The daemons and daemons_history tables are implemented as a horizontal and vertical pair. The daemons table contains the most recent status of the daemons that report to Quill. The daemons_history table holds the previous information about the daemons. The submitters table contains information from the schedd and submitter Condor ClassAds, including information about submitters' job status. The submitters_history table contains previous information about submitters. The submitters and submitters_history tables are included in the schema for future extensibility but are not currently populated by Quill.

The set of tables that contain runtime information are runs, events, and generic_messages. The runs table contains information regarding on which machine each job ran. The events table lists information about various events that occurred during a job run in the system. The generic_messages table allows for the Condor team to add new types of information to the database without having to change the schema.

The set of tables that contain database administration information are currencies, error_sqllogs, maintenance_log, quilldbmonitor, quill_schema_version, and throws. The currencies table lists the time each machine last reported to the database. The error_sqllogs table contains the SQL errors that Quill encounters while it is reading the SQL logs and making SQL updates to the database. The maintenance_log table consists of records about database maintenance events. The quilldbmonitor table holds the current size of the database in megabytes, as reported by the Dbmsd. The quill_schema_version table contains the current version number of the Quill schema. The throws table records information about log files that were deleted because they exceeded the maximum size.

The system tables are used internally by Quill to manage the database system. The set of tables that contain system information are dummy_single_row_table, history_jobs_to_purge, and jobqueuepollinginfo. The dummy_single_row_table allows certain system queries to have the same syntax in both Oracle and PostgreSQL. The history_jobs_to_purge table is a temporary table that is used while the system is purging data from the database. The jobqueuepollinginfo table is used by the Quill daemon to keep track of the job queue polling process.

The web interface tables are used by the user interface that we have developed (as discussed in

Section 4.4). This set of tables includes `cdb_users`, `l_eventtype`, and `l_jobstatus`. The `cdb_users` table contains information about the users who have access to the interface. The `l_eventtype` translates from the integer event code used by Condor to the text description of the event type represented by that code. The `l_jobstatus` table translates from the integer job status code to the text description of the job status represented by that code.

4.4 Example of a User Interface

We have developed a web-based user interface to the Quill database in order to demonstrate how the information collected by Quill can be easily accessed by Condor users and administrators. Our interface provides privacy by requiring users to log in and only allowing a user to see his/her own jobs. Only those users designated as administrators are allowed to view information about other users' jobs.

Our interface consists of two main sections: user information and administrator information. The user section allows users to view information about their jobs that are currently in the system and their jobs that have exited the system. Additionally, users can access the provenance subsection to obtain information about files and machines that were associated with a job run, as well as information about jobs that were part of a DAGMan job. The administrator section provides information about the machines in the pool, users and their jobs, and the recency of the Quill data sources.

5 Conclusions

This document discusses the technical details of Quill, a Condor add-on that centralizes Condor operational data into a relational database. Quill improves the data availability and query performance of Condor operational data. It also allows flexible querying and analysis over the data using SQL. By combining Quill with a well designed and intuitive user interface, Condor users and administrators have a powerful and easy method with which to view Condor operational data.

6 Acknowledgments

This work was supported in part by National Science Foundation Award SCI-0515491.

A Quill Schema

Notes:

- The type “timestamp(*precision*) with timezone” is abbreviated “ts(*precision*) w tz.”
- The column O. Type is an abbreviation for Oracle Type.
- The column P. Type is an abbreviation for PostgreSQL Type.

A.1 Administrative Tables

Attributes of currencies Table			
Name	O. Type	P. Type	Description
datasource	varchar(4000)	varchar(4000)	Identifier of the data source.
lastupdate	ts(3) w tz	ts(3) w tz	Time of the last update sent to the database from the data source.
INDEX: Index named currencies_idx on datasource			

Attributes of error_sqllogs Table			
Name	O. Type	P. Type	Description
logname	varchar(100)	varchar(100)	Name of the SQL log file causing a SQL error.
host	varchar(50)	varchar(50)	The host where the SQL log resides.
lastmodified	ts(3) w tz	ts(3) w tz	The last modified time of the SQL log.
errorsql	varchar(4000)	text	The SQL statement causing an error.
logbody	clob	text	The body of the SQL log.
errormessage	varchar(4000)	varchar(4000)	The description of the error.
INDEX: Index named error_sqllog_idx on (logname, host, lastmodified)			

Attributes of maintenance_events Table			
Name	O. Type	P. Type	Description
id	integer	integer	Identifier of the event.
msg	varchar(4000)	varchar(4000)	Message describing the event.
PRIMARY KEY: id			

Attributes of maintenance_log Table			
Name	O. Type	P. Type	Description
eventid	integer	integer	Identifier of the event.
eventts	ts(3) w tz	ts(3) w tz	Time the event occurred.
eventdur	interval day to second	interval	Duration of the event.
REFERENCE: eventid references maintenance_events(id)			

Attributes of quilldbmonitor Table			
Name	O. Type	P. Type	Description
dbsize	integer	integer	Size of the database in megabytes.

