# Transparently Gathering Provenance with Provenance Aware Condor

Christine F. Reilly
*University of Wisconsin-Madison*

Jeffrey F. Naughton
*University of Wisconsin-Madison*

## Abstract

We observed that the Condor batch execution system exposes a lot of information about the jobs that run in the system. This observation led us to explore whether this system information could be used for provenance. The result of our explorations is Provenance Aware Condor (PAC), a system that transparently gathers provenance while jobs run in Condor. Transparent provenance gathering requires that the application not be altered in order to run in the provenance system. This requirement allows any application that can run in Condor to also run in PAC. Through SQL queries, PAC is able to answer a wide range of questions about the files used by a job and the machines that execute jobs.

## 1 Introduction

Provenance, in the broadest use of the word, is the origin and history of an item. In computer science we focus on gathering provenance about data with the goal of being able to answer questions about where the data came from and the process that created it. There has been a great deal of work on the semantics of provenance and there are many proposals and systems for gathering provenance about specific types of data. While we think that work is essential, our work focuses on a largely orthogonal issue. Rather than aiming for ideal provenance or designing a provenance system for a specific type of application, we examine the possibility of gathering information that is useful for provenance while having little or no impact on the people writing applications or on the systems executing these applications.

We are motivated by our observations of the Condor system [3]. Condor is a production system that executes user jobs on clusters of machines, and is used on over 100,000 CPUs worldwide. It is used by groups of all sizes in universities, government labs, and private companies. A wide variety of applications are run in Condor, representing a diverse set of fields that includes computer engineering, biology, chemistry, physics, finance, and insurance.

We observed that Condor is privy to a lot of information about the jobs it runs. This led us to wonder if we could use Condor as the base for a system that transparently gathers provenance [11]. A transparent provenance system gathers information without requiring that applications be altered. Because we start with an existing job execution system, we also aim to transparently add provenance gathering capabilities to Condor by only changing the system in ways that are not visible on the surface.

A benefit of using Condor as the base of our provenance system is its ability to run a wide range of applications. Any application that runs on the same operating system as at least one machine in the cluster and is able to run as a background batch job can run in Condor.

A number of interesting questions arise from our proposed provenance system design. The most obvious question is if it is actually possible to gather provenance information without making changes that are visible to the application designer, Condor user, or Condor system administrator. Once we found that it is possible to design a transparent system, our next question was whether the provenance gathered by such a system is useful. We also wondered about the overhead imposed by this provenance system on applications and Condor in terms of running time and data storage. In order to explore these questions we built a prototype provenance system on top of Condor. We call the combination of Condor and our provenance system Provenance Aware Condor (PAC).

By adding provenance capabilities to the Condor system, our goal is to provide a baseline amount of provenance for any application that runs in Condor. We recognize that, by gathering provenance for a wide range of applications, we are not able to gather all of the information provided by special–purpose provenance systems. For this reason, we expect special–purpose provenance systems to continue to be used and developed for applications that are widely used, and for those that will be in use for a long duration of time. However, there are many applications that are unlikely to ever be modified solely for the purpose of gathering provenance. These applications are the target of our provenance system. Our question is not whether our system is as powerful as a special–purpose system, rather it is whether it can be useful at all while requiring virtually no modifications to user programs or the execution system.

## 2 Key Features

There are a number of features that are unique to our provenance system. In this section we identify these key features and compare them with the features offered by other provenance systems.

### 2.1 Transparency

A transparent provenance system is able to gather provenance from many types of applications without requiring the user to alter the application to conform with requirements of the provenance system. The trade off of transparency is that it can result in a reduction in the detail or specificity of the provenance, or in a decreased ease of use. In contrast, provenance systems that require a user to express her computation in a specific framework, or that embed application specific semantics, can provide detailed information related to the logical properties of the application.

PAC is transparent because it is able to gather provenance from any program that runs in Condor. Although PAC does require a program to run in Condor, we consider PAC to be transparent for two reasons. First, Condor is able to run any program that can run in the background. Second, any program that currently runs in Condor can utilize PAC without making any changes to the program or to how it is submitted to Condor. PAC provides provenance to programs that currently run in Condor for “free”. The Earth System Science Server (ES3) [6, 7] uses a similar approach for transparently collecting provenance. ES3 gathers provenance by monitoring programs as they run. Users do not need to alter their programs in order to use ES3.

Another strategy for obtaining detailed provenance is to require certain behaviors from applications. For example, Trio [1] requires the data used by the application to be stored in its database, and only records provenance for computations expressed as queries over this data. This design allows Trio to gather provenance at the granularity of a database tuple, but it excludes applications that do not conform to its data storage and computation model.

### 2.2 Complete Information

A provenance system has complete information when it collects information about everything the application did while it executed. This includes information about what files were used, along with details about the files. It also includes information about how the job used the execution system.

#### 2.2.1 Files

Because PAC collects file information by monitoring the open and close file calls made by a program, it obtains information about every file used by a job. The files used by a job will be recorded even if the user is not aware that some files are accessed through a shared file system. Information about system files and libraries allows the user to track problems related to changes in these files.

