Exploring Provenance in a Distributed Job Execution System⋆

Christine F. Reilly and Jeffrey F. Naughton

University of Wisconsin–Madison
Department of Computer Sciences
1210 West Dayton Street, Madison, Wisconsin 53706, USA
{chrisr, naughton}@cs.wisc.edu

Abstract. We examine provenance in the context of a distributed job execution system. It is crucial to capture provenance information during the execution of a job in a distributed environment because often this information is lost once the job has finished. In this paper we discuss the type of information that is available within a distributed job execution system, how to capture such information, and what the burdens on the user and system are when such information is captured. We identify what we think is the key data that must be captured and discuss the collection of provenance in the Quill++ project of Condor. Our conclusion is that it is possible to capture important provenance information in a distributed job execution system with relatively little intrusion on the user or the system.

1 Introduction

Scientific computing applications are continuously growing in computational complexity and in the amount of data consumed and produced [1,2,3,4]. Many scientists utilize distributed job execution systems to meet their computational needs [5]. Within a distributed job execution environment much information is generated and exchanged regarding the execution and data access activities of the scientific application. This information can be used for tracking jobs through the system, recalling the activities of completed jobs, and for system accounting and debugging purposes. By archiving this information in a system that is visible to the user it can also be used to provide provenance information.

Our goal is to capture the provenance information that is available within a distributed job execution system. We specifically focus on the Condor system [5]; however, our discussion of the requirements for providing provenance in the context of a job execution system is applicable to the general category of distributed job execution systems. Condor is a distributed job execution system that runs on a dedicated cluster of machines, on idle desktop workstations, or on a combination of both environments [5,6]. This paper presents preliminary work on providing provenance information in Condor. In this work we explore

---

⋆ To be published in: Proceedings of the International Provenance and Annotation Workshop, May 3-5, 2006, Chicago, IL. In: Lecture Notes in Computer Science.

categories of provenance in the context of a distributed job execution system, examine what provenance data is available in Condor, and discuss how this data can be captured.

The provenance gathered in Condor is an important part of the overall provenance of data items used by the jobs run in the system. This provenance must be gathered while a job is running because it is likely to be unavailable once the job has completed. Condor users have expressed the desire for being able to obtain provenance information about their jobs. Scientists are notorious for frequently changing their data and executable programs and keeping the same file name across multiple versions. As a result, it is often very difficult for a scientist to determine exactly which version of their program was applied to which version of their data to produce a given output. A second provenance need is the ability to determine if a job is affected by a hardware problem. A number of years ago, Intel reported a bug in the floating point unit of one of its processors. When this happened users wanted to know if their jobs were run on machines with a faulty processor. Our provenance system provides the information needed to meet both of these provenance needs.

We identify two types of provenance in Condor: logical provenance and infrastructure provenance. In our context, logical provenance consists of the input data items and executable program that create an output data item. Infrastructure provenance for an output data item consists of information about when the item was created and what parts of the Condor system were involved in the creation of the data item. These two types of provenance are described in detail in Sect. 2.

The first issue we discuss in this paper is what type and amount of information can be captured in Condor. It is widely agreed that provenance is useful for scientific applications [2,4] and that, intuitively, a system should provide as much information as possible. Two factors inhibit us from collecting all possible information: some types of information are difficult to detect, and it is infeasible to store all possible information. In Sect. 2 we examine the information available in Condor and discuss the benefits and drawbacks of various levels of provenance we are able to achieve with this information.

The second issue we address in this paper is how to gather provenance information in Condor. There are two entities that have information: the system and the user. Ideally we would like to design a provenance system that is transparent to the user and has few alterations to and impacts on Condor. Because both the user and the system hold provenance information, we require some amount of intrusion into each entity in order to gather provenance information. In Sect. 3 we discuss the provenance information gained from various levels of intrusion on both entities. We then discuss the provenance capabilities of the Condor Quill++ project in Sect. 4.

2   Categories of Provenance Information

This section examines the provenance information available in a distributed job execution system. In order to understand the space of provenance information we

divide it into various categories. The first division is on the type of information stored in the system: logical or infrastructure. Within each of these two types we present divisions of level of reproducibility and granularity.

Data provenance in a distributed job execution system involves three entities: the job execution system, the user, and the provenance system. The job execution system runs user submitted jobs that perform the transformation from input data to output data. The user submits jobs and manages data items and transformation functions. The provenance system stores information about data items, infrastructure items, and instances of data transformations.

