ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

Automated Data Partitioning for Highly Scalable and Strongly Consistent Transactions

Sujit Shinde[1], Mahesh Doiphode[2], Channappa Mirgale[3], Nived Kolhe[4], R. D. Bharti[5]

Student, Dept. of CSE, DYPIET, Pimpri, Pune, Savitribai Phule Pune University, Pune India[1234]

Professor, Dept. of CSE, DYPIET, Pimpri, Pune, Savitribai Phule Pune University, Pune India[5]

ABSTRACT: Modern transactional processing systems need to be fast and scalable, but this means many such systems settled for weak consistency models. It is however possible to achieve all of strong consistency, high scalability and high performance, by using fine-grained partitions and light-weight concurrency control that avoids superfluous synchronization and other overheads such as lock management. Independent transactions are one such mechanism that rely on good partitions and appropriately defined transactions. On the downside, it is not usually straightforward to determine optimal partitioning schemes, especially when dealing with non-trivial amounts of data. Our work attempts to solve this problem by automating the partitioning process, choosing the correct transactional primitive and routing transactions appropriately.

KEYWORDS: Distributed databases, code generation, concurrent programming

I. INTRODUCTION

Distributed transactional storage systems now days require increasing isolation levels, scalable performance, fault-tolerance and a simple programming model for being easily integrated with transactional applications. The recent growth of large scale infrastructures with dozens or hundreds of nodes needs transactional support ready to scale with them. Many of the modern transactional storage systems have abandoned strong consistency (e.g., serializability) in order to achieve good scalability and high performance. Weak consistency models (e.g., eventual consistency) incur the expense of allowing some non-serializable executions, which, if at all tolerated by the application requirements, are more difficult to deal with for the developers.

In fact, it was observed that developers prefer strong consistency when possible.Transactional storage systems that offer serializability without forsaking high speed and high scalability represent a very promising sweet spot in the overall design space. One system that approaches this sweet spot for On-Line Transaction Processing (OLTP) workloads is Granola, as proposed by Cowling and Liskov in. Granola employs a novel transactional model, independent transactions, to keep overheads and synchronization to a minimum while guaranteeing serializable distributed transactions. To help reach its high transaction throughput,Granola relies on storing the data in main memory and operating upon it using transactions expressed in the application's native programming language, as opposed to a query language like SQL. This essentially qualifies Granola as a Distributed Transactional Memory (DTM) system.One key enabler for good performance in the Granola model is having the data organized in fine-grained, high-quality partitions that promote the use of single-partitionand independent distributed transactions. This can be considereda drawback for Granola, as developers need to manuallyorganize the data, choose the transaction primitives, and route transactions appropriately. This work focuses on eliminating this drawback by automating the three tasks. To reach our goal, we adapt and extend an existing graph based data partitioning algorithm, Schism [9], originally proposedfor traditional, SQL-based databases. Our major contributionsare:

1. ***We extend Schism to support independent transactions.The extended algorithm will propose partitionsthat favor fast single-partition and independent transactionsagainst the slower; Two-Phase Commit (2PC)coordinated transactions.***
2. ***We develop a mechanism based on static program analysisfor determining edge weights in the graph thatSchism uses for proposing partitions. This essentiallyenables applying an algorithm like Schism to independenttransactions, or, more generally, to any DTMstyletransactions expressed in a native programminglanguage.***