Another system that gathers information by monitoring system activity is the Provenance–Aware Storage System (PASS) [10]. PASS is implemented as a layer on top of a conventional file system and stores provenance in an in–kernel port of Berkeley DB. Because PASS is a storage system, it maintains provenance for every file and detects all file activities. By integrating a provenance system with the storage system, PASS has knowledge of everything that happens to a file. In contrast, PAC is only able to record information about files when they are used by an application that is running in PAC. If a file is renamed or altered by a process that is not part of PAC, then PAC will have no knowledge of the alteration. A drawback of PASS is that it requires the use of its storage system. In contrast, PAC runs as on application on existing desktop computer systems.

#### 2.2.2 System Information

Because PAC is incorporated within a job execution system, it is able to record information about the system and about how a job uses the system. For each machine in the system, PAC records the machine architecture, operating system, and amount of memory. Information about the process that matches jobs with machines can be used to determine why Condor is not able to match a specific job with a machine in the cluster. For each job, PAC records the machine that executed the job and the environment variables used during job execution. PASS is the only other provenance system we are aware of that is tightly integrated with the underlying computing system.

### 2.3 Integration with an existing distributed computing system

By collecting provenance from with the Condor system, PAC records details about the job as the events occur.

Other systems use a wrapper around the execution system to collect information. The wrapper is unable to gather information about processes that occur entirely within the execution system.

The Pegasus system [9] refines an abstract workflow into an executable workflow. It uses Condor as the execution environment. Pegasus collects two sets of provenance: information about the workflow refinement, and information about the execution. Pegasus uses a wrapper around each job in the workflow to gather provenance from the job execution process. Because Pegasus performs its provenance collection outside of Condor, it cannot detect the use of system files, libraries, and other files that are not explicitly declared by the job.

## 2.4 PAC as a Base for a Workflow Management System

PAC can reconstruct a workflow by chaining jobs together based on one job using the same file for input as another job wrote as output. Because PAC does not have a concept of a workflow as an entity, it is unable to provide information about how a workflow was created or how it changes over time. Workflow management systems, such as Pegasus [9] and VisTrails [12] have the capability to provide detailed information about workflows, but lack detailed information about the execution of individual workflow nodes. Combining PAC with a provenance system that gathers higher level information would result in the collection of more provenance than either system could collect alone.

## 3 The Provenance Gathering System

Provenance Aware Condor is composed of three parts. The first part is the Condor job scheduling system [3]. Quill [3, 8], the second part of PAC, was added to Condor to capture more information about Condor and make the information available through a SQL interface. In this work we are examining the use of Quill as part of a provenance system. We added FileTrace, the final part of PAC, to take care of some deficiencies with respect to provenance in the Condor with Quill system.

One of our major design principles is that the additions we make to Condor are not disruptive to how Condor runs or to how a user interacts with Condor. We use the word “transparency” to describe this design principal. Because Condor is a production system, transparent changes allow users who are interested in provenance to utilize the new features while allowing other users to remain unaware of the changes.

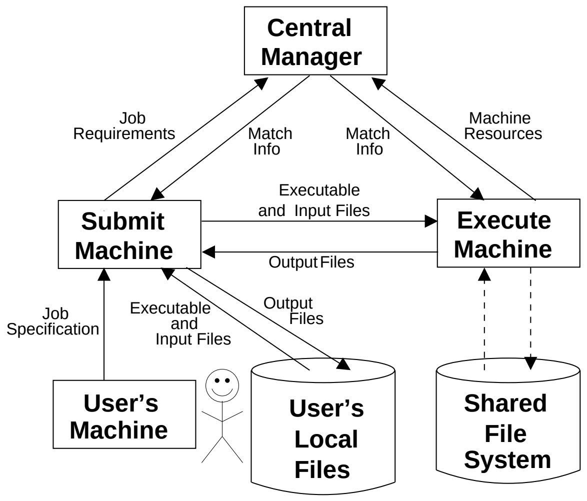


Figure 1: Interactions in a Condor System

## 3.1 Condor

Condor [3] is a high throughput computing system that matches user submitted computational jobs with available machines. A Condor system consists of three types of machines: the central manager, submit machines, and execute machines. The system contains a single central manager machine, which is responsible for matching jobs with execute machines. There is at least one submit machine and execute machine in the system, and systems often have many machines that fill these roles. Figure 1 illustrates the interactions between entities in a Condor system.

### 3.1.1 Roles of Machines in a Condor System

The central manager matches the user–submitted jobs with execute machines. Jobs and machines announce their presence by sending a message to the central manager. The message sent by a job specifies the hardware and software requirements of the job. Execute machines send information about their hardware and software resources to the central manager. Periodically, the central manager enters a match cycle where it matches jobs with machines. Any jobs that were not matched with a machine are carried over to the next match cycle.