2.1   Logical Provenance

Logical provenance describes the input data and transformation process that create some specific output data. This is the type of provenance that is discussed in much of the related work [4,7,8,9,10,11,12,13,14,15,16]. We define the logical provenance of an output data item as the input data items and transformation function that produced the output data item. Because we are looking at the portion of provenance that is related to a distributed job execution system, we focus only on how data is manipulated within the system. We assume the transformation function is deterministic and free of side-effects. Therefore, given the same input data the transformation function will always produce the same output data. The two variables we identify for logical provenance are its granularity and level of reproducibility. Granularity describes the level of detail represented by a data item. The level of reproducibility of logical provenance is determined by the method the system uses for identifying data items.

The level of granularity for logical provenance describes the level of detail represented by a data item. The desired granularity level depends on how the provenance information will be used [4]. Additionally, the granularity level that a system can provide depends on at what level that system can uniquely identify and track single data items. Some examples of granularities are file, portion of file, database tuple, and byte. We expect that in most distributed job execution systems a granularity of file level can be easily achieved because that is the granularity level at which these systems generally manage data.

The level of reproducibility provided by a logical provenance system is determined by what information is stored in the system for each provenance item. In this discussion we assume that every provenance item has a unique identifier that is provided by either the user or the job execution system. We define three reproducibility levels: inform, verify, and redo.

A system with logical inform provenance can tell the user what provenance item identifiers (e.g., file names) are associated with a specific use instance. If the user can associate the identifiers with the corresponding items in her possession then the user can reproduce the use instance. The system stores the unique name of the provenance item and identifies how it was used (i.e., input, executable, output). Logical verify provenance extends logical inform provenance by determining whether a proposed job is identical to a previous job, meaning that the two jobs have the same input and executable files. Verify provenance

is stronger than inform provenance because it detects, for example, if the same identifier is used for files that have different content. We suggest using a checksum to probabilistically verify that data items are identical because storing the entire data item is likely to require a large amount of storage.

A system with logical redo provenance is able to rerun a previously submitted job. This system stores the entire provenance item (e.g., entire data files and executables) along with its use type. Although logical redo provenance is an intuitively desirable feature [17], we do not view this level of reproducibility to be practical in most cases. Because redo provenance requires the system to store every data item, the storage requirements for such a system could quickly become unreasonable. One case where logical redo provenance may be practical is if the provenance system and user's data storage system are integrated such that the provenance system and user are using the same data storage system [18,19].

2.2 Infrastructure Provenance

Infrastructure provenance information describes the environment involved in the creation of a data item. There are two reasons why infrastructure provenance is useful. First, if the creation of a data item is dependent on specific environment variables then these variables are important portions of the provenance of the data item. Second, if part of the infrastructure is found to be defective then data items that were created using the defective infrastructure can be identified. Infrastructure provenance consists of the two same variables as logical provenance: granularity and level of reproducibility. However, these variables have slightly different definitions for infrastructure provenance.

For infrastructure provenance the level of granularity describes what information about the environment is stored by the provenance system. One category is information about the environment that created the data, such as the creation date, specific processor, operating system, and amount of memory. A second category is the system state when the data was created, for example the general system load, and the contents of the memory and disk on the machine that created the data.

For most systems there is a set of infrastructure information that is relatively easy to obtain and is fairly useful. Examples of such information are: creation time, specific processor, operating system, amount of memory, and general system load. If at a later date a processor, or the memory or disk associated with a specific processor, is found to be defective then the data items created with that processor can be identified. We can also picture infrastructure information that is difficult to record or recreate, such as the computer registry or specific state of the memory. Additionally some infrastructure information, such as the compiler used by the transformation function, is found at the user level. Depending on how provenance information in communicated to the provenance system this user level information may or may not be available.

Infrastructure provenance has two levels of reproducibility: inform and redo. For both levels the provenance system records infrastructure information specific

to a data transformation instance. A system with inform provenance can tell the user what infrastructure items were used in the creation of a specific data item. With redo provenance the system can recreate a specific data item using the same infrastructure as originally created that data item. We expect that in most cases infrastructure inform provenance is sufficient and redo provenance is unnecessary.

3   Obtaining Provenance Information

In this section we address the question of how the provenance system obtains provenance information. As in Sect. 2 we assume that three entities are involved in data provenance: the provenance system, the job execution system, and the user. The provenance system must obtain provenance information from a combination of both the user and the job execution system. We assume that at a minimum the job execution system provides the provenance system with system infrastructure information related to a job.