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

II. BACKGROUND

1. Granola: Independent Transactions

Granola [8] is a transaction coordination infrastructure proposed by Cowling and Liskov. Granola targets On-Line Transaction Processing (OLTP) workloads. Granola is a Transactional Memory (TM) system, as it expresses transactions in a native programming language and operates on data stored in main memory for performance reasons. Synchronization overheads are kept to a minimum by executing all transactions within the context of a single thread. This approach reduces the need for locking, and was shown to significantly improve performance compared to conventional databases in typical OLTP workloads [29, 12, and 15]. Granola employs a novel timestamp-based transaction coordination mechanism that supports three classes of one roundtransactions. Single-Repository Transactions are invoked and execute to completion on one repository (partition) without requiring any network stalls. Coordinated Distributed Transactions are the traditional distributed transactions that use locking and perform a two-phase commitment process. Additionally, Granola proposes Independent Distributed Transactions, which enable atomic commitment across a set of transaction participants, without requiring agreement, locking and with only minimal communication in the coordination protocol. Single-repository and independent transactions execute in timestamp mode. These transactions are assigned an upcoming timestamp, and execute locally in timestamp order. Repositories participating in an independent distributed transaction need to coordinate to select the same timestamp. Each participant proposes a timestamp for executing the transaction, and broadcasts its proposal (vote) to the other participants. Once all votes are received, the final timestamp is selected locally as the maximum among all proposals. This selection is deterministic, and the coordination it requires is very light-weight (needs only one messaginground). At the selected time, the transaction can execute without any stalls or network communication. In order to execute coordinated transactions, the repository needs to switch to locking mode. In locking mode, all transactions must acquire locks (thus incurring overheads), and cannot use the fast timestamp-based execution. Furthermore, coordinated transactions must undergo a slow two-phase commit. The repository can revert to timestamp mode when all coordinated transactions have completed. Granola provides strong consistency (serializability) and fault-tolerance. Data is partitioned between the Granola repositories {with each repository managing one partition {although it is also possible to keep some of the data replicated between repositories to improve performance. Each repository consists of one master and several replicas. The replicas are used for fault-tolerance, not for scalability. Most transactions must be executed by the master node of each repository { the only exception is for read-only, single-repository transactions, which can be serviced by replicas.

In Granola, single-repository and independent distributed transactions never conicts, because they are executed sequentially using a single thread. This means mechanisms employed for rollback and aborts, such as locking and indoor redo-logging, are not needed for these transaction classes, reducing overheads and improving performance. Granola transactions do have restrictions that limit their applicability and place further requirements on the potential partitioning schemes:

1.Independent transactions must reach the same commit decision, independently, on every participating repository. This is possible when the transaction never aborts (e.g., read-only transactions), or the commit decision is based on data replicated at every participating repository.

2. All transactions must be able to complete using only. data available at the current repository. This is arm requirement for single-repository and independent transactions, but could potentially be relaxed for coordinated transactions. Performance in Granola depends on how the workload and partitioning scheme are able to exploit fast single-repository and independent transactions.

3. The user must manually define the partitioning scheme; implement the transactions using the appropriate classes, and route transactions correctly. Furthermore, the partitioning scheme must be compatible with the Granola restrictions outlined above. This paper aims to automate this partitioning process.

2.Schisms: Graph-Based Partitioning

Curino et al. presented Schism [9], the approach for automated data partitioning that we build upon in our present work. Besides lacking support for independent transactions, Schism as it stands cannot be applied to stored-procedure style DTM transactions, which further motivates our work. For the sake of completeness, in this section we overview Schism

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

and describe how it works. Schism takes as input a representative workload in the form of an SQL trace, and the desired number of partitions. It then proposes partitioning and replication schemes that minimize the proportion of distributed transactions, while promoting single-partition transactions. This is done in order to increase performance, as single-partition transactions are fast. The proportion of distributed transaction reprent First, the graph representation phase converts the SQL trace into a graph. Nodes in this graph represent data items (database tuples/transactional objects) that were encountered in the trace. Two nodes are connected by an edge if they were accessed together within the same transaction. Thus, the representation of a transaction takes the form of a clique: the tuples accessed by the transaction are all interconnected. A number of heuristics are applied to promote scalability, such as tuple and transaction sampling, and coalescing tuples accessed by the same set of transactions into a single node. The graph is then modified by replacing each node with a star-shaped con_guration of nodes. This is done in support for data replication. A node A which previously had n neighbors, is replaced by n+1 nodes: one in the center, A0, which is connected to n new nodes (A1...An) by edges representing the cost of replicating the original node A. Each of these new nodes is then connected by a single edge to another node representing the original neighbors. This processing can also be explained as replacing each edge in the original graph by three edges connected in sequence: the two outer edges represent the cost of replicating the data, and the middle edge represents the cost of entering a distributed transaction. An example is illustrated.
In the partitioning phase, the previously constructed graph is partitioned using a standard k-way graph partitioning algorithm. The authors used the program METIS [16] for this purpose. This is an optimization problem, where the primary target is minimizing the cumulative cost of the edges that cut across partitions. This is equivalent to minimizing the number of distributed transactions. A secondary target is balancing the partitions with respect to node weights. Node weights can be assigned based on either data size, or number of transactions, depending on whether the partitions should be balanced in storage size or load. For small workloads, the output of the partitioning phase can be used as-is, by means of a lookup table. Newly created tuples would initially be placed on a random partition, while a separate background task periodically recomputedthe lookup table and migrates data appropriately. This method however cannot be applied to large datasets for two reasons:

