Parallel Provenance Databases for High Performance Workflows

Jennifer A. Steffens
Drake University
Des Moines, IA
jennifer.steffens@drake.edu

***Abstract*--In scientific computing, understanding the origins and derivation of data is crucial. Provenance models aim to provide a means of capturing this in an efficient and effective manner. For the Swift/T language, the current provenance handling system requires improvement. In this poster, we discuss the development of a new Swift/T provenance model, the Multiple Parallel Databases Model (MPDM), which parallelizes the real-time storage of provenance data in a user-accessible database system. Utilizing multiple databases in high performance, parallel workflows can increase the practicality of lightweight, relational databases engines such as SQLite, as we show MPDM to be more efficient and have better scalability than the previous, single database model.**

I. THE SWIFT SYSTEM

The Swift scripting language is designed for optimizing the execution of scientific computational experiments by performing independent tasks implicitly in parallel. The system operates on a given number of nodes, with one server node and many script-executing worker nodes. Swift/T, the current implementation of the language, utilizes MPI and ADLB libraries in its runtime, Turbine, letting it perform up to 1.5 billion tasks per second [1].

The provenance model for the previous generation of the Swift language, Swift/K, is based on the Open Provenance Model. After the execution of parallel scripts that specify many-task computations, this model extracts provenance information from the log files that Swift/K generates and stores it in a SQLite-driven database [2]. SQLite is simple and lightweight, with the advantage of being already present on most machines and leaving an extremely small memory footprint.

II. OUR APPROACH

Instead of upon program termination, our model collects provenance data during runtime, providing the benefit of access before a large workflow finishes executing and easy progress tracking. To achieve this, we integrated an SQLite-utilizing C program into the Swift/T source code that inserts information into the databases as it is processed. In this system, the two largest and data-intensive tables are **ApplicationExecution**, which details external application calls (leaf tasks) in a Swift script, and **ScriptRun**, which details important general information [Fig. 1].

ApplicationExecution	
tries	int
startTime	datetime
try_duration	int
total_duration	int
command	char (128)
stdios	char (128)
arguments	char (128)
notes	text
tries	int

ScriptRun	
scriptRunId	int
scriptFileName	datetime
logFileName	int
swiftVersion	int
turbineVersion	char (128)
finalState	char (128)
startTime	char (128)
duration	char (128)
scriptHash	text
scriptRunId	int

Fig. 1. Schema of ScriptRun and ApplicationExecution. These are modified versions of tables in the Swift/K model.

We assigned each worker its own database, which is schematically identical to the master database. By doing this, we can make sure each database is not being written to by multiple processes simultaneously [Fig. 2]. To query the data, we use attach statements to join all of the separate database files, eliminating the need to combine them into a single file.

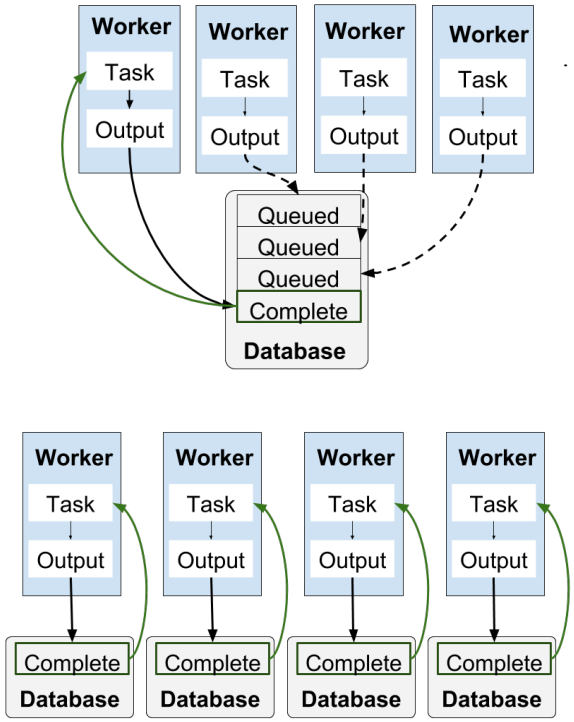

Fig. 2. A visualization of the single database system (top) and the Multiple Parallel Databases Model (bottom).

