Collaborative provenance for workflow-driven science and engineering

Altıntaş, İ.

Publication date
2011

8

Collaborative Provenance Database Implementation and Evaluation

"Everything that can be counted does not necessarily count; everything that counts cannot necessarily be counted."

– Albert Einstein

To validate the data model and queries presented in Chapter 6 and Chapter 7, we have implemented a collaborative provenance database in PostgreSQL based on a snapshot of the CAMERA Provenance Database. In this chapter, we explain the preparation of this database, the reasons for simplification of the CAMERA database, and an evaluation of the performance of the queries.

8.1 Database Implementation

8.1.1 CAMERA Workflows and Provenance Database

For the testing of the collaborative provenance database, we used the existing workflows in CAMERA (Sun *et al.* 2010, Altintas *et al.* 2010*a*) and the CAMERA Provenance Database (Altintas *et al.* 2010*c*) associated to the runs of these workflows. Currently, CAMERA supports 27 metagenomics workflows, including QC Filter, 454 Duplicate Clustering, different versions of BLAST, Gamma and Alpha Diversity (Rohwer), and RAMMCAP for Metagenomic data annotation and clustering. These workflows take metagenomics sequences (NT, protein, etc.) in one or more ($\sim$10) FASTA files, and can handle the reads (processing) of 1 million sequences. As illustrated by Figure 7.5, the CAMERA workflows are designed to fit together, allowing a user to pick a few of them and create her own methodical scientific

[0]This chapter is based on (Altintas *et al.* 2006*a*), (Altintas *et al.* 2010*c*) and (Altintas *et al.* 2010*f*) co-authored by Altintas.

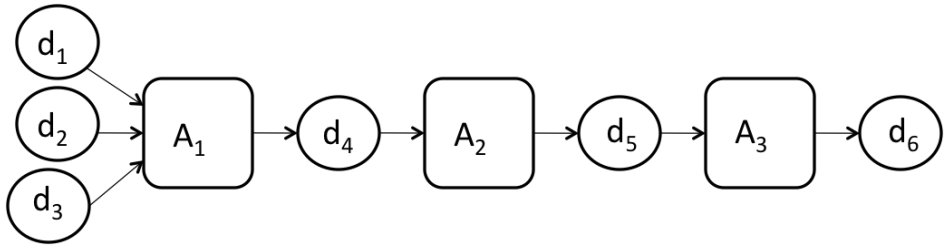


Figure 8.1: An example workflow execution; A_i illustrates Kepler's processing components (Actors), and d_i illustrates data.

process by executing the workflows of interest in the preferred order. To date, the CAMERA workflows have been executed using from a few thousand to hundreds of thousands of sequences as input over Sun Grid Engine-enabled resources. The provenance for each workflow execution is stored in an Oracle database. Over the past year (2009-2010), around 6000 workflow executions have been performed in the system, and the size of the provenance information for all workflow executions amount to around 3.7 GB.

With its large user base, diverse set of workflow executions and ever-growing data submissions, CAMERA is an ideal infrastructure for the testing of collaborative provenance model. Through a mapping tool (Altintas *et al.* 2010*c*) that was built to map workflow data identifiers to global data identifiers in CAMERA, the users can export workflow data (outputs) to the CAMERA database and workflows can exchange data with other workflows. However, the current CAMERA system is not ready for being used as it is for testing the collaborative usecase scenarios as discussed here. The current Kepler Provenance Schema[1] in CAMERA does not have complete data to answer the collaborative provenance queries. As mentioned in Chapter 6, the *ddep* and *ddep** tables should be generated in order to answer collaborative queries in addition to collecting system-level information about users, data they published or workflows they ran. Therefore, we based our database implementation on the actual runs and created a synthetic scenario based on these runs as discussed in the next section. The CAMERA usecase and the data model discussed in this thesis are being used as a basis for the design of collaborative provenance analysis framework in CAMERA.