Attributes of quill_schema_version Table			
Name	O. Type	P. Type	Description
major	int	int	Major version number.
minor	int	int	Minor version number.
back_to_major	int	int	The major number of the old version this version is compatible to.
back_to_minor	int	int	The minor number of the old version this version is compatible to.

Attributes of throws Table			
Name	O. Type	P. Type	Description
filename	varchar(4000)	varchar(4000)	The name of the log that was truncated.
machine_id	varchar(4000)	varchar(4000)	The machine where the truncated log resides.
log_size	numeric(38)	numeric(38)	The size of the truncated log.
throwtime	ts(3) w tz	ts(3) w tz	The time when the truncation occurred.

A.2 Daemon Tables

Attributes of daemons_horizontal Table			
Name	O. Type	P. Type	Description
mytype	varchar(100)	varchar(100)	The type of daemon ClassAd, e.g. "Master"
name	varchar(500)	varchar(500)	The name identifier of the daemon ClassAd.
lastreportedtime	ts(3) w tz	ts(3) w tz	Time when the daemon last reported to Quill.
monitorselftime	ts(3) w tz	ts(3) w tz	The time when the daemon last collected information about itself.
monitorselfcpuusage	numeric(38)	numeric(38)	The amount of CPU this daemon has used.
monitorselfimagesize	numeric(38)	numeric(38)	The amount of virtual memory this daemon has used.
monitorselfresidentsetsizesize	numeric(38)	numeric(38)	The amount of physical memory this daemon has used.
monitorselfage	integer	integer	How long the daemon has been running.
updatesequencenumber	integer	integer	The sequence number associated with the update.
updatestotal	integer	integer	The number of updates received from the daemon.
updatessequenced	integer	integer	The number of updates that were in order.
updateslost	integer	integer	The number of updates that were lost.
updateshistory	varchar(4000)	varchar(4000)	Bitmask of the last 32 updates.
lastreportedtime_epoch	integer	integer	The equivalent epoch time of last heard from.
PRIMARY KEY: (mytype, name)			
NOT NULL: mytype and name cannot be null			

Attributes of daemons_horizontal_history Table			
Name	O. Type	P. Type	Description
mytype	varchar(100)	varchar(100)	The type of daemon ClassAd, e.g. "Master"
name	varchar(500)	varchar(500)	The name identifier of the daemon ClassAd.
lastreportedtime	ts(3) w tz	ts(3) w tz	Time when the daemon last reported to Quill.
monitorselftime	ts(3) w tz	ts(3) w tz	The time when the daemon last collected information about itself.
monitorselfcpuusage	numeric(38)	numeric(38)	The amount of CPU this daemon has used.
monitorselfimagesize	numeric(38)	numeric(38)	The amount of virtual memory this daemon has used.
monitorselfresidentsetsize	numeric(38)	numeric(38)	The amount of physical memory this daemon has used.
monitorselfage	integer	integer	How long the daemon has been running.
updatesequencenumber	integer	integer	The sequence number associated with the update.
updatestotal	integer	integer	The number of updates received from the daemon.
updatessequenced	integer	integer	The number of updates that were in order.
updateslost	integer	integer	The number of updates that were lost.
updateshistory	varchar(4000)	varchar(4000)	Bitmask of the last 32 updates.
endtime	ts(3) w tz	ts(3) w tz	End of when the ClassAd is valid.

Attributes of daemons_vertical Table			
Name	O. Type	P. Type	Description
mytype	varchar(100)	varchar(100)	The type of daemon ClassAd, e.g. "Master"
name	varchar(500)	varchar(500)	The name identifier of the daemon ClassAd.
attr	varchar(4000)	varchar(4000)	Attribute name.
val	clob	text	Attribute value.
lastreportedtime	ts(3) w tz	ts(3) w tz	Time when the daemon last reported to Quill.
PRIMARY KEY: (mytype, name, attr)			
NOT NULL: mytype, name, and attr cannot be null			

Attributes of daemons_vertical_history Table			
Name	O. Type	P. Type	Description
mytype	varchar(100)	varchar(100)	The type of daemon ClassAd, e.g. "Master"
name	varchar(500)	varchar(500)	The name identifier of the daemon ClassAd.
lastreportedtime	ts(3) w tz	ts(3) w tz	Time when the daemon last reported to Quill.
attr	varchar(4000)	varchar(4000)	Attribute name.
val	clob	text	Attribute value.
endtime	ts(3) w tz	ts(3) w tz	End of when the ClassAd is valid.

