Parallel Provenance Databases for High Performance Workflows

Jennifer A. Steffens, *Drake University*

Abstract

- Swift/T's current provenance system requires improvement.
- The new provenance model, the **Multiple Parallel Database Model (MPDM),** parallelizes the real-time storage of data in a user-accessible database system.
- MPDM shows significant improvement in many ways over the previous model.

The swift System

- Swift is a scripting language designed to parallelize high performance workflows in order to optimize efficiency.
- It is composed of one server node and many worker nodes.
- The current implementation, Swift/T, Utilizes MPI and ADLB libraries in its runtime, Turbine.
- Swift/T can perform up to 1.5 billion tasks per second, improved from the Swift/K maximum of 500 tasks per second [1].
- The Swift System's current provenance model, which aims to collect metadata concerning a program's execution, outputs to a log text file, then parses and inserts information into a SQLite database upon program termination [2].

A New Approach

- MPDM collects provenance data during runtime.
- Many processes writing to a single database at the same time can be slow, corrupt data, or kill the processes [Figure 1].

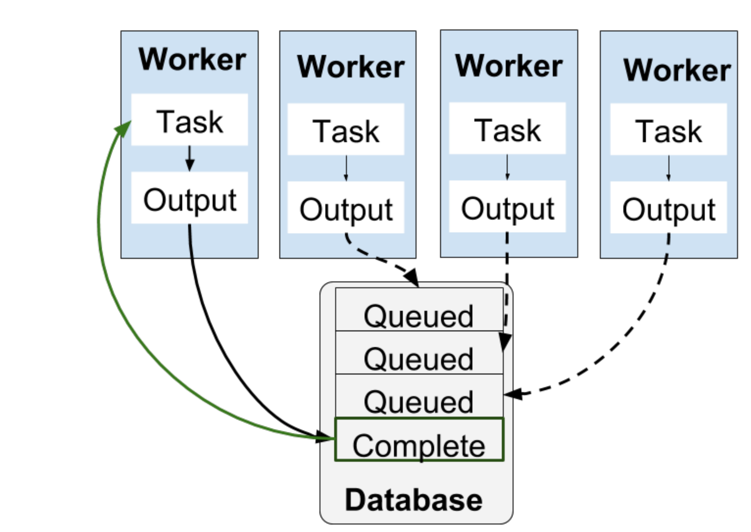


Figure 1: A visualization of the previous database model

- MPDM Solves this by assigning each worker node a separate, schematically identical database [Figure 2].
  - Data is queried by attaching the databases to each other, eliminating the need to combine the files.

