An autonomous hybrid data partition for NewSQL DBs

Geomar André Schreiner[1]
Coorientador: Denio Duarte[2]
Orientador: Ronaldo dos Santos Mello[1]

[1]Programa de Pós-Graduação em Ciência da Computação
Departamento de Informática e Estatística– Universidade Federal de Santa Catarina
Florianópolis – SC – Brasil

[2]Universidade Federal da Fronteira Sul
Chapecó – SC – Brasil

schreiner.geomar@posgrad.ufsc.br, duarte@uffs.edu.br, r.mello@ufsc.br

Nível : Doutorado
Admissão : Março de 2016
Exame de Qualificação : Junho de 2018
Conclusão : Março de 2020
Etapas Concluídas : Revisão bibliográfica e projeto
Etapas Futuras : Implementação e avaliação

Abstract. *Several applications like online games and financial market seek support for features such huge data volumes management, data streaming and handle thousands of OLTP transactions per second. Traditional Relational Databases (RDBs) in general are not suitable for these requirements. NewSQL is a new generation of DBs that combines the high scalability and availability with the ACID support. NewSQL is a promising solution to handle these application requirements. Although data partition is an important feature for tunning relational DBs, it stills an open problem field for NewSQL systems. This Thesis proposes an automated approach for hybrid data partitioning that automatically reorganizes data based on the current workload of the NewSQL DBs.*

Resumo. *Várias aplicações, como jogos on-line e mercado financeiro, buscam suporte para recursos como gerenciamento de grandes volumes de dados, data streaming e suporte a milhares de transações OLTP por segundo. Bancos de dados relacionais tradicionais (RDBs), em geral, não são adequados para esses requisitos. BDs NewSQL são uma nova geração de bancos de dados que combina alta escalabilidade e disponibilidade com o suporte as propriedades ACID. Os NewSQL mostram-se uma solução promissora para lidar com esses requisitos de aplicações. Embora a partição de dados seja um recurso importante para melhoria no dempenho de sistemas NewSQL é ainda um problema aberto na literatura. Esta Tese propõe um controle da evolução do particionamento dos dados de um BD NewSQL com suporte a streaming de dados, de maneira autônoma, a fim de melhorar o desempenho de transações OLTP e de consultas de streaming.*

Palavras Chave: data partitioning, NewSQL, OLTP Systems, Automate partitioning, Big Data

1. Introduction

Historically, application systems rely on OLTP transactions to perform small operations over the network, such as buying an item in an online store [Stonebraker 2012]. The number of users that uses online stores has increasingly grown, and so the number of OLTP transactions performed. With the emergence of *Web*, OLTP requests has changed as well to deal with transactions on on-line games, social networks, or even large financial companies. These applications are characterized by having a large number of user interactions with the system, generating a huge amount of data and multiple OLTP transactions per second.

For decades, traditional Relational Databases (RDBs) have been used as an efficient way to store AND handle applications data, but they are not suitable to handle these massive data volume associated with high availability and ACID support guarantees [Stonebraker 2012]. Based on problems faced when Big Data needs to be managed new architectures are emerging like NoSQL DBs. These new architectures were born in cloud environments, generally, capable of storing and managing huge data volumes, keeping high availability and scalability. NoSQL solves part of the problems related to Big Data management, they maximize availability rather than ACID support. Companies keep using RDBs for most of their data-applications because it is simpler to deal with the overhead of traditional ACID assurance than with the lack of these properties. With the goal of offering availability, scalability and ACID support, the *NewSQL* movement has arisen.

NewSQL DBs main advantage is try to combine the best of both worlds: scalability and availability of NoSQL DBs with ACID properties of traditional RDBs [Stonebraker 2012]. Usually, NewSQL DBs are distributed in-Memory DBs, and each node of the DB has a partition of the stored data [Pavlo and Aslett 2016, Taft et al. 2014, Elmore et al. 2015, Kallman et al. 2008]. Data partitioning in distributed systems affects directly the system performance, since data need to be reconstructed or merged to attend DBs operations. Thus, the proposition of efficient approaches to control data partitioning, in order to maximize access performance and OLTP processing, becomes essential in this type of data management system.