Attributes of submitters_horizontal table			
Name	O. Type	P. Type	Description
name	varchar(500)	varchar(500)	Name of the submitter ClassAd.
scheddname	varchar(4000)	varchar(4000)	Name of the schedd where the submitter ad is from.
lastreportedtime	ts(3) w tz	ts(3) w tz	Last time a submitter ClassAd was sent to Quill.
idlejobs	integer	integer	Number of idle jobs of the submitter.
runningjobs	integer	integer	Number of running jobs of the submitter.
heldjobs	integer	integer	Number of held jobs of the submitter.
flockedjobs	integer	integer	Number of flocked jobs of the submitter.
PRIMARY KEY: name			
NOT NULL: name cannot be null			

Attributes of submitters_horizontal_history table			
Name	O. Type	P. Type	Description
name	varchar(500)	varchar(500)	Name of the submitter ClassAd.
scheddname	varchar(4000)	varchar(4000)	Name of the schedd where the submitter ad is from.
lastreportedtime	ts(3) w tz	ts(3) w tz	Last time a submitter ClassAd was sent to Quill.
idlejobs	integer	integer	Number of idle jobs of the submitter.
runningjobs	integer	integer	Number of running jobs of the submitter.
heldjobs	integer	integer	Number of held jobs of the submitter.
flockedjobs	integer	integer	Number of flocked jobs of the submitter.
endtime	ts(3) w tz	ts(3) w tz	End of when the ClassAd is valid.

A.3 Files Tables

Attributes of files Table			
Name	O. Type	P. Type	Description
file_id	int	int	Unique numeric identifier of the file.
name	varchar(4000)	varchar(4000)	File name.
host	varchar(4000)	varchar(4000)	Name of machine where the file is located.
path	varchar(4000)	varchar(4000)	Directory path to the file.
acl_id	integer	integer	Not yet used, null.
lastmodified	ts(3) w tz	ts(3) w tz	Timestamp of the file.
filesize	numeric(38)	numeric(38)	Size of the file in bytes.
checksum	varchar(32)	varchar(32)	MD5 checksum of the file.
PRIMARY KEY: file_id			
NOT NULL: file_id cannot be null			

Attributes of fileusages Table			
Name	O. Type	P. Type	Description
globaljobid	varchar(4000)	varchar(4000)	Global identifier of the job that used the file.
file_id	int	int	Numeric identifier of the file.
usagetype	varchar(4000)	varchar(4000)	Type of use of the file by the job, e.g., input, output, command.
REFERENCE: file_id references files(file_id)			

Attributes of transfers Table			
Name	O. Type	P. Type	Description
globaljobid	varchar(4000)	varchar(4000)	Unique global identifier for the job.
src_name	varchar(4000)	varchar(4000)	Name of the file on the source machine.
src_host	varchar(4000)	varchar(4000)	Name of the source machine.
src_port	integer	integer	Source port number used for the transfer.
src_path	varchar(4000)	varchar(4000)	Path to the file on the source machine.
src_daemon	varchar(30)	varchar(30)	Condor demon performing the transfer on the source machine.
src_protocol	varchar(30)	varchar(30)	The protocol used on the source machine.
src_credential_id	integer	integer	Not yet used, null.
src_acl_id	integer	integer	Not yet used, null.
dst_name	varchar(4000)	varchar(4000)	Name of the file on the destination machine.
dst_host	varchar(4000)	varchar(4000)	Name of the destination machine.
dst_port	integer	integer	Destination port number used for the transfer.
dst_path	varchar(4000)	varchar(4000)	Path to the file on the destination machine.
dst_daemon	varchar(30)	varchar(30)	Condor daemon receiving the transfer on the destination machine.
dst_protocol	varchar(30)	varchar(30)	The protocol used on the destination machine.
dst_credential_id	integer	integer	Not yet used, null.
dst_acl_id	integer	integer	Not yet used, null.
transfer_intermediary_id	integer	integer	Not yet used, null; will use someday if a proxy is used.
transfer_size_bytes	numeric(38)	numeric(38)	Size of the data transferred in bytes.
elapsed	numeric(38)	numeric(38)	Number of seconds that elapsed during the transfer.
checksum	varchar(256)	varchar(256)	Checksum of the file.
transfer_time	ts(3) w tz	ts(3) w tz	Time when the transfer took place.
last_modified	ts(3) w tz	ts(3) w tz	Last modified time for the file that was transferred.
is_encrypted	varchar(5)	varchar(5)	(boolean) True if the file is encrypted.
delegation_method_id	integer	integer	Not yet used, null.
completion_code	integer	integer	Indicates whether the transfer failed or succeeded.