The submit machine accepts jobs from users and sends information about these jobs to the central manager. When a job is matched with an execute machine, the submit machine transfers the executable and any input files to the execute machine. After the job finishes, the submit machine downloads output files from the execute machine. A submit machine can manage many jobs from many users at the same time.

The role of an execute machine is, simply, to run jobs.

When matched with a job, the execute machine first receives the executable and any input files from the submit machine. Next, it runs the job. When the job finishes, the execute machine sends the output files to the submit machine.

#### 3.1.2 How Condor Accesses Files

The file access methods that are available to a Condor job depend on the Condor runtime environment, known as a universe. In this paper, we only consider the Vanilla Universe, a commonly used and flexible universe. The Vanilla Universe allows a job to access files through two methods: Condor file transfers and a shared file system. The Condor file transfer mechanism transfers files between the user's local storage and the execute machine, through the submit machine. The user provides the pathname of the executable and input files as part of the job specification. When an execute machine is ready to run a job the submit machine transfers the executable and input files to the execute machine. When the job finishes, the execute machine transfers all files that were created or modified by the job to the submit machine, and the submit machine sends the files to the user. When a job uses a shared file system, such as AFS, the job accesses files in the same way as it does when the user runs the executable outside of Condor.

A major difference between using the file transfer mechanism or a shared file system is how the job accesses files. The file transfer mechanism requires the user to specify all input files when the job requirements are sent to the submit machine. Condor transfers these files to the execute machine and they are the only input files available to the job. With a shared file system, the job can access any files, provided it has the proper permissions. It is possible for a job to access files using both file mechanisms.

#### 3.1.3 Condor Logs

The machines in a Condor system record operational information in log files. This information is used by the user and system administrator to monitor jobs and machines, and by developers for debugging. Users and administrators use command–line tools to access information from the Condor logs. Because these logs already existed, we began our evaluation of provenance in Condor by examining the logs. But the logging system was not designed for the purpose of collecting provenance, and we found that the information provided by the logs is insufficient for provenance.

Condor uses a flexible set of attribute–value pairs to store information about current and past jobs in the log files. This set of attribute–value pairs is flexible in that there are no attributes that are guaranteed to be used for every job, and the set of attributes is not necessarily constant between jobs. Additionally, the user is allowed to create arbitrary attributes for each job. Despite this flexibility, there is a set of common attributes that are used by most jobs.

Command-line tools allow the user and administrator to access information about the Condor system. The basic information available is the names of machines in the pool and their usage over time, and the names of users who have ran jobs in a given time period. The basic information can be used to derive more detailed information about system utilization. The Condor Team at the University of Wisconsin has developed visual representations of system utilization and displays the information on a series of web pages.

The job history log does not necessarily contain complete information about the files used by a job. The file information that is recorded depends on the information provided by the user during job submission and on how the executable accesses files. If the user provides relative pathnames, based on the directory where the job is submitted from, the initial working directory and output directory attributes could be used to obtain the absolute pathname. If the job has access to a shared filesystem it can access input and output files that are not specified by the user. Condor only allows one file to be specified as the output file; additional output files are not recorded in the job history log.

### 3.2 Condor with Quill

As part of the CondorDB research group, we observed that we could collect more information from a Condor system than is accessible (or easily accessible) from Condor's log files. We developed Quill [3, 8], an extension to Condor that gathers operational information and saves the information in a relational database. In addition to providing information that is not accessible in Condor, Quill's relational database makes it easier to obtain information that is not available from one of Condor's command–line tools.

We designed Quill to transparently interface with Condor, so that a user or an administrator who is not interested in Quill can remain unaware of its presence. The changes we made to Condor were superficial and we left the operational portions of Condor untouched. In order to achieve this goal of transparency, Quill follows Condor's data collection method of having the various daemons write information to log files. Periodically, Quill reads the logs and inserts the data into a relational database that is shared by all machines in the pool.

Queries to the Quill database can provide the same information as Condor's command–line tools. Two benefits of storing information about a Condor system in Quill's relational database are that SQL queries are often faster than running the command–line tool, and that custom queries are easy to write in SQL.

Quill provides more information about files than is available from Condor. For the files that use Condor's file transfer mechanism, Quill records the absolute path, checksum, size, and last modified time, thus providing version information.

### 3.3 PAC: Condor with Quill and FileTrace

As discussed above, the main area where Quill lacks provenance information is when a job accesses a file without using the file transfer utility. We address this problem with FileTrace, a program that gathers file information from the file system calls made by a program.

FileTrace is a modified version of the strace program [13]. FileTrace required relatively minor modifications to strace. The biggest change was adding the capability to obtain the file checksum. The remainder of the changes involved filtering and formatting the output so it is more human–readable and easier to insert into the database. We only altered how strace handles open and close operations, so the strace output for all of the other system calls remains unchanged. This approach of making minor modifications to only the necessary components fits with our goal of developing a transparent provenance collection system.