Generally, relational data can be partitioned in two ways: horizontally (by rows) and vertically (by columns). Both types of partition vertical and horizontal can improve query performance in different ways [Al-Kateb et al. 2016]. For example, when a selection is performed, if the tuples are in the same partition,the network traffic is optimized. The same reasoning may be used to the projection operation, vertical partition optimizes data access as well. We found some initiatives in literature that explore the use of vertical partition in OLTP transactions [Amossen 2010] and hybrid partitioning in systems that consider OLAP transactions [Arulraj et al. 2016], but data partitioning stills an open field for NewSQL systems.

H-Store [Kallman et al. 2008] and *S-Store* [Cetintemel et al. 2014] are NewSQL data management systems that consider a pre-workload plan to define an optimized partitioning. However, the data volume and workload of a DB are not static (*i.e.*, constantly change) and these proposals do not consider the evaluation and modification of these partitioning strategies. Other proposals, such as *Clay* [Serafini et al. 2016], *Accordion* [Serafini et al. 2014] and *E-Store* [Taft et al. 2014] have repartitioning tools. All

three have an autonomous system that just considers OLTP loads. However, none of them has considered an hybrid data partition or streaming application requirements.

In this Thesis, we propose a new automated partitioning system that evolves the partition scheme of a NewSQL DB. We consider hybrid data partition for minimizing distributed transactions on OLTP systems and considering data streaming requirements. Our proposal is a reactive system capable of generating new data partition schemes based on the workload of the system and reorganize data with no down time seeking for a best performance of the NewSQL system. Expected contributions of our work are: (*i*) a novel approach for data partitioning that considers hybrid data partition for OLTP and streaming operations minimizing the distributed transactions; (*ii*) partitioning based in a fine-grained level, storing tuples in an hybrid scheme (vertical and horizontal) based workload access; (*iii*) an adaptive approach that allows the system to reorganize data based on the current workload; and (iv) an approach that also considers replicas to increase availability and therefore decreases the number of distributed transactions.

The rest of this paper is organized as follows. First we present our propose in a high level structure showing a generic NewSQL architecture and our propose partition algorithm. Section 3 discuss related works. Section 4 presents the current status of our work and some future activities.

2. Proposal

Figure 1 presents the general architecture of the NewSQL system, and it is basically a distributed architecture with two components: *(i)* a *master* node (dotted border), and *(ii) worker* nodes named *sites* where the data partitions are stored.

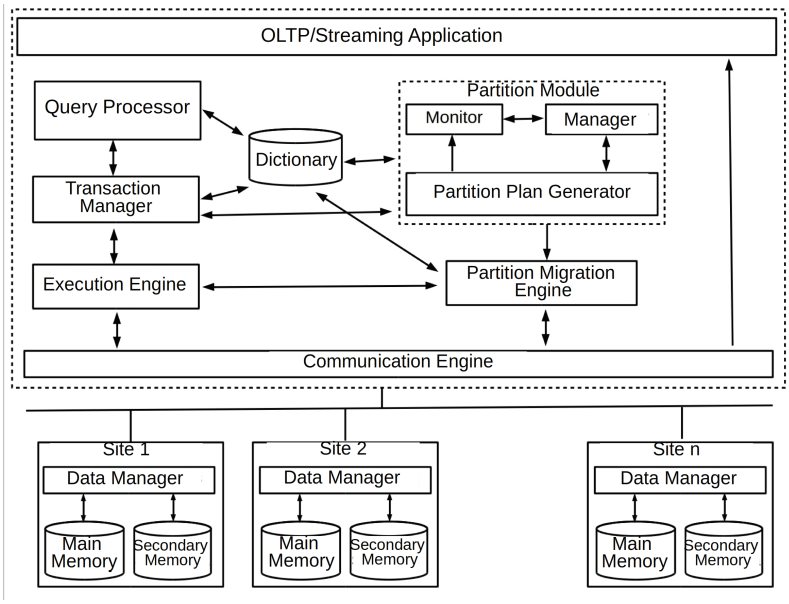

Figure 1. Proposed architecture

The *master* node receives all requests sent by applications through a specific *interface* (OLTP / Streaming Application module - Figure 1). The application connects to this interface and sends its requests and receive the responses. The master node is composed of: (*i*) Data dictionary that stores meta-data information such as data schema,