A.4 Interface Tables

Attributes of cdb_users Table			
Name	O. Type	P. Type	Description
userid	varchar(30)	varchar(30)	Unique identifier of the user
password	character(32)	character(32)	Encrypted password
admin	varchar(5)	varchar(5)	(boolean) True if the user has administrator privileges

Attributes of l_eventtype Table			
Name	O. Type	P. Type	Description
eventtype	integer	integer	Numeric type code of the event.
description	varchar(4000)	varchar(4000)	Description of the type of event associated with the eventtype code.

Attributes of l_jobstatus Table			
Name	O. Type	P. Type	Description
jobstatus	integer	integer	Numeric code for job status.
abbrev	char(1)	char(1)	Single letter code for job status.
description	varchar(4000)	varchar(4000)	Description of job status.
PRIMARY KEY: jobstatus			
NOT NULL: jobstatus cannot be null			

A.5 Jobs Tables

Attributes of clusterads_horizontal Table			
Name	O. Type	P. Type	Description
scheddname	varchar(4000)	varchar(4000)	Name of the schedd the job is submitted to.
cluster_id	integer	integer	Cluster identifier for the job.
owner	varchar(30)	varchar(30)	User who submitted the job.
jobstatus	integer	integer	Current status of the job.
jobprio	integer	integer	Priority for this job.
imagesize	numeric(38)	numeric(38)	Estimate of memory image size of the job in kilobytes.
qdate	ts(3) w tz	ts(3) w tz	Time the job was submitted to the job queue.
remoteusercpu	numeric(38)	numeric(38)	Total number of seconds of user CPU time the job used on remote machines.
remotewallclocktime	numeric(38)	numeric(38)	Committed cumulative number of seconds the job has been allocated to a machine.
cmd	clob	text	Path to and filename of the job to be executed.
args	clob	text	Arguments passed to the job.
jobuniverse	integer	integer	The Condor universe used by the job.
PRIMARY KEY: (scheddname, cluster_id)			
NOT NULL: scheddname and cluster_id cannot be null			
INDEX: Index named ca_h_i_owner on owner			

Attributes of clusterads_vertical Table			
Name	O. Type	P. Type	Description
scheddname	varchar(4000)	varchar(4000)	Name of the schedd that the job is submitted to.
cluster_id	integer	integer	Cluster identifier for the job.
attr	varchar(2000)	varchar(2000)	Attribute name.
val	clob	text	Attribute value.
PRIMARY KEY: (scheddname, cluster_id, attr)			
NOT NULL: scheddname, cluster_id, and attr cannot be null			

Attributes of jobs_horizontal_history Table – Part 1 of 3			
Name	O. Type	P. Type	Description
scheddname	varchar(4000)	varchar(4000)	Name of the schedd that submitted the job.
scheddbirthdate	integer	integer	The birth date of the schedd where the job is submitted.
cluster_id	integer	integer	Cluster identifier for the job.
proc_id	integer	integer	Process identifier for the job.
qdate	ts(3) w tz	ts(3) w tz	Time the job was submitted to the job queue.
owner	varchar(30)	varchar(30)	User who submitted the job.
globaljobid	varchar(4000)	varchar(4000)	Unique global identifier for the job.
numckpts	integer	integer	Number of checkpoints written by the job during its lifetime.
numrestarts	integer	integer	Number of restarts from a checkpoint attempted by the job in its lifetime.
numsystemholds	integer	integer	Number of times Condor-G placed the job on hold.
condorversion	varchar(4000)	varchar(4000)	Version of Condor that ran the job.
condorplatform	varchar(4000)	varchar(4000)	Platform of the computer where the schedd runs.
rootdir	varchar(4000)	varchar(4000)	Root directory on the system where the job is submitted from.
iwd	varchar(4000)	varchar(4000)	Initial working directory of the job.
jobuniverse	integer	integer	The Condor universe used by the job.
cmd	clob	text	Path to and filename of the job to be executed.
minhosts	integer	integer	Minimum number of hosts that must be in the claimed state for this job, before the job may enter the running state.
maxhosts	integer	integer	Maximum number of hosts this job would like to claim.
jobprio	integer	integer	Priority for this job.
negotiation_user_name	varchar(4000)	varchar(4000)	User name in which the job is negotiated.
env	clob	text	Environment under which the job ran.
userlog	varchar(4000)	varchar(4000)	User log where the job events are written to.
coresize	numeric(38)	numeric(38)	Maximum allowed size of the core file.