We added FileTrace to Condor using methods that are transparent to the user and have little or no affect on the operation of Condor. In order for FileTrace to gather information from an executable, the executable and its arguments are provided to FileTrace as arguments. Condor allows the pool administrator to create a wrapper for executables that are run by Condor. We use this wrapper to run all Condor jobs with FileTrace. The data collected by FileTrace is handled in the same way as data collected by Quill: the output from FileTrace is written to a file and is imported into the database by a process that is separate from the standard Condor operations. FileTrace is a prototype that we developed for the purposes of our research and is not currently shipped with Condor.

### 3.4 Answering Provenance Questions with PAC

Because PAC stores provenance in a relational database, it is able to answer any question that can be expressed as a SQL query over these tables. The portions of the PAC schema that are relevant to provenance include information about jobs, machines, and events in the Condor system. In this discussion we focus only on the information that is relevant to the provenance of jobs and files.

Table 1 provides a summary of the PAC schema. The full Quill schema is available in the Condor Manual [3]. The Jobs table contains information about the machine that ran the job, names of files used by the job, and whether the job finished successfully. PAC stores information about jobs that are currently in the Condor system, as well as jobs that have exited the system. Information recorded about machines includes the hardware, operating system, and state of a machine. One result of transparently building PAC on top of Condor is that PAC has four tables that store information about files. We are only discussing the File Transfers and FileTrace tables. We are omitting the other two tables because they contain redundant information, and they only provide information about files that are listed in the job submission file. The FileTrace table contains the information gathered by the FileTrace program, including when the file was used by Condor and information that can be used to determine the version of the file. The File Transfers table contains information about files that used Condor's File Transfer mechanism to move between the submit and execute machines. PAC contains separate tables for Matches and Rejects, but the schemas of these tables are nearly identical. At the end of a matchmaking cycle, information about matches between jobs and machines is recorded in the Matches table, and information about jobs that failed to match with a machine is recorded in the Rejects table. When a job runs in Condor, information about the run is recorded in the Runs table.

The information gathered by PAC can be used to answer a wide variety of questions. We show two example queries in Tables 2 and 3. These tables show a truncated version of the query results. The queries return a large number of rows due to two factors: PAC records all system files that are used by a program; and the program we used has many input and output files. We selected rows that provide a good illustration of the capabilities of PAC.

The first query (Table 2) finds the files that were used by a specific job. Our result shows the executable, input, and output files that were used by the job. A user can determine if this job used the current input and executable files by comparing the version information from her files with the information returned by the query. This query can also return information about the system files and libraries that the job used.

In the second query (Table 3) we are looking for files that were created by a bad machine. A user can compare the list of files returned by this query with her files in order to determine if she has any files that might be bad.

Table 1: Portions of the PAC Schema Relevant to Provenance

| **PAC Schema** | |
|---|---|
| **Table** | **Attributes** |
| Jobs | job id; submission date; user id; id of the execute machine; Condor version; machine platform; initial working directory; execution environment; Condor log file name; names of files where stdin, stdout, and stderr are redirected; name and size of the executable; exit status; completion time |
| Machines | machine id; operating system; architecture; memory; state; time when it entered current state; other information that allows Condor to know if a machine is available and for Condor to derive matchmaking information. |
| FileTrace | job id; file access time; activity type; pathname; open_flags; filesystem permissions; return value of filesystem call; file_pointer; last_modified time; size; checksum |
| File Transfers | job id; direction of transfer; bytes transferred; timestamp; elapsed time; checksum; last modified time |
| Matches and Rejects | timestamp; id of the user who submitted the job; job id; machine the job matched with (only for matches) |
| Runs | run id; job id; id of machine that ran the job; start and end time of the run; image size of the executable |

Table 2: Example Query: What files were used by this job?

| **SELECT pathname, activity_type, open_flags, last_modified, size, checksum**<br>**FROM filetrace WHERE globaljobid='my_job_id';** | | | | | |
|---|---|---|---|---|---|
| program.exe | exe | | 2008-10-06 17:52:16-05 | 687 | 8FC8316B8930835C9085E 953F2D36745C2877346 |
| /path/to/input.xml | open | O_RDONLY | 2008-10-06 17:52:28-05 | 9554599 | 57B7A53ACF390B1928A6 C4B6E935DD9F2CBE9F46 |
| /path/to/output.xml | open | O_WRONLY\|<br>O_CREAT | 2008-10-06 17:55:18-05 | 0 | DA39A3EE5E6B4B0D3255 BFEF95601890AFD80709 |
| /path/to/input.xml | close | | 2008-10-06 17:52:28-05 | 9554599 | 57B7A53ACF390B1928A6 C4B6E935DD9F2CBE9F46 |
| /path/to/output.xml | close | | 2008-10-06 17:55:19-05 | 19895258 | 9D32885FDCD9171E44F9 302BB0141DFC1FC6B772 |

Table 3: Example Query: Were any of my files generated by the bad machine?

