Automated Data Partitioning Scheme For High Performance Transactional Systems

Ashwed Patil

Abstract— Data Partitioning Strategies in modern transactional processing systems have a significant impact on the overall performance, throughput and scalability. In order to maximize the benefits, many such systems settle for weak consistency models. Modern Transactional Systems have a hybrid architecture that is a combination of client server machines and distributed environment. This paper proposes a system which uses an optimal automated data partitioning technique based on machine learning which can considerably reduce workload on server preventing its failure. A retail website framework and focus on local optimization (client server level) to achieve high performance is considered..

Index Terms— Data Partitioning, Machine Learning, Client Server Architecture, Transaction, Consistency, Scalability, Classification

1 INTRODUCTION

Data Partitioning strategies play a central role in the performance of transactional systems. Modern transactional systems have a hybrid architecture which is usually a combination of client server machines and distributed environment. Techniques such as replication or fragmentation provide efficient solutions at distributed level. However, partitioning strategies at client server level haven't improved much as compared to distributed level. Executing transactions usually involves JOIN operation on multiple tables which is time consuming and complex. This could result in server failure due to excessive load of recurring JOIN operations for every query. Data objects which are frequently accessed by multiple transactions, if placed on a separate partition can significantly reduce execution time and increase the performance. Use of machine learning in partitioning can automate the process while still maintaining the ACID properties.

Maintaining the reliability at incremental scaling is one of the biggest challenges for transactional systems based on hybrid architectures. For this purpose, many modern transactional systems use a decentralized, highly coupled service architecture. Dynamo, Amazon's key-value based distributed database[1] uses consistent hashing for partitioning[4] which offers incremental scaling but at the cost of increased complexity. The system also uses Merkle trees trees[5]to recover from server failures by using synchronization of replicas in the background which however, can increase performance overheads

Other systems such as BigTable[2] or PNUTS[3] use dedicated directory services that offer flexibility but their performance suffers significantly on client server level, especially when relying on remote directory services..Table 1 summarizes the data partitioning and storage strategies of some modern transactional systems based on hybrid architecture.

Table 1: Summary of techniques used in modern transactional systems.

System	Technique(s)	Advantage	Disadvantage
Dynamo: Amazon's Key-Value Store	Consistent Hashing, Merkle Trees	Incremental Scalability	Increased complexity, Synchronization Overheads
PNUTS: Yahoo's Data Serving Platform	Hash table mapping, message broker	Low latency, consistency guarantees	Centrally managed, high risk of server failure
BigTable: Google's Distributed Database	Google SSTable	High flexibility	Relaxed consistency

Thus, most of the modern transactional systems sacrifice one or the other performance metrics to achieve high efficiency. Also, there is a significant lack of optimized partitioning techniques at the client server level which increases the risk of server failure to a great extent. Our proposed system introduces a key innovative solution that uses machine learning to automated the data partitioning process without sacrificing any of the performance parameters.

- *Ashwed Patil is currently pursuing bachelors degree program in computer engineering in K.K. Wagh Institute of Engineering Education and Research, Savitribai Phule Pune University, India, PH-919673488925. E-mail: ashwed3194@gmail.com*

The paper is structured as follows. Section 2 presents the system architecture and Section 3 describes the implementation. Section 4 presents the result and Section 5 concludes the paper.

2. SYSTEM DESIGN AND ARCHITECTURE

2.1 DESIGN ASSUMPTIONS AND CONSIDERATIONS

In context of databases, a *transaction* is a single logical unit of work on data. ACID (*Atomicity, Consistency, Isolation, Durability)* properties guarantee that the transactions are processed reliably. A *partition* is a division of logical database and is usually done for higher performance or availability. Many systems tend to sacrifice ACID properties during data partitioning usually for higher availability and then operate on weak consistency.

The scope of the paper is limited to local optimization. We consider application of the proposed technique at client server level and it may not be efficient at distributed level. The project will partition the transactions to correct data partitions and does not extend to memory partitioning. The performance metrics are measured only with respect to consistency, scalability and throughput. The dataset used in the system described by the paper is relatively small as compared to modern real time applications.

Finally, we do not consider certain constraints such as network latency in client server environment and server storage capacity.

2.2 SYSTEM ARCHITECTURE

We consider a retail based framework based on a service oriented architecture. Traditionally, transactional processing systems have been storing their state in relational databases. The complex querying and management functionality of RDMS is largely kept hidden through abstraction with multiple logical views. This makes it suitable for service oriented architectures.

The data objects in the proposed system have a special attribute called *counter* which keeps a track of search record for that data object. A particular threshold value is set manually in the server settings during the implementation. If a product is searched more than its specified threshold value, then it is placed in the cache table. During the next search operation, the product will be searched from the cache table instead of performing JOIN operation on multiple tables. This will significantly reduce the execution time.

As mentioned earlier, use of machine learning in partitioning can automate the process while still maintaining the ACID properties.

For the sake of designing a prototype, we consider only a single client server environment. In case of real time hybrid architecture based applications, the mechanism can be replicated on each client server subsystems. Since we do not deal with distributed level, we assume some sort of replication or fragmentation mechanism to be already present at that level.

Figure 1 shows the overall system architecture of the proposed system.