which partitions the data is located, how often it is used, *stored procedures*, and updated information about the *workload*; (*ii*) Query Processor that receives a given SQL command and defines a query plan using meta-data information stored in the Dictionary; (*iii*) Transaction Manager which ensures the efficient execution of the query plan, or a transaction in general, in the correct order; (*iv*) Partitioning Module, part of our contribution, for generate new optimized partitioning scheme; (*v*) Execution Engine that is responsible for executing each of the operations requested by the Transaction Manager and to control the *streaming*-based queries; (*vi*) Partition Migration Mechanism which receives a new partition plan to migrate the data accordingly; and (*vii*) Communication Engine module which performs the communication between the master node and worker nodes in order to execute the operations that the Execution Engine requests.

In general, NewSQL systems aim to handle single node transactions by avoiding distributed transactions, so the *Transaction Manager* tries to assure the execution order of transactions in a single *site* otherwise, it coordinates the distribution of the operations among the *sites* and assures the ACID properties.

The Sites (Site 1 to n in Figure 1) have a simple structure and are used to store the data partitions, one partition per site. Sites are composed of only one module *Data Manager* that uses *Main Memory* (volatile and fast access) and *Secondary Memory* with slower access) to manage the data. *Data Manager* is responsible for receiving demands coming from the *master* node and managing which data is in main memory and which data should remain on the disk.

2.1. Partition Module

The Partitioning Module is responsible for monitoring the performance of the approach and creating optimized partitioning plans for the application demands. The module is organized into three components: *Monitor*, *Manager* and *Partition Plan Generator* (see Figure 1).

The *Monitor* component collects information about the overall state of the system in real time (is always active by collecting information about the overall state of the system). It collects access statistics information about the data schema present in the *Dictionary* as well as other information about the types of transactions executed directly in the *Transaction Manager* module. The information collected serves as the basis for *Manager* component decision and is used by the *Partition Plan Generator* component to optimize data partitioning.

Based on the information collected by *Monitor*, the *Manager* component decides whether or not a partition migration is required. The *Manager* analyses workload of the system using some heuristics to identify (possible) unbalancing points or critical points (recurrent distributed transactions) and calls the *Partition Plan Generator* for a new partition plan. With a new partition plan, the *Manager* checks, through a cost function, if the data migration does not present a very high cost for the performance gain. If the migration cost is too high and the performance gain is too small, the system does not migrate the partition.

We use a heat graph structure to help the *Partition Plan Generator* to create new partition plans. The heat graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a graph partitioned in n partitions composed of a set of vertices $\mathcal{V}$ and a set o edges $\mathcal{E}$. Each vertice $\nu \in \mathcal{V}$ represents an accessed

tuple. ν is composed of a tuple $< n_\nu, n_{table}, Attrs, k >$ where n_ν is the identifier of tuple (*rowid*), n_{table} name of the table that the tuple belongs, $Attrs$ is the set of attributes accessed by the tuple (a projection of SQL statement), and k is the number of times that the tuple was accessed, the value of k indicates the tuple temperature (bigger the value hottest is the tuple). Each edge $\epsilon \in \mathcal{E}$ represents tuples that were accessed together in a transaction, and ϵ is composed of a tuple $< \nu, w, \nu' >$ where ν and ν' are two vertices that represents two tuples accessed together, and w number of times that the tuples were accessed (weight). The same tuple can be accessed several times with different projections, for each different projection, a new vertice is created with the tuple identifier and the access projection. Vertices that represent the same tuple (have the same *identifier*) are connected by edges with no weight (w=0). The graph $\mathcal{G}$ is partitioned in n partition, each partition maps one *site* of the NewSQL systems.