| **SELECT f.globaljobid, f.pathname, f.last_modified, f.checksum, f.size**<br>**FROM filetrace f,**<br>**(SELECT f1.globaljobid, f1.pathname, f1.activity_time, f1.file_pointer**<br>**FROM filetrace f1, jobs_horizontal_history j**<br>**WHERE j.lastremotehost='foo.cs.wisc.edu' AND j.globaljobid=f1.globaljobid**<br>**AND f1.open_flags LIKE '%O_WRONLY%') fopen**<br>**WHERE f.activity_type='close' AND f.globaljobid=fopen.globaljobid AND f.pathname=fopen.pathname**<br>**AND f.file_pointer=fopen.file_pointer AND f.activity_time>=fopen.activity_time;** | | | | |
|---|---|---|---|---|
| job_id | /path/to/output.xml | 2008-08-25 14:43:40-05 | A0CA0D7D2D35C0AE6A4BD657A 115387B33167031 | 185191 |
| job_id | /path/to/log.txt | 2008-08-25 14:43:40-05 | 7A41E9EDF98605F67307C6404 D314C012F4B828E | 148 |

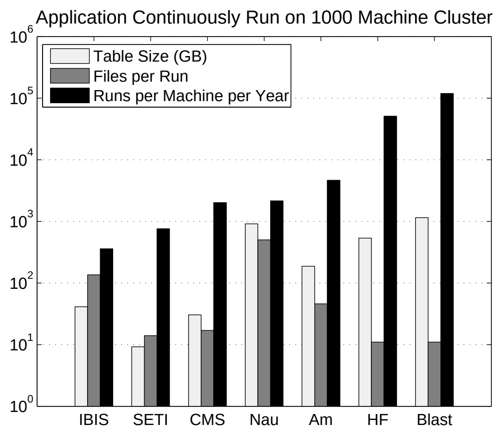


Figure 2: Estimated Size of FileTrace Table After Running Application for One Year

## 4 Performance and Storage Implications of PAC

In order to examine the impacts of PAC on the performance of Condor and to measure the growth rate of the FileTrace table, we performed experiments using a synthetic program. The program writes random numbers to files. In the arguments to the program we specify the number of files to be generated and the approximate duration of the run of the program.

For these experiments we used two identical clusters of machines. Each machine runs Red Hat Enterprise Linux 5, and has a Pentium 2.40 GHz Intel Core 2 Duo processor, 4GB RAM, and two 250GB SATA-I hard disks. One cluster runs Condor without Quill or FileTrace. The other cluster runs PAC: Condor with Quill and FileTrace. Each cluster consists of a central manager machine, a submit machine, and 10 execute machines. The PAC cluster also has a machine for the PostgreSQL database. Because Condor counts each processor as a machine, each of our clusters has 20 execute machines from the perspective of Condor.

### 4.1 Database Size

An issue faced by PAC, and by provenance systems in general, is the amount of data that must be stored. Recording and saving provenance produces a continuously increasing amount of data. We use two methods to estimate the growth rate of the FileTrace table: calculations based on resource usage data for a set of programs, and by directly measuring the size of the FileTrace table after running programs in our cluster. We have not performed any compression on the FileTrace table. It is likely that compression would significantly reduce the size of the table.

Our first estimation of the FileTrace table growth rate uses resource usage data for seven scientific applications [2]. The seven applications are: BLAST (genetics); IBIS (earth system simulation); CMS (high–energy physics); Nautilus (molecular dynamic simulation); Messkit Hartree–Fock (HF) (interactions between atomic nuclei and and electrons); AMANDA (astrophysics); and SETI@home (searches space noise for indications of extraterrestrial intelligence). Each of these applications has a different combination of number of files used and run time.

We estimate the growth rate of the FileTrace table by computing the resulting table size for each application when it is continuously run for one year on 1000 machines. Figure 2 shows the result of these computations. The FileTrace table is a reasonable size for all of these applications. The largest table is 1.15 terabytes, and would easily fit on a couple of currently available disks. As expected, the size of the FileTrace table is affected by both the number of files used by each run of the application and by the time it takes for the application to run. A large value for either variable leads to a big FileTrace table. The biggest FileTrace table, at 1.15 terabytes, is created by Blast. Blast uses the smallest number of files per job, but the large number of runs per year results in a large table size. The second largest table, approximately 900 gigabytes, is generated by Nautilus. Nautilus uses the most files per run and has a medium run time.

This estimate of the size of the FileTrace table is useful for obtaining a rough idea of how fast the table will grow. But FileTrace also records information about the system files and libraries. The table size estimate could also be inaccurate if the application opens and closes the same file multiple times. Including information about these additional file open and close calls will provide a more accurate estimate of the table size.

We used our synthetic program to compare our estimate of the size of the FileTrace table with the table that is produced by PAC. We ran four sets of the program, with the parameters for each set loosely based on the scientific applications that are described above. The parameters are the run time of the program and the number of files produced by the program. The four sets of the program represent the edges of the combined parameters: long run time, many files; long run time, few files; short run time, many files; and short run time, few files. We ran an additional set of jobs that contained an equal number of each of these four configurations. For each set, we ran as many Condor jobs as would take approximately 12 hours to complete.

