MANTECH Publications

Recent Trends in Computer Science and Software Technology
Volume 6, Issue 2, May-August 2021

Efficient Distribution for Distributed OLTP DBMS

P.R.K. Ayyappa[1], P. Veena[2]

Assistant Professor[1, 2] Department of Computer Science

Engineering St. Martins Engineering College, Hyderabad[1], Khammam

Institute of Technology and Science, Khammam[2]

Corresponding Authors' E-mail: *ayyappar4@gmail.com[1], veena.pentyala@gmail.com[2]*

Abstract

Distributed online transaction processing (OLTP) database management systems (DBMS) are characterized with ACID properties, that is atomicity, consistency, isolation, durability. However, these OLTP DBMSs that handle massive data schema, need to be scalable that cannot give up strong transactional and consistency requirements. The large-scale schemas are deployed on distributed server and each server's memory is limited, in that case, how this system partition is the key factor to affect its performance. In this paper, we present a partition approach' that applies to general schema and taking care of load balancing and distributed transaction rates. Our method achieve this by periodically update the partition plan and migration hot data among the servers. OLTP workload are modelled as a graph and using a graph partition algorithm to partition set of data into same server that are often co-accessed by one transaction. To evaluate our method, our framework Assort is integrated into a distributed, main-memory DBMS and shows that it can partition schema in OLTP DBMS and enable the system to outperformance traditional method 3-4 h throughput.

Keywords*: - OLTP, DBMS, partition, datamigration*

Page 42-49 © MANTECH PUBLICATIONS 2021. All Rights Reserved

INTRODUCTION

Nowadays distributed, modern OLTP databases have found an increasingly wide utilization in diverse enterprises. These systems run on multiple servers and the data is partitioned into each server. The routing table can route transaction queries to the corresponding server based on the data these transaction touch.

However, these system's performance, are very sensitive to how data partitioned. Traditional hash/range partition strategy will cause great reduction of performance because of distributed transactions, which mean transactions access multiple keys.

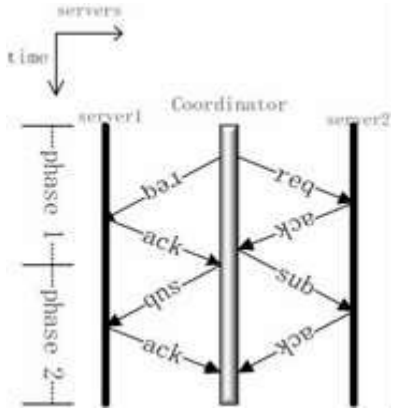


Figure 1 A Typical Two Phase Commit Example in Modern OLTP Database

In order to meet the NoSQL's scalability, but also to meet the relational database's transactional attributes, there is a new database was raised, called NewSQL[1]. Current distributed OLTP DBMSs are characterized with massive distributed transactions, which means accessing mutiple key in one transaction. In most cases, the consistency of transactions for online transactions more focuses on atomicity semantics of conceptual ACID database transactions. Traditional database distributed transactions are implemented using protocols such as two-phase commit (2PC) in Figure1. A similar protocol can result in huge performance overhead because of distributed transactions.

To make distributed OLTP scalable. The current OLTP system requires a efficient database partition strategy. For this database,there are two factors that affect its performance: distributed transaction and workload skew. It is important to repartition without interrupting the database because OLTP systems have periodic load fluctuations.

RELATED WORK

There is a significant amount of research on how OLTP databases are partitioned to balance data load and maximize parallelism.

Although a number of automatic partitioning schemes have been investigated, the mostly usually used methods are round- robin (send each

Page 42-49 © MANTECH PUBLICATIONS 2021. All Rights Reserved

successive tuple to a different partition), range (divide up tuples according to a set of predicates), or hash partitioning (assign tuples to partitions by hashing them)[2]. All of these are useful for large relational database queries and analysts. However, These methods ignore the inherent relationship between data and always cause great amount of distributed transactions.

Tsangaris and Naughton's ramdon approach using a graph partitioning heuristics for clustering in OODB (object-oriented database)[3]. However, our system also integrates replication and is designed for different requirements of distributed database. There are also "simpler" distributed storage systems, such as BigTable [4] or PNUTS [5]. These systems continually rearrange the data locality and balance the workload across multiple servers. Although this method is more dynamic, it does not apply to OLTP transactions for multiple tables, general schema.