8.1.2 Preparation of Collaborative Provenance Experimental Dataset

While Kepler's Provenance Schema keeps track of process-level data dependencies, collaborative provenance model requires dependencies to be captured (or inferred) at the workflow execution level. For instance, Figure 8.1 illustrates a workflow run with inputs d_1, d_2, and d_3, and the final output d_6. Through a set of Kepler API calls, the process-level dependencies can be determined as $\{(d_4, d_1), (d_4, d_2), (d_4, d_3), (d_5, d_4), (d_6, d_5)\}$. However, the collaborative provenance data model needs a mapping of these process-level dependencies to

[1]The latest Kepler Provenance Schema is explained at:
https://code.kepler-project.org/code/kepler/trunk/modules/provenance/docs/provenance.pdf.

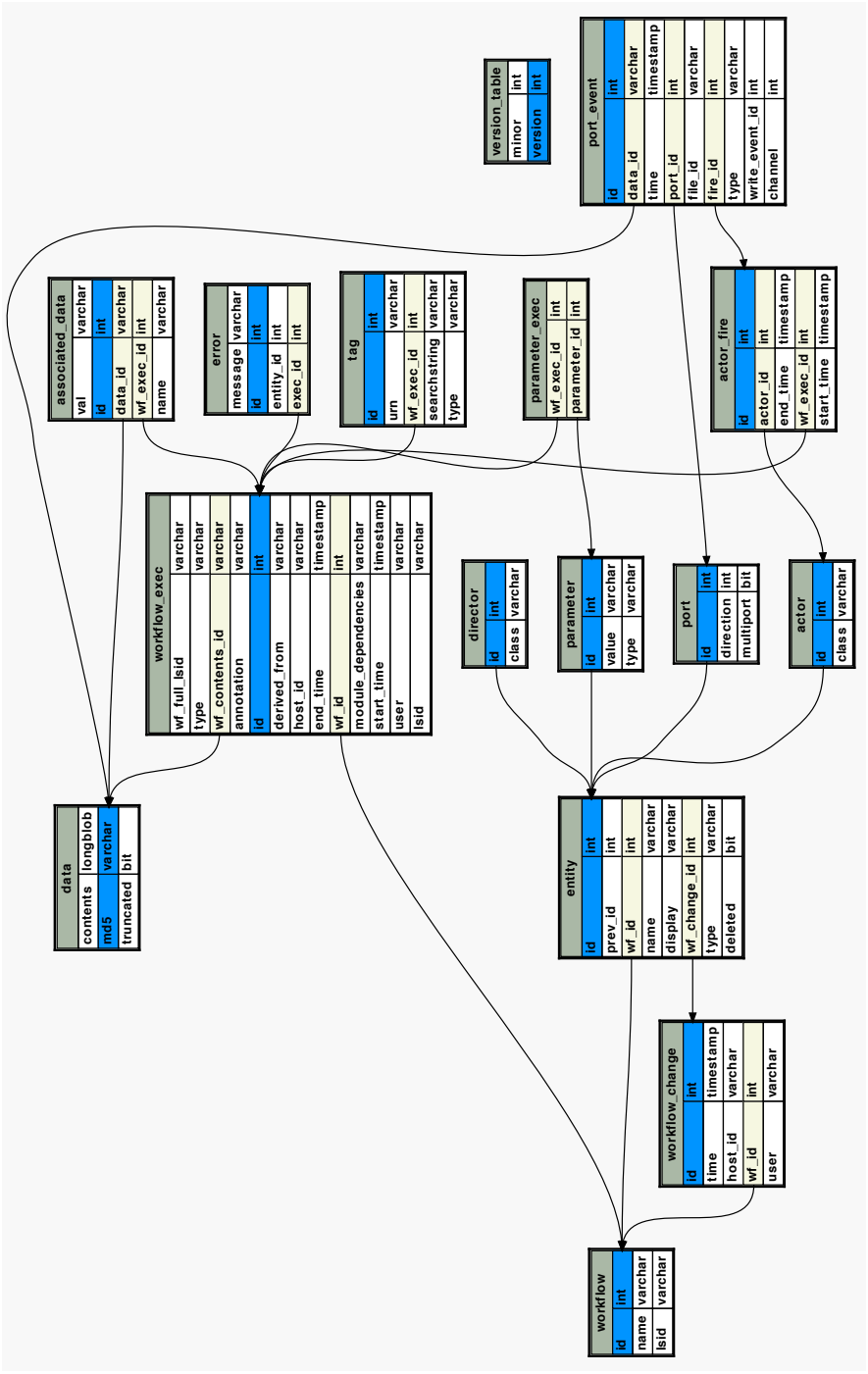


Figure 8.2: A snapshot of the tables in the Kepler Provenance Database Schema.

workflow run-level dependencies, i.e., dependencies between initial inputs and final outputs of a workflow (*ddep*). This relationship is described in the *ddep* table in our schema with the attributes $data_{to}$, and $data_{from}$. Thus, the *ddep* table associated to the scenario in Figure 8.1 consists of $\{(d_6, d_1), (d_6, d_2), (d_6, d_3)\}$.

Figure 8.2 depicts the tables in the relational schema of the Kepler Provenance Database. Using the CAMERA database based on this schema, we retrieve the data for collaborative provenance from four tables, namely, *workflow*, *workflow_exec*, *actor_fire*, and *port_event*. *workflow* and *workflow_exec* provide the *w* for the workflow(w, u) table and *r* in the run(r, w, u) table in the collaborative provenance schema. The remaining two tables (*actor_fire* and *port_event*) provide the direct data dependencies, i.e., data for ddep(dto, dfrom), in the collaborative provenance schema. The table *actor_fire* records information about actor firings for a particular actor (*actor_id*) in a particular workflow execution (*wf_exec_id*). When a workflow is executed, there are many intermediate inputs and outputs. Multiple components (actors) in the workflow might produce intermediate outputs as intermediate inputs for other components before the final output(s) is produced. The actors that process the data and produce the output(s) for a particular workflow have the same foreign key (*wf_exec_id*) in the *actor_fire* table. However, for populating the collaborative provenance database, we are only