Table Continues on Next Page

Attributes of jobs_horizontal_history Table – Part 2 of 3			
Name	O. Type	P. Type	Description
killsig	varchar(4000)	varchar(4000)	Signal to be sent if the job is put on hold.
stdin	varchar(4000)	varchar(4000)	The file used as stdin.
transferin	varchar(5)	varchar(5)	(boolean) For globus universe jobs. True if input should be transferred to the remote machine.
stdout	varchar(4000)	varchar(4000)	The file used as stdout.
transferout	varchar(5)	varchar(5)	(boolean) For globus universe jobs. True if output should be transferred back to the submit machine.
stderr	varchar(4000)	varchar(4000)	The file used as stderr.
transfererr	varchar(5)	varchar(5)	(boolean) For globus universe jobs. True if error output should be transferred back to the submit machine.
shouldtransferfiles	varchar (4000)	varchar(4000)	Whether Condor should transfer files to and from the machine where the job runs.
transferfiles	varchar(4000)	varchar(4000)	Deprecated. Similar to shouldtransferfiles.
executablesize	numeric(38)	numeric(38)	Size of the executable in kilobytes.
diskusage	integer	integer	Size of the executable and input files to be transferred.
filesystemdomain	varchar(4000)	varchar(4000)	Name of the networked file system used by the job.
args	clob	text	Arguments passed to the job.
lastmatchtime	ts(3) w tz	ts(3) w tz	Time when the job was last successfully matched with a resource.
numjobmatches	integer	integer	Number of times the negotiator matches the job with a resource.
jobstartdate	ts(3) w tz	ts(3) w tz	Time when the job first began running.
jobcurrentstartdate	ts(3) w tz	ts(3) w tz	Time when the job's current run started.
jobruncount	integer	integer	Number of times a shadow has been started for the job.
filereadcount	numeric(38)	numeric(38)	Number of read(2) calls the job made (only standard universe).
filereadbytes	numeric(38)	numeric(38)	Number of bytes read by the job (only standard universe).
filewritecount	numeric(38)	numeric(38)	Number of write calls the job made (only standard universe).
filewritebytes	numeric(38)	numeric(38)	Number of bytes written by the job (only standard universe).

Table Continues on Next Page

Attributes of jobs_horizontal_history Table – Part 3 of 3			
Name	O. Type	P. Type	Description
fileseekcount	numeric(38)	numeric(38)	Number of seek calls that this job made (only standard universe).
totalsuspensions	integer	integer	Number of times the job has been suspended during its lifetime
imagesize	numeric(38)	numeric(38)	Estimate of memory image size of the job in kilobytes.
exitstatus	integer	integer	No longer used by Condor.
localusercpu	numeric(38)	numeric(38)	Number of seconds of user CPU time the job used on the submit machine.
localsyscpu	numeric(38)	numeric(38)	Number of seconds of system CPU time the job used on the submit machine.
remoteusercpu	numeric(38)	numeric(38)	Number of seconds of user CPU time the job used on remote machines.
remotesyscpu	numeric(38)	numeric(38)	Number of seconds of system CPU time the job used on remote machines.
bytessent	numeric(38)	numeric(38)	Number of bytes sent to the job.
bytesrecvd	numeric(38)	numeric(38)	Number of bytes received by the job.
rsbytessent	numeric(38)	numeric(38)	Number of remote system call bytes sent to the job.
rsbytesrecvd	numeric(38)	numeric(38)	Number of remote system call bytes received by the job.
exitcode	integer	integer	Exit return code of the user job. Used when a job exits by means other than a signal.
jobstatus	integer	integer	Current status of the job.
enteredcurrentstatus	ts(3) w tz	ts(3) w tz	Time the job entered into its current status.
remotewallclocktime	numeric(38)	numeric(38)	Cumulative number of seconds the job has been allocated to a machine.
lastremotehost	varchar(4000)	varchar(4000)	The remote host for the last run of the job.
completiondate	ts(3) w tz	ts(3) w tz	Time when the job completed; 0 if job has not yet completed.
enteredhistorytable	ts(3) w tz	ts(3) w tz	Time when the job entered the history table.
PRIMARY KEY: (scheddname, scheddbirthdate, cluster_id, proc_id)			
NOT NULL: scheddname, scheddbirthdate, cluster_id, and proc_id cannot be null			
INDEX: Index named jobs_hor_his_ix1 on owner			
INDEX: Index named jobs_hor_his_ix2 on enteredhistorytable			