The hybrid-range partitioning strategy (HRPS) [6] is a hybrid method to hash and range partitioning based on query analysis. The scheme divides the permitted queries for a period of time and localizes the minimum query scope, which is accomplished by running in parallel on multiple nodes and localize small range queries. However, HRPS does not consider replication, and applies to range queries of single- table over one dimension.

SYSTEM DESGIN

The modern in-memory OLTP system is highly distributed, row-store based relational database. It runs on a data center on shared-nothing executor nodes. A single instance is defined as a cluster of two or more servers deployed within the same administrative sites. A site is the basic operational entity in the OLTP system. Each site is a daemon that can communicate with the client in order to execute a transaction. We assume that each node in the database has more than one core, and each site is assigned to execute on exactly one core. Each site is running independently and does not share any data or memory with sites located on the same server. Every table schema in the database is cut up into one or more partitions. A partition is replicated on multiple sites, forming a k-safety replica. OLTP applications send request to the system to repeatedly execute predefine SQL statements. A system of a store procedure initiated by OLTP application is called a transaction.

Page 42-49 © MANTECH PUBLICATIONS 2021. All Rights Reserved

Transaction

We now provide description of two classes of transactions and discuss how OLTP system generates an optimal execution strategy. To figure out how concurrent control affects the performance of current OLTP cluster, we illustrate how single-sited transaction works and how multiple-sited transaction makes OLTP system slow.

Single-sited transaction: a transaction is single-sited if all the queries accessing data on just one computational site in the OLTP data center. Although due to replicated data, the transaction should be sent to two or more sites for execution, it does not require consistency and real-time processing. This class of transaction is the most desirable because it does not demand to use concurrency control mechanism, or communication with other sites, except for communication with replica data. When the system receives a single-partition transaction request, the site receiving the request executes the concurrency control and any compiled query plans and transfers it to the target site. The new site instead of the originally requested site begins to process the transaction, and performs the query at the new site. The cost of transferring requests between two sites is offset by the performance gain of avoiding cross-site network traffic.

Multi-sited transaction: A transaction can be multi- sited if it cannot execute on a single site, but each separate query fragment will execute on only one site instead of all queries on the same site. Unlike single- partitioned transactions, we assume that mulple- site transaction does not execute on the same site for the same transaction. If a transaction is declared as a multi- site, the OLTP system performs a simple optimization technique: the request is forwarded to the most visited site for that instance of the transaction in the same way as a single-site action.

Partition

Each table in an OLTP system is horizontally split into multiple sub-tables, each consisting of disjoint subsections of the entire table. The boundaries of the partition of the table are based on the selection of the partitioned attributes (i.e., columns) of the table; each tuple is assigned to a partition based on the hash value of its attributes. In order to support multiple columns of partitioned tables, the associated partitions in multiple tables are merged into a different partition than all the other partitions. Each partition can only be assigned to one site.

Page 42-49 © MANTECH PUBLICATIONS 2021. All Rights Reserved

All tuples in our system are stored in memory on each server; The system never needs to access the disk while accessing the data. But it must copy partitions to ensure data durability and availability in case of node failure. Data replication is done in two ways: (1) Partitioning on multiple nodes; (2) Copying of the entire table in all partitions. To design efficient partitioner, a database generate a representative workload (e.g., a transaction trace), and input the number of partitions desired. Partitioner provides a partitioning and replication strategy that balances the size of partitions while minimizing the overhead of running workloads and cross-site communications. The approach consists of five steps:

Monitoring and Logging: The system have a logging system that can store the transaction trace in a period of time. The system calculates and writes each transaction in the trace of the workload into the system collections. The collecting data includes the each transaction and the tuples number one transaction accessed.

Creating the graph: Create databases and workloads and model them into graph models. A node is for each tuple. An edge represents an access to a tuple within a transaction. If one side is accessed by the same transaction, connect two tuples. The weight of the side of the transaction access frequency. The graph the is algorithm model used for efficient partition and create the partition plan.

Partitioning the graph: Graph partitioning algorithm is used to generate the balanced minimum cut of the graph divided into k partitions. Each tuple is assigned to a partition, and each partition is assigned to a physical node. We use a open-source partition software, called METIS [8]. The algorithms implemented in METIS are based on multi-level recursive dichotomy, multi-level k- way and multi-constraint allocation schemes. Because most OLTP DBMS partition the database using horizontal partition. We only consider horizontal partition and each tuple will be one row in the table.