We found that the actual size of the FileTrace table was half of the estimated size. The estimated and actual

Table 4: Size of FileTrace Table After Running a Synthetic Application

| **Size of FileTrace Table** | | | | |
|---|---|---|---|---|
| Files/Job | Time/Job (min) | Num Jobs | Est. Size (mb) | Measured Size (mb) |
| 500 | 20 | 720 | 303 | 134 |
| 300 | 5 | 2880 | 728 | 323 |
| 10 | 20 | 720 | 6.7 | 3.4 |
| 10 | 5 | 2880 | 27 | 13 |
| mix | mix | 1800 | 266 | 119 |

Table 5: Comparison of Run Time in Condor and PAC

| **Run Time** | | | | | |
|---|---|---|---|---|---|
| Files/Job | min/Job | System | Total time (hh:mm) | time/job (sec) | StdDev |
| 500 | 20 | Condor | 11:00 | 1091 | 21 |
| | | PAC | 10:53 | 1090 | 18 |
| 300 | 5 | Condor | 13:09 | 328 | 6 |
| | | PAC | 13:11 | 328 | 6 |
| 10 | 20 | Condor | 13:07 | 1305 | 21 |
| | | PAC | 13:08 | 1307 | 25 |
| 10 | 5 | Condor | 13:11 | 328 | 6 |
| | | PAC | 13:13 | 329 | 6 |

FileTrace table sizes are listed in Table 4. The FileTrace schema contains a number of variable size attributes. In our estimate, we were conservative and assigned values on the large side of the expected range for the variable size attributes. As stated above, our estimate does not account for the system and library files recorded by FileTrace, but these “invisible” files do not have enough weight to overcome our conservative estimate of the size per record. Although different applications will have different number of “invisible” files, this experiment confirms that the FileTrace table grows at the rate predicted by the size we estimated.

## 4.2 Overhead of PAC

PAC requires Condor to write additional logs. Because Condor already writes many logs, we do not expect this additional log activity to impact the performance of Condor. The interactions with the database are performed in the background and should have little or no impact on the performance of Condor.

In order to confirm that PAC has little impact on the performance of Condor, we ran four sets of synthetic applications in a non–PAC Condor cluster and in a PAC cluster. These are the same applications as used for the FileTrace table size experiments in Section 4.1. For each set, we used the Condor history data to obtain the execution time of each job as well as the total time from submitting all of the jobs to their completion. These results are presented in Table 5. As expected, the job run time in PAC is almost identical to the run time in Condor without Quill or FileTrace. The variation in job run time, both between PAC and Condor and within each system, is well within the variation expected due to the Condor matching process. Condor typically begins a matchmaking cycle every 5 minutes. Depending on when jobs are submitted and when machines become available, machines could remain idle for a few minutes until the next matchmaking cycle begins. From this experiment, we conclude that adding provenance awareness to Condor does not impact the operation of Condor.

## 5 Running a Community Information Management Program with PAC

We use DBLife, a community information management program, to examine how well PAC works with a real–world application. Community information management (CIM) programs [5] provide a central location for collecting information about an online community and for discovering relationships between the entities in that community. DBLife [4] is a CIM prototype that manages information about the database research community.

As illustrated in Figure 3, DBLife creates and main-

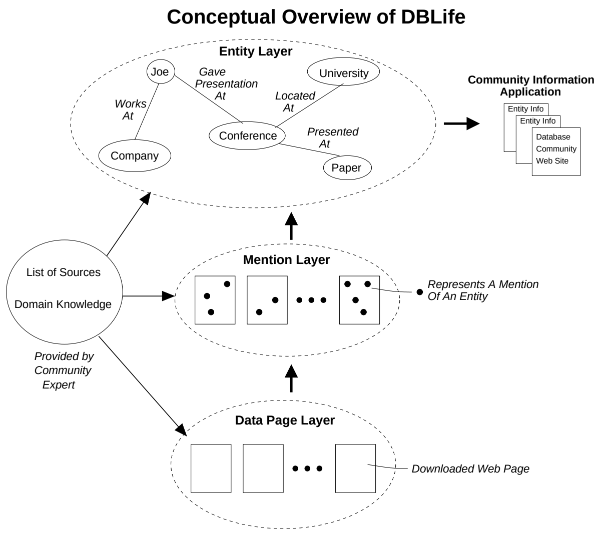
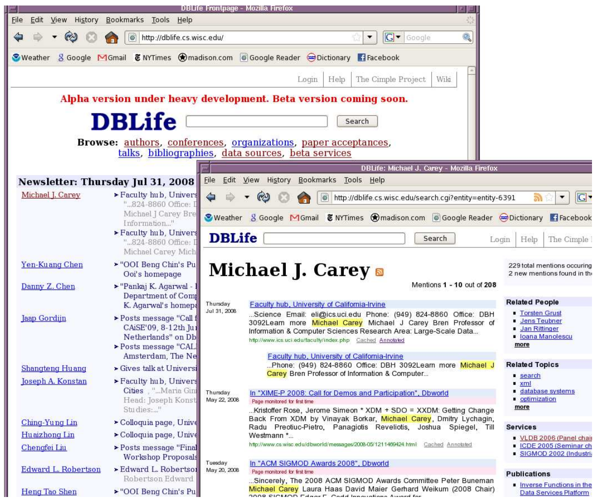


Figure 3: Conceptual Overview of DBLife. Adapted from DeRose, et al. [4]


DBLife Frontpage - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Weather Google Gmail NYTimes madison.com Google Reader Dictionary Facebook
Login Help The Cimple Project Wiki
Search
DBLife: Michael J. Carey - Mozilla Firefox
Cached Annotated

http://dblife.cs.wisc.edu/

Alpha version under heavy development. Beta version coming soon.