interested in the data dependencies between the initial inputs and final outputs of the particular workflow run. In order to retrieve these run-level input and output data dependencies, we need to determine which data among the data processed by actors are the initial inputs and final output(s). In this case, *port_event* table records the read or write event of a particular actor whenever the actor fires. Each token[2] read or write is stored as a row in this table. A port event occurrs at a time, on port *port_id*, and on channel from actor firing *fire_id*. The token's value is referenced by *data_id*. If the data is a file, a reference to the contents of the file is in *file_id*. If the port event represents a read, *write_event_id* is the *port_event.id* of the port event that generated the token, otherwise (port event is a write) *write_event_id* is -1.

For the transformation of this existing process-level data into a workflow run-level dataset that conforms to the designed collaborative provenance data model, we implemented scripts to infer run level data dependencies from process level data dependencies in the CAMERA provenance database. The algorithm we followed in these scripts to retrieve records for *ddep* table is as follows:

1. Retrieve all the tokens involved in a workflow execution.

2. Retrieve all the actors involved in this specific run.

3. For each token, determine whether there has been a *write* event from an actor that processed it. If there has not been a *write* event, which processed the token, then it is the initial input. In Figure 8.1, data tokens d_1, d_2, and d_3 do not have an actor that processed a *write* event to these tokens. Therefore, they are initial inputs.

[2] Actor inputs and outputs are wrapped as tokens in Kepler.

4. Also for each token, determine whether there has been a *read* event from an actor to process the token. If there is no *read* event, which processed the token, then it is the final output. In Figure 8.1, data token d_6 is the final output since there was no read event that processed it.

To make the query execution faster, we decided to compute and materialize the transitive closure of *ddep* relation (*ddep**). In addition, since the Kepler Provenance Schema does not have information about user specific actions through the CAMERA portal (publishing data and workflows, executing workflows), we created a mapping to capture this information from the CAMERA database.