(i) Creating and partitioning the graph without sampling is limited by the available memory and processing time.

(ii) The lookup table size is similarly limited bythe available memory.

These reasons motivated Schism's explanation phase. In the explanation phase a more compact model is formulated to capture the tuple partition mappings as they were produced in the partitioning phase. Schism does this by employing machine learning, or more specifically, C4.5 decision trees [25] as implemented in the Weak data mining software [19]. The resulting models are essentially sets of range rules, and are useful if they satisfy several criteria: they are based on attributes present as WHERE clauses in most SQL queries, they do not significantly reduce the quality of the partitions by misclassification, and finally, they work for new, previously unseen queries, as opposed to being over fitted to the training set. To satisfy these criteria, the authors employed strategies such as placing limitations on the input attributes to the classifier, using aggressive pruning and cross-validation, and discarding classifiers that degrade the partitioning quality. Lastly, the final partitioning scheme is chosen in the final validation phase. The candidates considered are (i) the machine-learning based range rules, (ii) the fine-grained lookup table, (iii) a simple hash-based partitioning, and (iv) full-table replication. The scheme chosen is the one with the fewest distributed transactions. In case two schemes lead to similar results, the simpler of the two is chosen.

III. ALGORITHMS USED

ALGORITHM 1 FORWARD DATA-FLOW ANALYSIS PSEUDOCODE.

```
1.  for each unit   allUnits do
2.  allDepsunit.getDirectDeps()
3.  (nodeDeps, valueDeps)   allDeps.partition( isNodeDep )
4.  valueDepsoriginUnitsvalueDeps.getOriginUnits()
5.  valueDeps states   getStateForAll( valueDepsoriginUnits )
```

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

```
6.    merged valueDepStatemergeAll( valueDeps states )
7.    currentState   new DepState( valueDeps, nodeDeps )
8.    newState   merge( currentState, merged valueDepState )
9.    storeStateForUnit( unit, newState )
10.   end for
```

IV. PROPOSED METHODOLOGY AND DISCUSSION

Our partitioning methodology was designed and implemented in the context of a Distributed Transactional Memory (DTM) system. DTM systems store data in main-memory, and access it using transactions expressed in a programming language (usually the same as the rest of the application), as opposed to a separate query language. Declaring and running transactions in DTM should be as simple as possible: ideally the transaction code is simply written inside an atomic block. Our implementation is based around Hyflow2, a JVM-based DTM framework written in Scala. We implemented the Granola protocol in this DTM framework. Unlike Granola, which relies on opaque up calls from the framework to the application and lets the application code to handle locking and rollback mechanisms, we opted to provide a friendlier API and let the framework deal with these mechanisms It Shows an example transaction.