III. EVALUATION

To test the efficiency of our system, we ran a simple Swift script thrice per trial on a variable amount of nodes hosted by the Cooley computing system, a collection of 126 compute nodes housed at Argonne National Laboratory that is composed of Two 2.4 GHz Intel Haswell E5-2620 v3 processors per node (6 cores per CPU, 12 cores total), and has 384GB RAM per node. Our script generated a hundred tasks per worker node, and assigned them each their own database. We compared this to the same script ran with a single database and found significant increases in performance for the multiple database models [Fig. 3].

The unpopulated multiple database model executed a creation statement for each database and this caused it to not show scalability long term. However, in practical use, the databases will only need to be created in the first script execution, and the execution of any script following will only add to them. Therefore, the populated models, which were shown to scale, portray realistic use. The 12 processes per node model gave our maximum efficiency of 432.654 tasks per second, at 8 worker nodes and 800 tasks. It continues to be the most efficient model, but when executed on more than 60 nodes, it mimics the behavior of its single process per node counterpart.

IV. CONCLUSION

Our model provides a significant improvement in speed, with the maximum observed efficiency of the Multiple Parallel Databases Model being one hundred times that of the

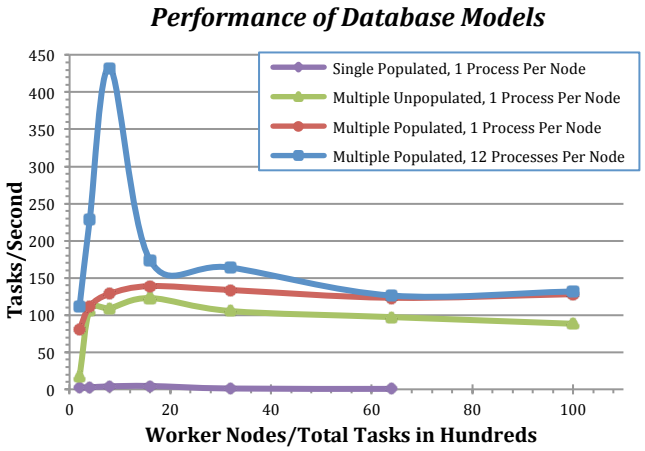

Fig. 3. A visualization comparing the speed of the systems.

previous model. The efficiency of this model also increases as more worker nodes are added, mimicking the behavior of the Swift language.

Since provenance data is viewable at runtime, researchers can now analyze output as soon as it is collected and observe its change in real time. This aids in tracing output, identifying errors, and accelerating program improvement and efficiency. In addition, since the method of use of the database engine rather than the engine itself is modified, this model can be applied to other database engines to improve performance.

We believe parallelizing databases in this fashion will make simple database engines practical for high performance computing. For the Swift/T language, the Multiple Parallel Databases Model offers easy storage and access to valuable data collected, available as soon as it is processed.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under awards NSF-1461260 (REU) and used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE- AC02-06CH11357. This work was supervised by Justin Wozniak.

REFERENCES

[1] Wozniak, Justin M., Timothy G. Armstrong, Michael Wilde, Daniel S. Katz, Ewing Lusk, and Ian T. Foster. "Swift/T: large-scale application composition via distributed-memory dataflow processing." In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pp. 95-102. IEEE, 2013.

[2] Gadelha, L. M., B. Clifford, M. Mattoso, M. Wilde, and I. Foster. *Provenance management in Swift with implementation details*. No. ANL/MCS-TM-311. Argonne National Laboratory (ANL), 2011.

[3] Gadelha Jr, Luiz MR, Michael Wilde, Marta Mattoso, and Ian Foster. "MTCProv: a practical provenance query framework for many-task scientific computing." *Distributed and Parallel Databases* 30, no. 5-6 (2012): 351-370.