Attributes of jobs_vertical_history Table			
Name	O. Type	P. Type	Description
scheddname	varchar(4000)	varchar(4000)	Name of the schedd that submitted the job.
scheddbirthdate	integer	integer	The birth date of the schedd where the job is submitted.
cluster_id	integer	integer	Cluster identifier for the job.
proc_id	integer	integer	Process identifier for the job.
attr	varchar(2000)	varchar(2000)	Attribute name.
val	clob	text	Attribute value.
PRIMARY KEY: (scheddname, scheddbirthdate, cluster_id, proc_id, attr)			
NOT NULL: scheddname, scheddbirthdate, cluster_id, proc_id, and attr cannot be null			

Attributes of procads_horizontal Table			
Name	O. Type	P. Type	Description
scheddname	varchar(4000)	varchar(4000)	Name of the schedd that submitted the job.
cluster_id	integer	integer	Cluster identifier for the job.
proc_id	integer	integer	Process identifier for the job.
jobstatus	integer	integer	Current status of the job.
imagesize	numeric(38)	numeric(38)	Estimate of memory image size of the job in kilobytes.
remoteusercpu	numeric(38)	numeric(38)	Total number of seconds of user CPU time the job used on remote machines.
remotewallclocktime	numeric(38)	numeric(38)	Cumulative number of seconds the job has been allocated to a machine.
remotehost	varchar(4000)	varchar(4000)	Name of the machine running the job.
globaljobid	varchar(4000)	varchar(4000)	Unique global identifier for the job.
jobprio	integer	integer	Priority of the job.
args	clob	text	Arguments passed to the job.
shadowbday	ts(3) w tz	ts(3) w tz	The time when the shadow was started.
enteredcurrentstatus	ts(3) w tz	ts(3) w tz	Time the job entered its current status.
numrestarts	integer	integer	Number of times the job has restarted.
PRIMARY KEY: (scheddname, cluster_id, proc_id)			
NOT NULL: scheddname, cluster_id, and proc_id cannot be null			

Attributes of procads_vertical Table			
Name	O. Type	P. Type	Description
scheddname	varchar(4000)	varchar(4000)	Name of the schedd that submitted the job.
cluster_id	integer	integer	Cluster identifier for the job.
proc_id	integer	integer	Process identifier for the job.
attr	varchar(2000)	varchar(2000)	Attribute name.
val	clob	text	Attribute value.
PRIMARY KEY: (scheddname, cluster_id, proc_id, attr)			
NOT NULL: scheddname, cluster_id, proc_id, and attr cannot be null			

A.6 Machines Tables

Attributes of machines_horizontal Table – Part 1 of 2			
Name	O. Type	P. Type	Description
machine_id	varchar(4000)	varchar(4000)	Unique identifier of the machine.
opsys	varchar(4000)	varchar(4000)	Operating system running on the machine.
arch	varchar(4000)	varchar(4000)	Architecture of the machine.
state	varchar(4000)	varchar(4000)	Condor state of the machine.
activity	varchar(4000)	varchar(4000)	Condor job activity on the machine.
keyboardidle	integer	integer	Number of seconds since activity has been detected on any keyboard or mouse associated with the machine.
consoleidle	integer	integer	Number of seconds since activity has been detected on the console keyboard or mouse.
loadavg	real	real	Current load average of the machine.
condorloadavg	real	real	Portion of load average generated by Condor
totalloadavg	real	real	
virtualmemory	integer	integer	Amount of currently available virtual memory in kilobytes.
memory	integer	integer	Amount of RAM in megabytes.
totalvirtualmemory	integer	integer	
cpubusytime	integer	integer	Time in seconds since cpuisbusy became true.
cpuisbusy	varchar(5)	varchar(5)	(boolean) True when the CPU is busy.
currentrank	real	real	The machine owner's affinity for running the Condor job which it is currently hosting.
clockmin	integer	integer	Number of minutes passed since midnight.
clockday	integer	integer	The day of the week.
lastreportedtime	ts(3) w tz	ts(3) w tz	Time when the Condor central manager last received a status update from this machine.
enteredcurrentactivity	ts(3) w tz	ts(3) w tz	Time when the machine entered the current activity.
enteredcurrentstate	ts(3) w tz	ts(3) w tz	Time when the machine entered the current state.
updatesequencenumber	integer	integer	Each update includes a sequence number.