To generate a new partition plan the *Partition Plan Generator* first analyzes $\mathcal{G}$ to search for unbalanced partitions. As described previously, each partition of $\mathcal{G}$ maps the *sites* and respectively tuples (*vertices*). To detect unbalancing partitions we use a function that sums all vertices weights for each partition, if a partition have a total weight very different from the others, then it is unbalanced. When an unbalancing partition is detected, the approach selects the overloaded partition and analyzes the hottest tuples, searching for one tuple t that can migrate to an underloaded partition. Based on $\mathcal{G}$ a new graph $\mathcal{G}'$ is created migrating t to the new partition. Each edge that connects t to other tuples t' in a different partition is considered as a critical edge. We analyze all critical edges (edges that connect vertices from different partitions) and reallocate them to the same partition. That is, tuples that are accessed together but are in different partitions should be in the same partition. We try to keep as few as possible vertices belonging to different partitions to be connected by heavy edges, *i.e.*, giving $<\nu, w, \nu'>$ and $\nu \in p_1$ and $\nu' \in p_2$ ($p_1 \neq p_2$), the closer w is to 0, the better. Finished this phase, $\mathcal{G}'$ is translated by a decision three in a partition plan and sent to *Manager* component. The *Manager* component, using a cost function, analyzes if $\mathcal{G}'$ partition plan is better than $\mathcal{G}$. If $\mathcal{G}'$ is better than $\mathcal{G}$, then data migrating tool is called to reorganize the partitions.

If no unbalancing point is detected the approach analyses the critical edges of $\mathcal{G}$. If a critical edge e is found with a high weight, the approach evaluates the migration options for the vertices connected by e. A new graph $\mathcal{G}'$ is created based on $\mathcal{G}$, and $\mathcal{G}'$ is used to migrate tuples among partitions.

Our migration approach will be based on the Squall [Elmore et al. 2015]. Squall makes a fine-grained migration, at tuple level with no downtime. Squall synchronizes all sites and each *site* is responsible for evaluating the partition plan and the routing table (that stores where each tuple is) to migrate their data The migration transactions are routed to the target *site* and added with no priority on its transactions queue. Eventually, if a transaction T is routed to a *site*, and the data is waiting for migration, then T is put in a lock state, the site execute the migration immediately , and then T is unlocked and execute their operations.

3. Related work

Our approach fits into related work that offers a way of data partition for systems with the main focus in OLTP transactions (NewSQL DBs). All ap-

proaches found in the literature that propose partition approaches for OLTP-based systems use horizontal partitioning to organize the data, that is, they all split the tables into sets of tuples, and each partition maintains a subset of the table. *H-Store* [Kallman et al. 2008], *Horticulture* [Pavlo et al. 2012], *Accordeon* [Serafini et al. 2014], and *S-Store* [Meehan et al. 2015] partition their tuples horizontally using the identifier of each tuple. Each partition stores a set of tuples with sequential identifiers. On the other hand, *Schism* [Curino et al. 2010], *E-Store* [Taft et al. 2014], and *Clay* [Serafini et al. 2016] group tuples by affinity. In this way, tuples usually accessed together will be stored in the same partition.

Most of the approaches are based on autonomous partitioning. *H-Store* and *S-Store* leave it to the DBA to create or use an external tool that will generate suitable partitioning for the desired demands. However, other approaches have support to the autonomous partitioning aiming at decreasing the distributed transactions. *Horticulture* only takes into account the operations provided by the DBA and the data schema to feed its LNS (Large-Neighborhood Search) algorithm, which generates the partitioning. *E-Store* partitions its data using some statistics to identifying the most accessed tuples. These tuples are allocated in different nodes, balancing the loads. *Accordion*, in addition to statistics generated from the workload, takes into account the maximum capacity of each server and groups the partitions accessed together on the same server or nearby servers. Unlike the others, *Clay* and *Schism* use a heat graph to accomplish this task. This graph is created based on the tuple access. Each tuple is represented by a vertice with a temperature (number of times the tuple was accessed). Edges connecting vertices represent tuples accessed together in the same transaction.

Only three of the approaches use replicas of data to improve partitioning performance. *Horticulture* and *Schism* create a partitioning plan that takes into account block replication to facilitate transaction execution and increase system availability. *S-Store*, although it does not have a repartitioning model, performs data replication for *streaming* support. Data accessed by a *streaming* operation are replicated into tables created exclusively for this purpose.