V. ARCHITECTURE

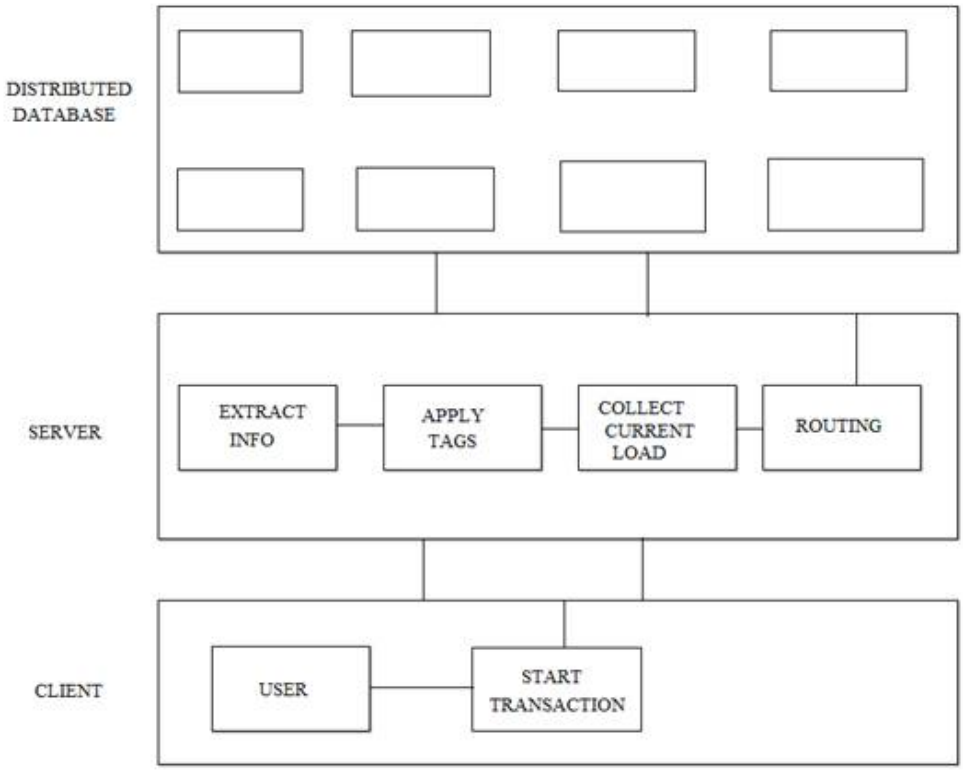

Fig No 01 System Architecture

VI. EXPERIMENTAL SETUP

We evaluate our partitioning process using TPC-C [7], apopular On-line Transaction Processing (OLTP) also used in other significant recent works [30, 15, and 8]. While these works assume optimal manual partitioning, we employ our system in order to automatically derive a partitioning scheme. TPC-C emulates an order processing system for a

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

wholesale supplier with multiple districts and warehouses. The workload was configured with between 3 and 15 warehouses. Throughput measurements were obtained on Future Grid [11] with up to 15 virtual machines. Each node is an 8-core 2.9GHz Intel Xeon with 7GB RAM. We used three classifier types (Naive Bayes [14], Multilayer Perceptron [13] and C4.5 decision trees [25]) for both object placement and transaction routing. Figure 7 shows results for a sample TPC-C workload. In this workload, approximately 10.3% of all issued transactions span more than one warehouse. These transactions would be executed as distributed transactions under the best known manual partitioning for TPC-C, i.e., each warehouse in its own partition,