Table Continues on Next Page

Attributes of machines_horizontal Table – Part 2 of 2			
updatestotal	integer	integer	The number of updates received from the daemon.
updatessequenced	integer	integer	The number of updates that were in order.
updateslost	integer	integer	The number of updates that were lost.
globaljobid	varchar(4000)	varchar(4000)	Unique global identifier for the job.
lastreportedtime_epoch	integer	integer	The equivalent epoch time of lastreportedtime.
PRIMARY KEY: machine_id			
NOT NULL: machine_id cannot be null			

Attributes of machines_horizontal_history Table – Part 1 of 2			
Name	O. Type	P. Type	Description
machine_id	varchar(4000)	varchar(4000)	Unique identifier of the machine.
opsys	varchar(4000)	varchar(4000)	Operating system running on the machine.
arch	varchar(4000)	varchar(4000)	Architecture of the machine.
state	varchar(4000)	varchar(4000)	Condor state of the machine.
activity	varchar(4000)	varchar(4000)	Condor job activity on the machine.
keyboardidle	integer	integer	Number of seconds since activity has been detected on any keyboard or mouse associated with the machine.
consoleidle	integer	integer	Number of seconds since activity has been detected on the console keyboard or mouse.
loadavg	real	real	Current load average of the machine.
condorloadavg	real	real	Portion of load average generated by Condor
totalloadavg	real	real	
virtualmemory	integer	integer	Amount of currently available virtual memory in kilobytes.
memory	integer	integer	Amount of RAM in megabytes.
totalvirtualmemory	integer	integer	
cpubusytime	integer	integer	Time in seconds since cpuisbusy became true.
cpuisbusy	varchar(5)	varchar(5)	(boolean) True when the CPU is busy.
currentrank	real	real	The machine owner's affinity for running the Condor job which it is currently hosting.
clockmin	integer	integer	Number of minutes passed since midnight.
clockday	integer	integer	The day of the week.
lastreportedtime	ts(3) w tz	ts(3) w tz	Time when the Condor central manager last received a status update from this machine.
enteredcurrentactivity	ts(3) w tz	ts(3) w tz	Time when the machine entered the current activity.
enteredcurrentstate	ts(3) w tz	ts(3) w tz	Time when the machine entered the current state.
updatesequencenumber	integer	integer	Each update includes a sequence number.
Table Continues on Next Page			

Attributes of machines_horizontal_history Table – Part 2 of 2			
Name	O. Type	P. Type	Description
updatestotal	integer	integer	The number of updates received from the daemon.
updatessequenced	integer	integer	The number of updates that were in order.
updateslost	integer	integer	The number of updates that were lost.
globaljobid	varchar(4000)	varchar(4000)	Unique global identifier for the job.
end_time	ts(3) w tz	ts(3) w tz	The end of when the ClassAd is valid.

Attributes of machines_vertical Table			
Name	O. Type	P. Type	Description
machine_id	varchar(4000)	varchar(4000)	Unique identifier of the machine.
attr	varchar(2000)	varchar(2000)	Attribute name.
val	clob	text	Attribute value.
start_time	ts(3) w tz	ts(3) w tz	Time when this attribute–value pair became valid.
PRIMARY KEY: (machine_id, attr)			
NOT NULL: machine_id and attr cannot be null			

Attributes of machines_vertical_history Table			
Name	O. Type	P. Type	Description
machine_id	varchar(4000)	varchar(4000)	Unique identifier of the machine.
attr	varchar(4000)	varchar(4000)	Attribute name.
val	clob	text	Attribute value.
start_time	ts(3) w tz	ts(3) w tz	Time when this attribute–value pair became valid.
end_time	ts(3) w tz	ts(3) w tz	Time when this attribute–value pair became invalid.

A.7 Matchmaking Tables

Attributes of matches Table			
Name	O. Type	P. Type	Description
match_time	ts(3) w tz	ts(3) w tz	Time the match was made.
username	varchar(4000)	varchar(4000)	User who submitted the job.
scheddname	varchar(4000)	varchar(4000)	Name of the schedd that the job is submitted to.
cluster_id	integer	integer	Cluster identifier for the job.
proc_id	integer	integer	Process identifier for the job.
globaljobid	varchar(4000)	varchar(4000)	Unique global identifier for the job.
machine_id	varchar(4000)	varchar(4000)	Identifier of the machine the job matched with.
remote_user	varchar(4000)	varchar(4000)	User that was preempted.
remote_priority	real	real	The preempted user's priority.

Attributes of rejects Table			
Name	O. Type	P. Type	Description
reject_time	ts(3) w tz	ts(3) w tz	Time when the job was rejected.
username	varchar(4000)	varchar(4000)	User who submitted the job.
scheddname	varchar(4000)	varchar(4000)	Name of the schedd that submitted the job.
cluster_id	integer	integer	Cluster identifier for the job.
proc_id	integer	integer	Process identifier for the job.
globaljobid	varchar(4000)	varchar(4000)	Unique global identifier for the job.