Based on the literature review, we can see the lack of an approach that explores some specific aspects: *(i)* a hybrid tuples partitioning that decides when it is advantageous to use vertical or horizontal partitioning; *(ii)* a solution that encompasses data streaming and OLTP load issues, enabling the system to meet the requirements of a wide range of applications; *(iii)* a solution that considers replicas to increase availability and therefore decreases the number of distributed transactions; and *(iv)* a solution that performs a gradual and periodic evolution of the existing partitions, avoiding periods of high latency until its partitioning is optimized.

4. Final Considerations

This thesis proposes an autonomous approach for data partitioning for NewSQL BDs, taking into consideration the *workload*, OLTP loads and data streaming support. The proposed approach is unprecedented in the literature for generating a hybrid data partition scheme (vertical and horizontal), offering data optimization and data storing based on access workload, furthermore we propose a partitioning algorithm that considers data replication to reduce overhead of distributed transactions.

This Thesis is in half way of its development, so we have some future works in mind. In current status, we are developing support for vertical/hybrid partitioning in *VoltDB*. The next steps involve the partition module development for VoltDB to validate the ideas of the approach. To validate our approach, we will use consolidated benchmarks (TPC-W, YCSB) comparing with state-of-the-art approaches. We also plan to evaluate the use of machine learning techniques to predict when a partition reorganization should be triggered.

References

Al-Kateb, M., Sinclair, P., Au, G., and Ballinger, C. (2016). Hybrid row-column partitioning in teradata®. *Proc. VLDB Endow.*, 9(13):1353–1364.

Amossen, R. R. (2010). Vertical partitioning of relational oltp databases using integer programming. In *ICDEW*, pages 93–98. IEEE.

Arulraj, J., Pavlo, A., and Menon, P. (2016). Bridging the archipelago between row-stores and column-stores for hybrid workloads. In *SIGMOD 2016*, New York, NY, USA. ACM.

Cetintemel, U., Du, J., Kraska, T., and Madden, e. a. (2014). S-store: A streaming newsql system for big velocity applications. *Proc. VLDB Endow.*, 7(13).

Curino, C., Jones, E., Zhang, Y., and Madden, S. (2010). Schism: A workload-driven approach to database replication and partitioning. *Proc. VLDB Endow.*, 3(1-2).

Elmore, A. J., Arora, V., Taft, R., Pavlo, A., Agrawal, D., and El Abbadi, A. (2015). Squall: Fine-grained live reconfiguration for partitioned main memory databases. In *2015 ACM SIGMOD*, pages 299–313, New York, NY, USA. ACM.

Kallman, R., Kimura, H., Natkins, Stonebraker, M., et al. (2008). H-store: a high-performance, distributed main memory transaction processing system. *VLDB*, 1(2).

Meehan, J., Tatbul, N., Zdonik, S., Aslantas, C., et al. (2015). S-store: Streaming meets transaction processing. *Proc. VLDB Endow.*, 8(13).

Pavlo, A. and Aslett, M. (2016). What's really new with newsql? *SIGMOD Rec.*, 45(2):45–55.

Pavlo, A., Curino, C., and Zdonik, S. (2012). Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems. In *2012 ACM SIGMOD*, pages 61–72, New York, NY, USA. ACM.

Serafini, M., Mansour, E., Aboulnaga, A., Salem, K., Rafiq, T., and Minhas, U. F. (2014). Accordion: Elastic scalability for database systems supporting distributed transactions. *Proc. VLDB Endow.*, 7(12):1035–1046.

Serafini, M., Taft, R., Elmore, A. J., Pavlo, A., Aboulnaga, A., and Stonebraker, M. (2016). Clay: Fine-grained adaptive partitioning for general database schemas. *Proc. VLDB Endow.*, 10(4):445–456.

Stonebraker, M. (2012). New opportunities for new sql. *Commun. ACM*, 55(11).

Taft, R., Mansour, E., Serafini, M., Duggan, J., Elmore, A. J., Aboulnaga, A., Pavlo, A., and Stonebraker, M. (2014). E-store: Fine-grained elastic partitioning for distributed transaction processing systems. *Proc. VLDB Endow.*, 8(3).