8.1.3 Implementation

The collaborative provenance schema was implemented as a PostgreSQL database. We retrieved fifty workflow executions from the CAMERA provenance database and determined the run-level data dependencies (*ddep*) for each workflow using the steps described in the previous subsection. After inserting the retrieved data in the *ddep* table, we used the WITH RECURSIVE function in PostgreSQL to compute the transitive closure on the *ddep* table and populated the result in *ddep** table. After all the necessary data for the collaborative schema is inserted in the new database, we expressed and ran all the queries and views in SQL against the PostgreSQL database. Table 8.1 shows the queries for collaborative provenance views in SQL, and Table 8.2 and Table 8.3 show the example queries and the query expressions in SQL. All of these queries were executed in PostgreSQL for the evaluation of the model.

To measure the scalability of the implementation, we gradually increased the datasets by having the run-level dependencies expanding from 5 to 10, 25, and 50 workflow runs. Table 8.4 shows the number of rows in each table of the experimental database for each increment. Note that DB_5 in Table 8.4 matches the example scenario provided in Figure 7.4 with an extra workflow run, and the rest of the database is populated similarly to expand the run and corresponding data dependencies.

A preliminary implementation of an online collaborative provenance browser based on HTML5 and CSS3 is currently under development. The browser provides three different querying interfaces for visualizing data dependencies, run dependencies and user collaborations. The implemented database is queried based on the parameter selections by the users.

8.2 Evaluation

In this section, we present a short evaluation of the proposed collaborative provenance model and the implemented PostgreSQL database. This evaluation has two primary goals:

1. A validation of the possibility of implementing the proposed data model in correctly answering collaborative queries.

Table 8.1: Collaborative provenance views for data dependency, run dependency and user collaboration expressed in PostgreSQL.

DATA-DEP	SELECT * FROM ddep_star ;
RUN-DEP	SELECT t.run, f.run FROM ddep AS d INNER JOIN produces AS f ON d.data_from = f.data JOIN produces AS t ON d.data_to = t.data ;
USER-COLLAB	CREATE VIEW userCollab AS SELECT r.executed_user AS uto, w.id AS e, w.published_user AS ufrom FROM workflow AS w INNER JOIN run AS r ON w.id = r.workflow_id UNION SELECT r.executed_user AS uto, u.data AS e, p.users AS ufrom FROM run as r INNER JOIN uses AS u ON r.id = u.run JOIN publishes AS p ON u.data = p.data UNION SELECT r1.executed_user AS uto, u.run AS e, r2.executed_user AS ufrom FROM run as r1 INNER JOIN uses as u ON r1.id = u.run JOIN produces as p ON u.data = p.data JOIN run as r2 ON p.run = r2.id ;

Table 8.2: Example CAMERA queries Q1 through Q6 expressed in PostgreSQL.

Q1	Which data artifacts were used directly or indirectly to generate data with id 194119?	SELECT data_from FROM ddep_star WHERE data_to = 194119
Q2	Which runs were used in the generation of data with id 183215?	SELECT run FROM ddep_star INNER JOIN produces ON data=data_from WHERE data_to = 183215
Q3	If data artifact with id 49774 is detected to be faulty, which users should be notified of the error?	SELECT distinct (executed_user) FROM ddep_star AS d INNER JOIN produces AS p ON d.data_to= p.data JOIN run AS r ON p. run = r.id WHERE d.data_from = 49774
Q4	What are all the datasets that depended on data artifact with id 49774, i.e. the *"impact"* of 49774?	SELECT data_to FROM ddep_star WHERE data_from = 49774
Q5	Which users depended on data artifact 49775, directly or indirectly?	SELECT distinct(executed_user) FROM ddep_star INNER JOIN produces ON data_to=data JOIN run ON run = id WHERE data_from= 49775
Q6	Which users did user with id 11111 depend on, i.e., *"collaborate with"*, directly? What is the nature and strength of each collaboration?	SELECT uto FROM userCollab WHERE ufrom=11111; SELECT e FROM userCollab WHERE ufrom= 11111; SELECT uto, e, COUNT(e) FROM userCollab WHERE ufrom= 11111 GROUP BY (e,uto);

2. An analysis of the changes in the cost of the collaborative views and example queries over an increasing number of run and data dependencies.