We describe the trade-offs between the amount of provenance information gathered and intrusions on the system and the user with a cube where the amount of intrusion on the system is on the x-axis, the amount of intrusion on the user is on the y-axis, and the amount of provenance information is on the z-axis (Fig. 1). The range of each axis is 0 to 1. A job execution system that has no provenance capabilities is located at the (0,0,0) point. A system located anywhere on the back face of the cube, where the z-axis is equal to 1, collects all possible provenance information. The ultimate, and perhaps unachievable, goal is the (0,0,1) point, where all provenance information is provided with no intrusion on either the system or user. Our goal is a system that provides a large amount of provenance information while having small intrusions on both the user and the job execution system. The point in Fig. 1 labeled "Goal" is intended to loosely suggest a desirable location, where the cost of moving further back in the cube would require dramatic increases in the intrusion on the user and/or system.

We discuss three configurations of how information is provided to the provenance system: job execution system based, user based, and shared. For each of these configurations we discuss the feasibility of implementing the method and the reliability of that method for gathering the provenance information. The feasibility of a configuration refers to how likely we think it is that current systems could and would be altered in order to implement the method. A high feasibility means that it is very likely that the configuration could be implemented because it requires few or no changes to current systems. A low feasibility means that it is unlikely that the configuration could be implemented because it requires many or difficult changes to current systems. The reliability of a configuration describes how likely we think it is that the method will capture provenance information. We have greater trust in the system than in the user for providing accurate provenance information. Therefore a high reliability means that provenance information is fully provided by the system and a low reliability means that provenance information is fully provided by the user.
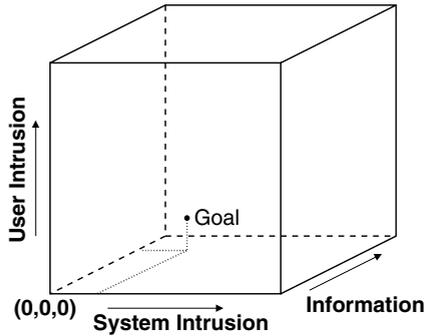
Fig. 1. Provenance Trade-offs Cube

When the job execution system provides all provenance information to the provenance system, the user can remain ignorant of the provenance system unless she requests provenance information. This scenario exists when the user intrusion equals zero on the provenance trade-off cube (Fig. 1). In such a scenario it is hard to capture all provenance information without intrusion on the system. For example, system intrusion is necessary for detecting access to files that are not declared in the job submission file. For such reasons we view a purely system-based approach to be of low feasibility.

If all provenance information is provided by the user then few to no alterations to the job execution system are necessary. This scenario exists when the system intrusion equals zero on the provenance trade-off cube (Fig. 1). We categorize this configuration of information gathering as high feasibility since few if any changes to the job execution system are necessary. However, this configuration has low reliability because we are completely depending on the user to provide accurate and complete information.

If both the user and the job execution system are aware of the provenance system then both can be relied upon for the gathering of provenance information. This is the scenario represented by the point labeled "Goal" in Fig. 1. In this case we rely on the job execution system to send messages to the provenance system. The user is required to be aware of the provenance system to enable the job execution system to send reliable messages to the provenance system. Exactly how the job execution system gathers provenance information and what the user must do depend on the structure of the job execution system.

4   Provenance in Condor

Quill++[20], an addition to Condor, was originally developed by the CondorDB team to provide better support for accounting and system monitoring, but we quickly realized that it could also be used to provide support for provenance. Quill++ writes information about machines, jobs, and workflows to logs then sniffs these logs and inserts the information into a central database. This ap-

proach allows Quill++ to make minimal changes to the Condor code and prevents Quill++ from blocking Condor. Quill++ has logical verify provenance capabilities and infrastructure inform provenance capabilities. Logical provenance information is stored at the granularity of files with the file identifier and checksum stored for each file. Infrastructure provenance includes information about machine hardware, software, and activity. Machine hardware information includes: processor identification, processor architecture, and amount of memory. Machine software information includes: operating system and Condor version. Machine activity information includes: if the machine is claimed by a Condor job, if the machine is idle, the time when Condor last heard from the machine, and statistics regarding machine load and activity.

Provenance information gathering is shared between the system and user. Information about jobs and machines is gathered from the Condor system by Quill++. File information is gathered from the job submission file. In order for Quill++ to obtain information about files the user must specify the file identifier and use type in the job submission file. It is possible that a job may use files that are not specified in the job submission file leaving Quill++ unaware of such files.

5   Conclusions and Future Work

We have shown that a good amount of provenance functionality can be achieved by storing information that is readily available within a distributed job execution system. For example, Quill++ stores information about the files used by a job, when and where the job ran, and some system state information. Quill++ imposes a minimal burden on the execution system and user, and provides what we hope is a useful amount of provenance information.