A.8 Runtime Tables

Attributes of events Table			
Name	O. Type	P. Type	Description
scheddname	varchar(4000)	varchar(4000)	Name of the schedd that submitted the job.
cluster_id	integer	integer	Cluster identifier for the job.
proc_id	integer	integer	Process identifier for the job.
globaljobid	varchar(4000)	varchar(4000)	Global identifier of the job that generated the event.
run_id	numeric(12,0)	numeric(12,0)	Identifier of the run that the event is associated with.
eventtype	integer	integer	Numeric type code of the event.
eventtime	ts(3) w tz	ts(3) w tz	Time the event occurred.
description	varchar(4000)	varchar(4000)	Description of the event.

Attributes of generic_messages Table			
Name	O. Type	P. Type	Description
eventtype	varchar(4000)	varchar(4000)	The type of event.
eventkey	varchar(4000)	varchar(4000)	The key of the event.
eventtime	ts(3) w tz	ts(3) w tz	The time of the event.
eventloc	varchar(4000)	varchar(4000)	The location of the event.
attname	varchar(4000)	varchar(4000)	The attribute name.
attvalue	clob	text	The attribute value.
atttype	varchar(4000)	varchar(4000)	The attribute type.

Attributes of runs Table			
Name	O. Type	P. Type	Description
run_id	numeric(12)	numeric(12)	Unique identifier of the run.
machine_id	varchar(4000)	varchar(4000)	Identifier of the machine where the job ran.
scheddname	varchar(4000)	varchar(4000)	Name of the schedd that submitted the job.
cluster_id	integer	integer	Cluster identifier for the job.
proc_id	integer	integer	Process identifier for the job.
spid	integer	integer	
globaljobid	varchar(4000)	varchar(4000)	Identifier of the job that was run.
startts	ts(3) w tz	ts(3) w tz	Time when the job started.
endts	ts(3) w tz	ts(3) w tz	Time when the job ended.
endtype	smallint	smallint	The type of ending event.
endmessage	varchar(4000)	varchar(4000)	The ending message.
wascheckpointed	varchar(7)	varchar(7)	Whether the run was checkpointed.
imagesize	numeric(38)	numeric(38)	The image size of the executable.
runlocalusageuser	integer	integer	The time the job spent in usermode on execute machines (only standard universe).
runlocalusagesystem	integer	integer	The time the job was in system calls.
runremoteusageuser	integer	integer	The time the shadow spent working for the job.
runremoteusagesystem	integer	integer	The time the shadow spent in system calls for the job.
runbytessent	numeric(38)	numeric(38)	Number of bytes sent to the run.
runbytesreceived	numeric(38)	numeric(38)	Number of bytes received from the run.
PRIMARY KEY: run_id			
NOT NULL: run_id cannot be null			
INDEX: Index named runs.idx1 on (scheddname, cluster_id, proc_id)			

A.9 System Tables

Attributes of dummy_single_row_table Table			
Name	O. Type	P. Type	Description
a	varchar(1)	varchar(1)	A dummy column.

Attributes of history_jobs_to_purge Table			
scheddname	varchar(4000)	varchar(4000)	Name of the schedd that submitted the job.
cluster_id	integer	integer	Cluster identifier for the job.
proc_id	integer	integer	Process identifier for the job.
globaljobid	varchar(4000)	varchar(4000)	Unique global identifier for the job.

Attributes of jobqueuepollinginfo Table			
Name	O. Type	P. Type	Description
scheddname	varchar(4000)	varchar(4000)	Name of the schedd that submitted the job.
last_file_mtime	integer	integer	The last modification time of the file.
last_file_size	numeric(38)	numeric(38)	The last size of the file in bytes.
last_next_cmd_offset	integer	integer	The last offset for the next command.
last_cmd_offset	integer	integer	The last offset of the current command.
last_cmd_type	smallint	smallint	The last type of command.
last_cmd_key	varchar(4000)	varchar(4000)	The last key of the command.
last_cmd_mytype	varchar(4000)	varchar(4000)	The last my ClassAd type of the command.
last_cmd_targettype	varchar(4000)	varchar(4000)	The last target ClassAd type.
last_cmd_name	varchar(4000)	varchar(4000)	The attribute name of the command.
last_cmd_value	varchar(4000)	varchar(4000)	The attribute value of the command.
INDEX: Index named jq_i_schedd on scheddname			