Table 8.3: Example CAMERA query Q7 expressed in PostgreSQL.

Q7	Who are the potential acknowledgements for a publication involving data with id 194119, i.e., which user collaborations were involved in the derivation of data with id 194119?	SELECT distinct (executed_user) FROM produces INNER JOIN run ON run=id WHERE data = 194119; SELECT distinct (w.published_user) FROM produces AS p INNER JOIN run AS r ON p. run = r.id JOIN workflow AS w ON r.workflow_id =w. id WHERE data = 194119; SELECT distinct(executed_user) FROM ddep_star INNER JOIN produces ON data_from = data JOIN run ON run = id WHERE data_to = 194119; SELECT distinct(users) FROM ddep_star INNER JOIN publishes ON data_from = data WHERE data_to = 194119; SELECT distinct (w.published_user) FROM ddep_star AS d JOIN produces AS p ON d.data_from = p.data JOIN run AS r ON p.run = r.id JOIN workflow AS w ON r.workflow_id = w.id WHERE data_from = 194119;

For the evaluation, we executed the collaborative provenance views, and the example queries on datasets DB_5 through DB_50 to measure the feasibility and effectiveness of our implementation. Table 8.6 shows the execution times for collaborative provenance views, DATA-DEP, RUN-DEP and USER-COLLAB, along with the execution times for queries Q1 through Q7. The columns represent the query response time in milliseconds for SQL queries over PostgreSQL and the rows represent the datasets that were used to run these queries. Note that the execution time for Q6 and Q7 show the sum of the execution times for each sub-query, specifically, three sub-queries for Q6 and five sub-queries for Q7.

Table 8.4: The size of database (in number of tuples) for different datasets.

Table	DB_5	DB_10	DB_25	DB_50
users	5	10	25	50
workflow	5	10	25	50
run	5	10	25	50
data	11	16	31	56
publishes	6	6	6	6
uses	9	14	29	54
produces	5	10	25	50
ddep	12	17	32	57
ddep*	17	52	307	1232

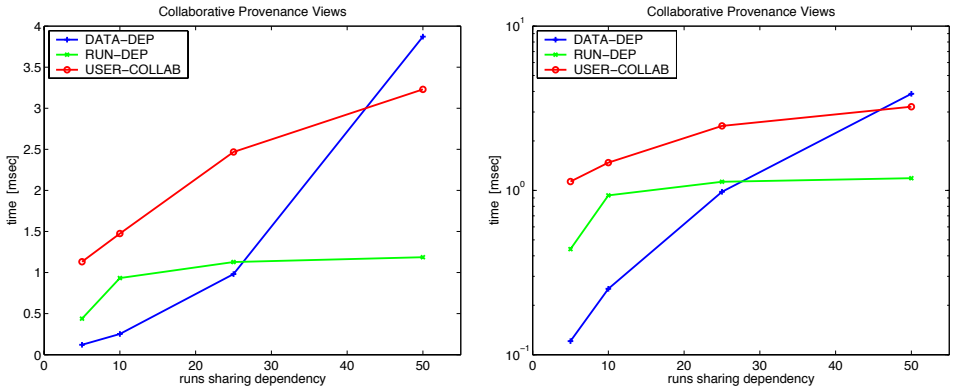
Table 8.5: The query execution time (in ms) for collaborative provenance views over datasets of 5, 10, 25 and 50 run dependencies.

	DATA-DEP	RUN-DEP	USER-COLLAB
DB_5	0.121	0.439	1.132
DB_10	0.252	0.933	1.475
DB_25	0.981	1.128	2.468
DB_50	3.872	1.187	3.230

Table 8.6: The query execution time (in ms) for evaluation queries 1 through 7 over datasets of 5, 10, 25 and 50 run dependencies.