We have identified a number of items to explore in the future. Our first goal is to extend Quill++ to perform more system based gathering of provenance information by recording file information when Condor transfers files to the machine running a job. A second problem is to analyze the storage requirements of the provenance system in Quill++. Our preliminary analysis shows that in a cluster of thousands of machines the provenance portion of Quill++ generates a manageable amount of information over the period of one year. However, at some point in time the provenance information will need to be archived. Our third area of future work is to examine whether the provenance information regarding workflows must be explicitly recorded or if workflow provenance is recoverable from the provenance recorded for the component jobs.

Acknowledgments

References

1. Bose, R., Frew, J.: Lineage retrieval for scientific data processing: A survey. ACM Computing Surveys **37** (2005) 1–28
2. Jagadish, H., Olken, F.: Data management for the biosciences: Report of the NSF/NLM workshop on data management for molecular and cell biology, national library of medicine. Technical Report LBNL Report LBNL-52767, Lawrence Berkeley National Laboratory (2003)
3. Simmhan, Y.L., Plale, B., Gannon, D.: A survey of data provenance in e-science. SIGMOD Record **34** (2005) 31–36
4. Simmhan, Y.L., Plale, B., Gannon, D.: A survey of data provenance techniques. Technical Report IUB-CS-TR618, Computer Science Department, Indiana University, Bloomington, Indiana (2005)
5. Condor: Project homepage, http://www.cs.wisc.edu/condor/ (2006)
6. Tannenbaum, T., Wright, D., Miller, K., Livny, M.: Condor – A distributed job scheduler. In Sterling, T., ed.: Beowulf Cluster Computing with Linux. MIT Press (2001)
7. Buneman, P., Khanna, S., Tan, W.C.: Why and where: A characterization of data provenance. In: International Conference on Database Theory (ICDT). (2001)
8. Cui, Y., Widom, J.: Lineage tracing for general data warehouse transformations. In: Proceedings of the 27th VLDB Conference, Roma, Italy. (2001)
9. Cui, Y., Widom, J.: Lineage tracing for general data warehouse transformations. Technical report, Stanford University Database Group (2001)
10. Cui, Y., Widom, J.: Lineage tracing for general data warehouse transformations. VLDB Journal **12** (2003) 41–58
11. Fan, H., Poulovassilis, A.: Tracing data lineage using schema transformation pathways. In B.Omelayenko, Klein, M., eds.: Knowledge Transformation for the Semantic Web. IOS Press (2003)
12. Foster, I., Vockler, J., Wilde, M., Zhao, Y.: Chimera: A virtual data system for representing, querying, and automating data derivation. In: 14th International Conference on Scientific and Statistical Database Management. (2002)
13. Frew, J., Bose, R.: Earth system science workbench: A data management infrastructure for earth science products. In: Thirteenth International Conference on Scientific and Statistical Database Management, Fairfax, Virginia. (2001) 180–189
14. Widom, J.: Trio: A system for integrated management of data, accuracy, and lineage. In: CIDR. (2005)
15. Woodruff, A., Stonebraker, M.: Supporting fine-grained data lineage in a database visualization environment. In: Proceedings of the 13th International Conference on Data Engineering, Birmingham, England. (April 1997) 91–102
16. Cui, Y., Widom, J.: Storing auxiliary data for efficient maintenance and lineage tracing of complex views. In: Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW), Stockholm, Sweden. (2000)
17. Szomszor, M., Moreau, L.: Recording and reasoning over data provenance in web and grid services. In Meersman, R., Tari, Z., Schmidt, D.C., eds.: On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Volume 2888 of Lecture Notes in Computer Science., Springer (2003) 603–620
18. Barga, R.: Automatic generation of workflow execution provenance. In: International Provenance and Annotation Workshop (IPAW'06), Chicago, May 2006. (2006) http://www.ipaw.info/ipaw06.

19. Braun, U., Garfinkel, S., Holland, D.A., Muniswamy-Reddy, K.K., Seltzer, M.I.: Issues in automatic provenance collection. In: International Provenance and Annotation Workshop (IPAW'06), Chicago, May 2006. (2006) http://www.ipaw.info/ipaw06.
20. Huang, J., Kini, A., Reilly, C., Robinson, E., Shankar, S., Shrinivas, L., DeWitt, D., Naughton, J.: An overview of Quill++: A passive operational data logging system for Condor. https://www.cs.wisc.edu/condordb (2006)