DBLife

Browse: authors, conferences, organizations, paper acceptances, talks, bibliographies, data sources, beta services

Newsletter: Thursday Jul 31, 2008

| | |
|---|---|
| Michael J. Carey | Faculty hub, Univers "...824-8860 Office: I Michael J Carey Bre Information..." |
| | Faculty hub, Univers "...824-8860 Office: I Michael Carey Mich |
| Yen-Kuang Chen | "OOI Beng Chin's Pu Ooi's homepage |
| Danny Z. Chen | "Pankaj K. Agarwal - I Department of Comp K. Agarwal's homepa |
| Jaap Gordijn | Posts message "Call f CAiSE'09, 8-12th Ju Netherlands" on Db |
| | Posts message "CAL Amsterdam, The Ne |
| Shangteng Huang | Gives talk at Universi |
| Joseph A. Konstan | Faculty hub, Univers Cities , "...Maria Gin Head: Joseph Konst Studies:..." |
| Ching-Yung Lin | Colloquia page, Unive |
| Huaizhong Lin | Colloquia page, Unive |
| Chengfei Liu | Posts message "Final Workshop Proposals |
| Edward L. Robertson | Edward L. Robertson Robertson Edward |
| Heng Tao Shen | "OOI Beng Chin's Pu |

http://dblife.cs.wisc.edu/search.cgi?entity=entity-6391

DBLife

Michael J. Carey

229 total mentions occuring
2 new mentions found in th

Mentions **1 - 10** out of **208**

Thursday Jul 31, 2008

Faculty hub, University of California-Irvine
...Science Email: eli@ics.uci.edu Phone: (949) 824-8860 Office: DBH 3092Learn more Michael Carey Michael J Carey Bren Professor of Information & Computer Sciences Research Area: Large-Scale Data...
http://www.ics.uci.edu/faculty/index.php

Faculty hub, University of California-Irvine
...Phone: (949) 824-8860 Office: DBH 3092Learn more Michael J Carey Bren Professor of Information & Computer...

Thursday May 22, 2008

In "XIME-P 2008: Call for Demos and Participation", Dbworld
Page monitored for first time
...Kristoffer Rose, Jerome Simeon * XDM + SDO = XXDM: Getting Change Back From XDM by Vinayak Borkar, Michael Carey, Dmitry Lychagin, Radu Preotiuc-Pietro, Panagiotis Reveliotis, Joshua Spiegel, Till Westmann *...
http://www.cs.wisc.edu/dbworld/messages/2008-05/1211469424.html

Tuesday May 20, 2008

In "ACM SIGMOD Awards 2008", Dbworld
Page monitored for first time
...Sincerely, The 2008 ACM SIGMOD Awards Committee Peter Buneman Michael Carey Laura Haas David Maier Gerhard Weikum (2008 Chair)

**Related People**
- Torsten Grust
- Jens Teubner
- Jan Rittinger
- Ioana Manolescu

more

**Related Topics**
- search
- xml
- database systems
- optimization

more

**Services**
- VLDB 2006 (Panel chai
- ICDE 2005 (Seminar ch
- SIGMOD 2002 (Industri

**Publications**
- Inverse Functions in the Data Services Platform

Figure 4: Screenshots of the DBLife web portal.

tains an entity-relationship graph that describes the database community. It starts with domain knowledge and a list of sources. Domain knowledge includes a dictionary of entity names (conferences, universities, etc.), and dictionaries of equivalent names for entities (Robert/Rob/Bob; University of California, San Diego/UC San Diego/UCSD). Sources are the database community web pages where DBLife starts its crawl. The process of building the entity-relationship graph is performed by a series of modules that are organized into three layers: data page layer, mention layer, and entity layer.

The data page layer crawls the sources and saves local copies of the pages. It obtains metadata for each page, such as when it first appeared and when it last changed. Additionally, the data page layer detects structural elements in the pages, such as lists of names of people and conferences. The mention layer uses the dictionaries supplied by the domain expert to find mentions of those entities in the downloaded pages. Finally, the entity layer builds the entity-relationship graph. This series of modules is run daily. Both the mention layer and entity layer detect and track changes that occur between runs of DBLife. The entity-relationship graph is used by a web portal that displays news and information about the database community and provides information about the entities within the community. Screenshots of the web portal are shown in Figure 4.