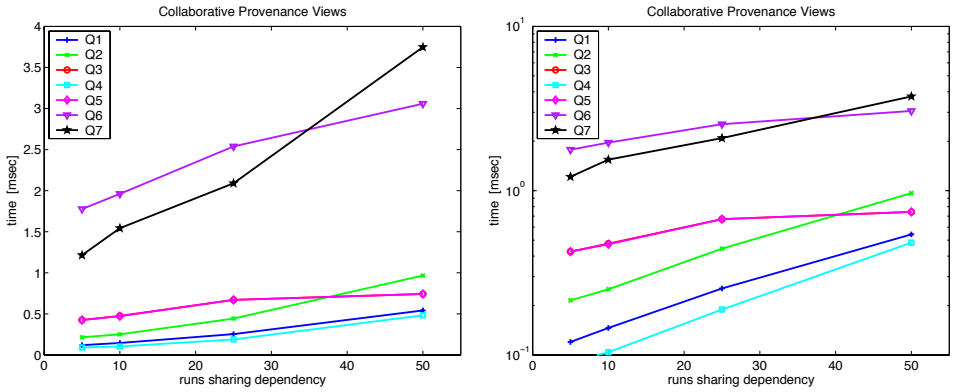
	Q1	Q2	Q3	Q4	Q5	Q6	Q7
DB_5	0.12	0.215	0.426	0.091	0.426	1.777	1.215
DB_10	0.146	0.251	0.476	0.104	0.472	1.961	1.544
DB_25	0.254	0.444	0.671	0.189	0.671	2.539	2.090
DB_50	0.542	0.967	0.743	0.483	0.743	3.058	3.748

Figure 8.3(a) shows the query response time for data dependency (DATA-DEP), run dependency (RUN-DEP), and user collaboration (USER-COLLAB) views on the datasets DB_5 through DB_50 on a linear time scale. Although the execution time for the DATA-DEP view looks like it grows exponentially, Figure 8.3(b) on a logarithmic scale shows that all three



(a) (b)

Figure 8.3: Query execution time cost for data dependency view, run dependency view and collaboration view in: (a) linear time scale, and (b) logarithmic time scale. 5, 10, 25 and 50 indicate the number of run dependencies in the dataset.



(a) (b)

Figure 8.4: Query execution time cost for evaluation queries 1 through 7 in: (a) linear time scale, and (b) logarithmic time scale. 5, 10, 25 and 50 indicate the number of run dependencies in the dataset.

queries scale linearly. However, the response time clearly shows that the execution time for DATA-DEP view is effected by the large number of datasets that share dependency relationships, as expected.

Similarly, Figure 8.4 is a plot of the query times for example queries Q1 through Q7. Q6 and Q7 take a longer time compared to queries Q1 through Q5 as expected since they are combined (added) cost of multiple sub-queries. Although the linear time scale in Figure 8.4(a) looks like the query execution times do not grow too fast, a plot of the data on a logarithmic scale in Figure 8.4(b) shows that the execution times for test queries grow exponentially with increasing number of data and run dependencies except for the execution times for queries

Q5 and Q6.

Through this database implementation, we have shown that the proposed model can be implemented over a larger number of runs relative to the example scenario. The model is able to answer the collaborative provenance views and example queries in a reasonable time for our experimental datasets. Although the data dependencies in the dataset were not very complicated, the evaluation demonstrates that as the number of data dependencies increase the queries that rely on these dependencies take longer. We identified the views and queries that are taking longer time as potential queries to be optimized, which we endeavor to do as part of our future work as discussed in the conclusions of this thesis. We also plan to expand the evaluation dataset with more dependencies.

Summary

We provided an initial implementation of the relational collaborative provenance data model in PostgreSQL. Using the CAMERA workflow executions and provenance database as a basis, we executed and evaluated the example queries. Next, we discuss the technical challenges and our existing work related to interoperability in collaborative provenance scenarios involving scientific workflows implemented in different systems.