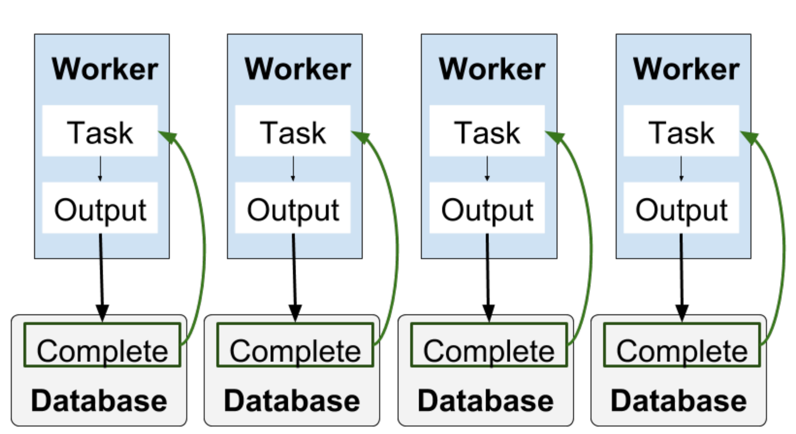


Figure 2: A visualization of the Multiple Parallel Database Model

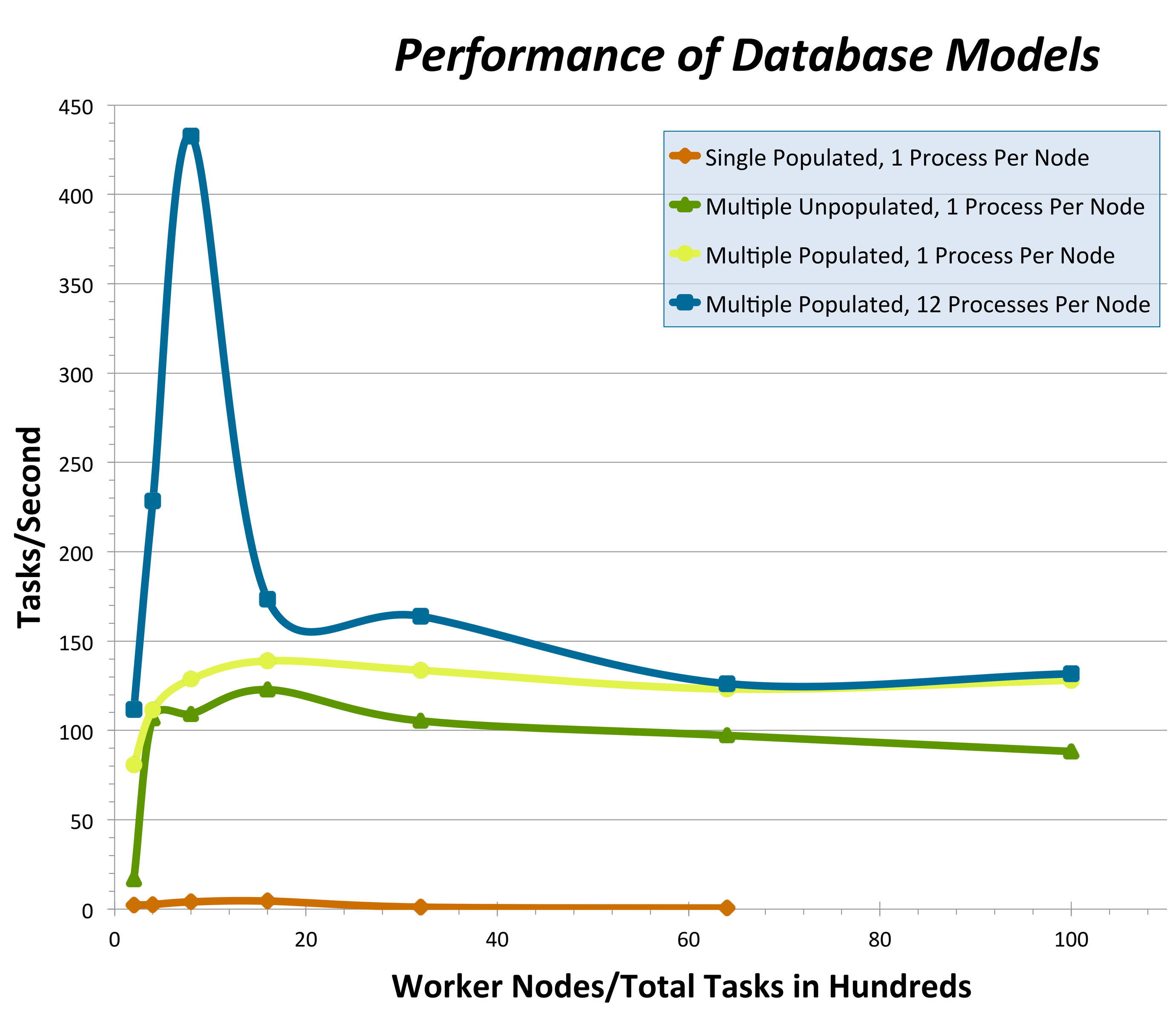
Evaluation

- SQLite database operations were implemented in C and integrated into the Swift/T source code.
  - Data was inserted into two major tables during runtime [Figure 3].
- A Swift/T script was run that generated 100 tasks for each worker node.
  - The speeds of a multiple database system to that of a single database system were compared [Figure 4].
  - Unpopulated database models executed create statements to set up the pre-determined schema.
  - Populated models added to existing databases.
  - Each model was given three separate trials for each node count, and the average speed was recorded.
- The program was executed on Cooley, a 126-node supercomputer hosted by Argonne National Laboratory.
  - Two 2.4 GHz Intel Haswell E5-2620 v3 processors per node (6 cores per CPU, 12 cores total).
  - One NVIDIA Tesla K80 (with two GPUs) per node.
  - 384GB RAM per node, 24 GB GPU RAM per node (12 GB per GPU).

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

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

Figure 3: Schema of ApplicationExecution and ScriptRun



- The single database model showed to be significantly less efficient.
- The single database model and the unpopulated multiple model showed decreasing end behavior, implying a bottleneck.
- Both populated multiple models showed increasing end behavior implying scalability.
- The 12 PPN populated model showed the most efficient for 60 workers and less and then showed similar efficiency to its 1 PPN counterpart.

Worker Nodes/ Total Tasks	Single Populated 1 Process Per Node	Multiple Unpopulated 1 Process Per Node	Multiple Populated 1 Process Per Node	Multiple Populated 12 Processes Per Node
2	2.2172	16.785	80.667	111.669
4	2.594	106.8	111.592	228.506
8	4.056	109.27	128.673	432.654
16	4.506	122.822	138.925	173.461
32	1.085	105.351	133.662	163.913
64	0.5926	97.207	123.193	126.203
100	N/A	88.235	128.084	131.73

Figure 4: Comparison of the efficiency of database systems

Conclusions

- **Significant increase in speed**: The maximum observed efficiency of MPDM is one hundred times that of the previous model.
- **Long term scalability**: Efficiency increases as more worker nodes are added, mimicking the behavior of the Swift language.
- **Accessibility:** With provenance data being 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.
- **Flexibility:** Since the method of use of the database engine rather than the engine itself is modified, this method can be applied to other database engines to improve their performance as well.

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.

Acknowledgements

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 of Argonne National Laboratory.

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.