Automated Data Partitioning for Independent Distributed Transactions

Alexandru Turcu, Roberto Palmieri, Binoy Ravindran
Department of Electrical and Computer Engineering, Virginia Tech, Virginia, USA
{talex, robertop, binoy}@vt.edu

A framework for exploiting independent transactions in distributed transactional store.

- Distributed transactional storage systems nowadays require increasing isolation levels, scalable performance, fault- tolerance and a simple programming model for being easily integrated with transactional applications.
- Cowling and Liskov in recently proposed a new model for scalable distributed transactional storage, called Granola, in which transactions are processed locally and only one round of communication between the user and system is needed.
- Additionally, this approach exploits the presence of multiple machines, where possible, to process independent portions of the same transaction in parallel on different nodes, in order to overlap their computation (named independent transactions).

Main Challenge

- **Exploiting the Granola model advantages on general-purpose distributed transactional applications.**

Granola model's drawbacks

- Even though the Granola model is appealing, its assumptions restrict its applicability.

1. Data needs to be well partitioned such that transactions can be executed according to the independent transaction paradigm, namely split into multiple execution flows running in parallel on different partitions.
2. Users executing the transactions require a-priori knowledge of partitions accessed by submitted transactions for contacting the right nodes.

Contributions. We extend an existing graph-based data partitioning algorithm, Schism, to be compatible with the additional insights and requirements of the Granola model. Additionally, we solve the problem of routing transactions, developing a new efficient mechanism based on machine learning that overcomes the lack of knowledge about the nodes storing objects. We also provide simple transactional abstractions to the programmer for interfacing with the distributed system, making our proposal easily pluggable for existing applications.

Two phases: static analysis and run-time support.

Partitioning Process

- The first phase in our partitioning workflow performs static analysis and byte-code rewriting on all transactional routines in the workload. With this step, it collects data dependency information which is later used to ensure the proposed partitioning schemes are able to comply to our chosen one-round transactional model.
- The second phase is collecting a representative trace for the current workload, which includes a record for every transactional operation performed. Each record contains the transaction identifier, the type of operation, the affected object, and the unique identifier of the operation.
- The next phases are similar to the corresponding phases in Schism. The graph representation phase processes the previously collected workload trace and creates a graph where nodes represent objects and edges represent transactions.
- The final phase is concerned with transaction routing and model selection. We introduce a machine-learning based routing phase. The data used to train this classifier is derived from the workload trace, using the object-to-partition mapping. Finally a transaction model is selected for every transaction class depending on the number of partitions it executes on.

Run-Time Behavior

- The first set of classifiers is tasked with object-to-partition mapping. These classifiers determine the object placement. While it may reduce the quality of the resulting partitions, misclassification at this stage is mostly harmless, since it is the classifier that dictates the final object placement.
- The second set of classifiers are the routing classifiers. They are used on the client side to decide which nodes to contact for the purpose of executing the current transaction. Inputs for these classifiers are the parameters passed to the transaction. A misrouted transaction may not have access to all objects needed to execute successfully. In this case we allow such misrouted transactions to abort and restart on a larger set of nodes.
- Users are not aware of the partitioning scheme or the transaction execution model when writing transaction code. Thus, they should be able to write a single atomic block, and the system would make sure the appropriate code branches will execute at the corresponding partitions.

Evaluation

Configuration

- Our implementation is based on Hyflow2, a JVM-based DTM framework written in Scala.
- We evaluate our partitioning process using TPC-C as in the original Granola.
- The workload was configured with between 3 and 15 warehouses, and traces of different lengths were obtained.
- The trace used in the experiment contains approximately 1200 transactions.
- We used three classifier types for both object placement and transaction routing. We find that using *C4.5 decision trees* for placement and routing gives the best results.

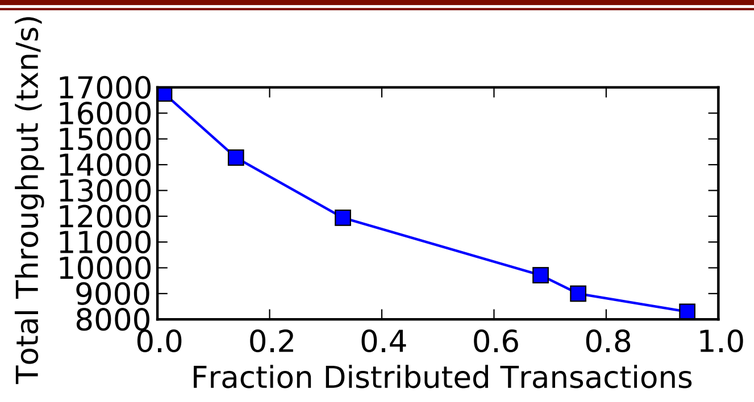

Figure shows transactional throughput of a 3-warehouse workload, partitioned across three repositories. Throughput is high, comparable with the original Granola implementation, where partitioning and routing is done manually.

Finally…

We have developed a methodology for using automatic data partitioning in a Granola-based distributed transactional memory. Independent transactions are powerful and guarantee high-performance but they have a limited applicability. Our proposal seeks to overcome this lack.

ACKNOWLEDGMENTS

This work is supported in part by US National Science Foundation under grants CNS 0915895, CNS 1116190, CNS 1130180, and CNS 1217385.