Data Migration: When the database through the period of trace calculated partition plan, the data offline through the generation of migration plan. The database migrates the data to the corresponding partition through the migration plan. During the migration, update the database routing table.

Page 42-49 © MANTECH PUBLICATIONS 2021. All Rights Reserved

EVALUATION

Experimental Environment

We use run our experiment on platform Alibaba Cloud. The cluster have 6 nodes, and each nodes with 4 cores(2.3GHz), each core executes a partition. There are total 24 partitions for OLTP processing. Each node is with memory 8GB. The network bandwidth limit is 100Mbps.

Table 1 Experiment configuration details

Item	Configuration
CPU	Each with 4 cores(2.3GHz)
Memory	Each with 8GB
Network bandwidth limit	100Mbps
Number of nodes	6 nodes
Partition	24 partitions

Performance Evaluation

To test the performance on real-world schema, We integrated out framework into H-Store [9], a distributed main memory OLTP DBMS. We use twitter benchmark from OLTP-bench testbed[7]. This is the inspiration for the popular Weibo site. To make the baseline access model more realistic, we defined a "follow" relationship between users based on a subset of the true Twitter social graph of

51 million users and 2 billion "follow" relationships in August 2009. We use 6 nodes and 2 client to send two kind of transaction, "get tweets from followers" and "get user from user id",and mix the transaction rate.

For partition the graph, we use METIS[8] that partition a graph into k subgraph and minimizes the edge cuts.

Figure 2 shows improvement of throughput after using metis on efficient partition. The mutiple tuple transaction rate range from 80% to 10%. Figure 3 shows the reduction of distributed transaction(multi-sited transaction) rate when mutiple tuple transaction vary form 80% to 10%. We can see a significant increase on throughput, the larger multiple tuple transaction rate, the more throughput improvement. Also, the distributed transaction shows a small reduction. The experiment well reached our goal, validated our ideas.

Page 42-49 © MANTECH PUBLICATIONS 2021. All Rights Reserved



Figure 2 Throughput when mutiple-tuple transaction rate varies



Figure 3 Distributed transaction when mutiple-tuple transaction rate varies

CONCLUSIONS

This paper present an efficient partitioner for modern OLTP DBMS. Found the problem of slow access when there is distributed transactions. To decrease distributed transaction and balances load. Our algorithm metis can partition data that co-accessed frequently on the same partition thus improving the performance. Our experiments shows that our partitioner outperforms the state-of-art database.

ACKNOWLEDGMENT

Thanks to Xiaoming Huang at tcloud lab in Shanghai Jiaotong University for technical assistance which includes providing instruments, equipment or related experimental materials, collaborating on experimental work, providing useful inspiration, advice, guidance, reviewing, taking on certainty assistive work, and more.

Page 42-49 © MANTECH PUBLICATIONS 2021. All Rights Reserved

REFERENCES

1. A.Moniruzzaman, "NewSQL: towards next-generation scalable rdbms for online transaction processing (oltp) for big data management," arXiv preprint arXiv:1411,7374, 2014.
2. DeWitt D, Gray J. Parallel database systems: the future of high performance database systems[J]. Communications of the ACM, 1992, 35(6): 85-98.
3. Tsangaris M M, Naughton J F. A stochastic approach for clustering in object bases[M]. ACM, 1991.
4. Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data[J]. ACM Transactions on Computer Systems (TOCS), 2008, 26(2): 4.
5. Cooper B F, Ramakrishnan R, Srivastava U, et al. PNUTS: Yahoo!'s hosted data serving platform[J]. Proceedings of the VLDB Endowment, 2008, 1(2): 1277-1288.
6. Ghandeharizadeh S, DeWitt D J. Hybrid-Range Partitioning Skate y: A New Deelustering Strategy for Multiprocessor 8 atabase Machines[J]. 1990.
7. Difallah D E, Pavlo A, Curino C, et al. Oltp-bench: An extensible testbed for benchmarking relational databases[J]. Proceedings of the VLDB Endowment, 2013, 7(4): 277-288.
8. Karypis G, Kumar V. METIS--unstructured graph partitioning and sparse matrix ordering system, version 2.0[J]. 1995.
9. Kallman R, Kimura H, Natkins J, et al. H-store: a high-performance, distributed main memory transaction processing system[J]. Proceedings of the VLDB Endowment, 2008, 1(2): 1496-1499

Page 42-49 © MANTECH PUBLICATIONS 2021. All Rights Reserved