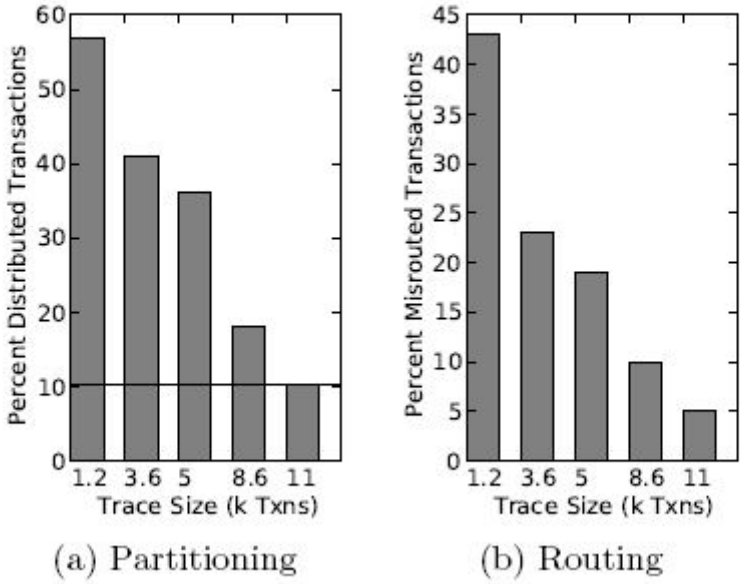

Figure 2: Quality of partitioning and routing with respect to increasing trace size. (1 warehouses). In 9(a), horizontal line represents best known manual partitioning.

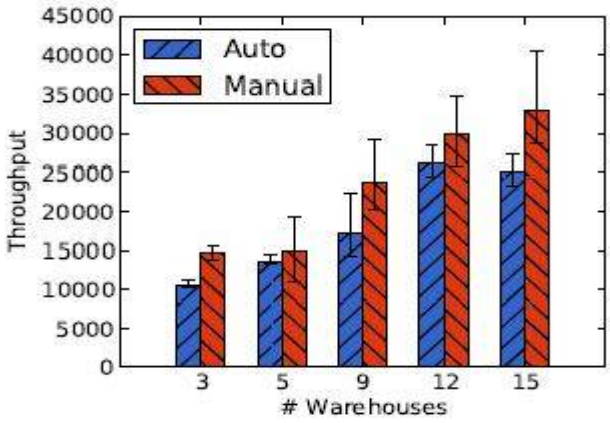

Figure 3: Total throughput with increasing number of nodes/warehouses and 10.3% distributed txns. The bar is the average value. The lower error bar is the standard deviation. The upper error bar is the maximum value.

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

VII. CONCLUSION

We have developed a methodology for using automatic data partitioning in a Granola based Distributed Transactional Memory. We perform static byte-code analysis to determine transaction classes that can be executed using the independent transaction model. We also use the analysis results to propose partitions that promote independent transactions. Due to our DTM focus, we take a machine-learning approach for routing transactions to the appropriate partitions.

REFERENCES

1) G. Karypis and V. Kumar. Metis - serial graph partitioning and fill-reducing matrix ordering, version 5.1, 2013.
2) Pavlo et al. Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems. In SIGMOD, 2012
3) RimmaNehme and Nicolas Bruno. Automated partitioning design in parallel database systems. In SIGMOD ’11, 2011.
4) R. Palmieri et al. Osare: Opportunistic speculation in actively replicated transactional systems. In SRDS ’11
5) Lakshman and P. Malik. Cassandra: A decentralized structured storage system. SIGOPS Oper. Syst. Rev., 2010.
6) Pavlo et al. Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems. In SIGMOD, 2012.
7) S. Peluso, P. Romano, and F. Quaglia. Score: A scalable one copy serializable partial replication protocol. In Middleware’12.
8) [S. Peluso, P. Ruivo, P. Romano, F. Quaglia, and L. Rodrigues. When scalability meets consistency: Genuine multi version update-serializable partial data replication. In ICDCS, 2012.
9) L. Rodriguez and XiaoOu Li. A dynamic vertical partitioning approach for distributed database system. In SMC ’11.
10) Y. Sovran et al. Transactional storage for geo-replicated systems. In SOSP ’11.
11) A Thomson et al. Calvin: fast distributed transactions for partitioned database systems. In SIGMOD ’12.
12) [12] S. Harizopoulos, D. J. Abadi, S. Madden, andM. Stonebraker. Oltp through the looking glass, andwhat we found there. SIGMOD '08.
13) Simon Haykin. Neural Networks: A Comprehensive
14) Foundation. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 199, George John and Pat Langley. Estimating continuousdistributions in bayesianclassi_ers. In In Proceedingsof the Eleventh Conference on Uncertainty in Arti_cialIntelligence, pages 338{345. Morgan Kaufmann, 1995.