We chose DBLife as an example program because DBLife uses a large, complex workflow that accesses many files. DBLife illustrates the types of questions that PAC can answer well, and types of provenance questions that PAC is not able to answer.

Examples of questions that PAC provides useful answers for are:

- Is this the version of the person name extraction program that was used in the creation of the current DBLife data set?
- Was this unreliable machine used in the creation of any part of the DBLife data set?
- Has DBLife crawled the most recent version of my web page?

Examples of questions that PAC cannot answer are:

- Why does the DBLife portal show that person X is affiliated with institution Y?
- Why doesn't my page on the DBLife portal include that I am on the program committee of a conference, even though the conference web page includes my name?

From these example questions we see that PAC is able to provide answers for the types of questions that developers and system administrators might ask about a run of DBLife. The types of questions that PAC cannot answer require knowledge about the semantics of the application. This result is not surprising because we know that PAC collects information about activities in the execution system, but does not collect information about the internal processes of the jobs that run in the system.

We measured the size of the FileTrace table for DBLife by executing a run of DBLife using 10 data sources (web crawl start points). This run produced a 100 megabyte FileTrace table. A full scale run of DBLife uses 1000 data sources. By scaling our test run up to full scale we obtain an estimated table size of 10 gigabytes. DBLife is scheduled to run on a daily basis so over the course of a year the FileTrace table would grow to 3.7 terabytes. This size borders on being too large for some users. We have not yet explored compressing the FileTrace table. It is likely that compression will result in a more manageable table size.

## 6 Conclusions

By instrumenting Condor with provenance awareness, we found that it is possible to transparently gather useful provenance with little impact on Condor. PAC provides provenance for a wide variety of applications. This allows users who do not have the resources or need for a specialized provenance system to obtain useful provenance. Our experiments show that PAC has reasonable storage requirements and does not have a noticeable performance impact on Condor.

One result of a provenance system that is able to accommodate a wide range of applications is that it does not gather some of the details that are gathered by provenance systems that require certain behaviors from applications. But we observed that workflow management systems can use PAC to run the individual workflow nodes. The combination of PAC and a more targeted provenance system provides more provenance details than either system does on its own.

## 7 Acknowledgments

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

## References

[1] BENJELLOUN, O., SARMA, A. D., HAYWORTH, C., AND WIDOM, J. An introduction to ULDBs and the trio system. *IEEE Data Engineering Bulletin, Special Issue on Probabilistic Databases 21*, 1 (March 2006), 5–16.

[2] BENT, J. *Data–Driven Batch Scheduling*. PhD thesis, University of Wisconsin–Madison, 2005.

[3] CONDOR. Project homepage, http://www.cs.wisc.edu/condor/, 2008.

[4] DEROSE, P., SHEN, W., CHEN, F., LEE, Y., BURDICK, D., DOAN, A., AND RAMAKRISHNAN., R. Dblife: A community information management platform for the database research community. In *CIDR-07* (2007).

[5] DOAN, A., RAMAKRISHNAN, R., CHEN, F., DEROSE, P., LEE, Y., MCCANN, R., SAYYADIAN, M., AND SHEN., W. Community information management. *IEEE Data Engineering Bulletin, Special Issue on Probabilistic Databases. 29*, 1 (2006).

[6] FREW, J., METZGER, D., AND SLAUGHTER, P. Automatic capture and reconstruction of computational provenance. *Concurrency and Computation: Practice and Experiment*, 20 (2008), 485–496.

[7] FREW, J., AND SLAUGHTER, P. ES3: A demonstration of transparent provenance for scientific computation. In *Provenance and Annotation of Data and Processes, Second International Provenance and Annotation Workshop, IPAW 2008* (Salt Lake City, Utah, USA, June 2008).

[8] HUANG, J., KINI, A., PAULSON, E., REILLY, C., ROBINSON, E., SHANKAR, S., SHRINIVAS, L., DEWITT, D., AND NAUGHTON, J. An overview of Quill: A passive operational data logging system for Condor. https://www.cs.wisc.edu/condordb, 2007.

[9] MILES, S., DEELMAN, E., GROTH, P., VAHI, K., MEHTA, G., AND MOREAU, L. Connecting scientific data to scientific experiments with provenance. In *Third IEEE Conference on e-Science and Grid Computing* (December 2007), pp. 179–186.

[10] MUNISWAMY-REDDY, K.-K., HOLLAND, D. A., BRAUN, U., AND SELTZER, M. Provenance-aware storage systems. In *Proceedings of the 2006 USENIX Annual Technical Conference* (Boston, Massachusetts, June 2006).

[11] REILLY, C. F., AND NAUGHTON, J. F. Exploring provenance in a distributed job execution system. In *Proceedings of the International Provenance and Annotation Workshop (IPAW)* (May 2006).

[12] SILVA, C. T., FREIRE, J., AND CALLAHAN, S. P. Provenance for visualizations: Reproducibility and beyond. *IEEE Computing in Science & Engineering 9*, 5 (2007), 82–89.

[13] STRACE. http://sourceforge.net/projects/strace/, 2008.