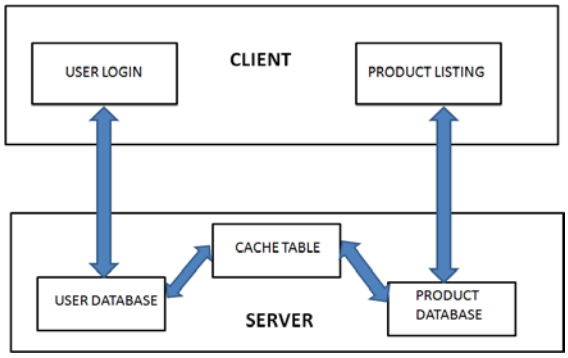

Figure 1: System architecture based on the retail framework

2.3 ALGORITHM

We use the basic classification technique based on the counter value to determine whether the data object can be classified to the new partition or not. The partitioning mechanism for placing the product data object is described below :

1. Get the Product name search string
2. Check the counter attribute of the Product
3. If counter value > threshold value, mark the product data object
4. If product is searched once more than its counter value, add the product to cache table.
5. If product data object exists in cache table, display result from cache otherwise goto Step 2.

We illustrate the technique further with an example. Consider a product X which is to be searched. Consider that the threshold value for a particular value is set to be 3 and the counter value for X is initially 0. When searched for the 4th time, the product is added to the cache table i.e. on a new partition. When the product is searched for the 5th time, the details are accessed from the new partition rather than multiple JOIN operations on the relevant table.

3. IMPLEMENTATION

A 337kb dataset for building the system prototype was obtained from the open source solution *www.opencart.com.*
A relational database consisting of multiple tables is the global data structure in the system. The transaction query is stored in a log table which is referred by the system during its execution.

3.1 RISK ANALYSIS

The risks that may hinder the performance of the system were analyzed within the constraints of time and quality. The overall impact of these risks was considered during the system implementation.

Table 2: Risk Analysis for the system

Description	Probability	Impact
Server Failure	Moderate	High
Storage Constraints	Low	High
Product not placed in cache	Moderate	High

4. RESULTS

The system described in the paper showed an improvement of 61% in performance when a data object was accessed from the cache.

When searched for the first time, the product *SamsungSyncMaster 941BW62* took **0.0120029s**. The counter threshold was set to 3. After searching for the 4th time, the product was added to the new partition. When searched for the 5th time, the time take was only **0.0046410s.**

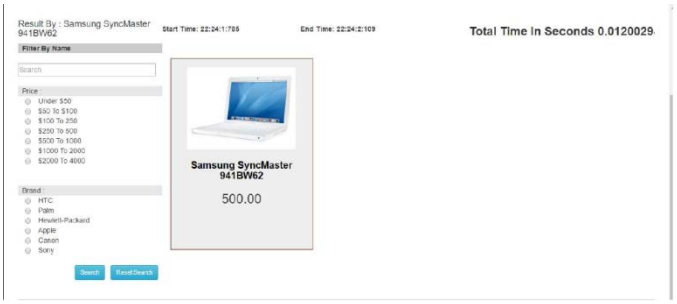

Figure 2: Product searched for the first time

Figure 2: Product search result after it was accessed from the cache partition.

Table 3 gives a brief summary of product counter values and the corresponding time required to display the search results.

Table 3 : Summary of product search results

Counter Value	Search Time
1	0.0120029
2	0.0120031
3	0.0110432
4	0.0204588
5	0.0046410

5. CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSION

Modern Transactional systems sacrifice consistency to achieve high scalability and increased performance. Partitioning Strategies at client-server level are not optimized and significantly degrade the performance. The caching mechanism used in our system is an automated process. It is found to considerably reduce the time to fetch product searched by the user. It does so by avoiding complex table JOIN operations on every search query. Hence, our proposed system used a machine learning based automated data partitioning technique for client server machines to achieve high performance.

5.2 FUTURE SCOPE

The performance of the system can be more accurately analyzed by comparing the efficiency at different data sizes. The partitioning technique proposed in the project can be modified for application at distributed level. A hybrid mechanism involving replication, fragmentation and the technique described in the paper can be considered for more enhancements.

ACKNOWLEDGMENTS

I would like to thank Prof. Dr. S.S. Sane and Prof. W.W. Pingle from the Department of Computer Engineering, K.K.

Wagh Institute of Engineering Education and Research, Nashik, India for their helpful comments and pointers for the literature survey.

References

[1] G. DeCandia et al. Dynamo: Amazon's highly available keyvalue store". In SOSP '07

[2] F. Chang et al. Bigtable: a distributed storage system for structured data. In OSDI '06

[3] B. F. Cooper et al. Pnuts: Yahoo!'s hosted data serving platform. VLDB '08.

[4] Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., and Lewin, D. 1997. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In Proceedings of the Twenty-Ninth Annual ACM Symposium on theory of Computing (El Paso, Texas, United States, May 04 - 06, 1997). STOC '97. ACM Press, New York, NY, 654-663.

[5] Blaise Gassend et al. Caches and Merkle Trees for Efficient Memory Authentication. In HPCA'03.

[6] A. Turcu, R. Palmieri, and B. Ravindran. Automated data partitioning for highly scalable and strongly consistent transactions. In SYSTOR '14.

[7] S. Papadomanolakis and A. Ailamaki. Autopart: automating schema design for large scientific databases using data partitioning. In SSDBM '04