Prepared Scan: Efficient Retrieval of Structured Data from HBase

Francisco Neves Ricardo Vilaça José Pereira Rui Oliveira
francisco.t.neves@inesctec.pt, {rmvilaca,jop,rco}@di.uminho.pt
HASLab - INESC TEC & University of Minho
Braga, Portugal

ABSTRACT

The ability of NoSQL systems to scale better than traditional relational databases motivates a large set of applications to migrate their data to NoSQL systems, even without aiming to exploit the provided schema flexibility. However, accessing structured data is costly due to such flexibility, incurring in a lot of bandwidth and processing unit usage. In this paper, we analyse this cost in Apache HBase and propose a new scan operation, named Prepared Scan, that optimizes the access to data structured in a regular manner by taking advantage of a well-known schema by application. Using an industry standard benchmark, we show that Prepared Scan improves throughput up to 29% and decreases network bandwidth consumption up to 20%.

CCS Concepts

•Software and its engineering → Cloud computing;

Keywords

NoSQL; structured data; performance optimization

1. INTRODUCTION

In recent years, processed and exchanged data among connected devices grew exponentially, leading traditional relational databases to reveal their limitations in scaling properly. Those limitations led to design and development of more scalable database systems, namely NoSQL data stores [1, 7, 3].

NoSQL data stores are usually defined as non-relational databases because they store schema-less data in a *shared nothing* fashion. These systems favor data denormalization, meaning duplicated portions of data exist across multiple tables. Besides the flexibility of NoSQL data stores provided by the schema-less characteristic, several applications migrate from relational databases to NoSQL databases exclusively for scalability purposes, keeping their schema untouched. However, the existing NoSQL databases rely on simplified and heterogeneous query interfaces, that constitute a barrier on their adoption. Projects like Google BigQuery, Hive [5], DQE [10] or Tenzing [8] try to mitigate this constraint by providing an interface based on SQL over a MapReduce framework and a key-value store. On a different perspective, other approaches make use of object mapping tools that allow to bypass the database lower level interfaces. By using [4], the user has at its disposal the generic object interfaces like JPA and JDO that allow it to use NoSQL databases in an almost transparent way, leveraging the knowledge already existent in the area.

NoSE [9] is a system for recommending database schemas for NoSQL applications. NoSQL uses a cost-based approach based on a novel binary integer programming formulation to guide the mapping from the application's conceptual data model to a database schema. In [6] the authors present a solution for the group of columns in a large data table into column families in order to increase the performance of query processing.

Existing work ease the migration of legacy SQL applications to NoSQL databases but without full exploit of schema flexibility provided by NoSQL systems. However, accessing structured data is costly due to such flexibility, incurring in a lot of bandwidth and processing unit usage. Operations like the table's *scan* make more evident the negative impact in performance associated with redundant meta-information that these systems store and process.

We propose a new operation named Prepared Scan for Apache HBase, a non-relational, distributed and open-source database. It is an alternative to the native *scan* operation that decreases the amount of data returned to client in applications that store their data structured in a regular manner.

2. PREPARED SCAN

Apache HBase is a distributed, scalable and open-source non relational database. Inspired by Google BigTable, it can be thought as a multi-dimensional sorted map indexed by the triple: row key, column name and timestamp. Row keys might have an unbounded and dynamic number of qualifiers, grouped into column families, that are created in runtime as long as new key-value pairs are inserted. Each column is identified by *family:qualifier*. Row key and value are arbitrary not-interpreted byte arrays and data is maintained in a lexicographic order by the row key. Additionally, the timestamp field is used to keep several versions of key-value pairs. The whole key-space is horizontally partitioned into Regions and distributed across several nodes named RegionServers. To access data, *get*, *put*, *scan* and *delete* operations

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SAC 2017, April 03-07, 2017, Marrakech, Morocco
© 2017 ACM. ISBN 978-1-4503-4486-9/17/04...$15.00
DOI: http://dx.doi.org/10.1145/3019612.3019863

are provided.

The scan operation is optionally applied to a specific row range and returns key-value pairs to client including all mentioned fields. This is still true even if the set of desired columns is passed to the scan operation, considering that the application knows how data is structured. Since the native scan does not take in account the structure of stored data, it also fetches and returns meta-information of key-value pairs.

The new operation Prepared Scan aims to optimize accesses to structured data in Apache HBase by taking advantage of the knowledge about data in applications. It extends the client-server communication protocol through the implementation of an endpoint co-processor, which is similar to stored procedures of traditional relational databases. This operation consists in the following three steps: preparation, execution and conclusion.

Client Interface. The client interface was designed in order to match the mentioned steps. The preparation step requires a scan instance that specifies, among other options, the set of columns for further optimization of returned results. The execution step requires a row key range and a cache hint related the amount of returned rows. In the end, as the native scan does, it returns a ResultScanner for further iteration. Moreover, this step can be called several times after a single prepare. For instance, clients may scan data that respects the schema already defined in the preparation step but changing only the row key range throughout consecutive executions.

In the conclusion step, all state and allocated resources in preparation step are freed.

Since this interface is similar to the native scan interface, it requires only small changes to migrate from the native scan to this new optimized scan.

2.1 Implementation

Most of the functionality of Prepared Scan falls into an endpoint co-processor implementation, which relies on Google Protocol Buffers for data serialization.

In this new operation, we follow the same approach as the native scan follows in terms of how results are returned to client. This means we only serialize meta-information using Protocol Buffers and send results as they are. However, natively, endpoint co-processors force messages to be serialized utilizing Protocol Buffers, degrading the performance when operations returns a lot of data. In order to avoid it, we must use external connections to behave the same way as the native scan.

Preparation. In preparation step, a connection to the table where a given *scan* is scoped is created. Through this connection, and since it is not possible to predict the row range where the scan instance will be applied in further steps, we sent the Scan instance to all active Regions of the connected table. When a given Region receives a preparation message with Scan instance, it is stored in the region-level shared map, indexed by the identification of each client. Afterwards, an unique external connection is created between a client and each one of target RegionServers.

Execution. An execution request message containing those parameters is sent to the first eligible Region in the specified row range. In its turn, the Region merges the parameters in the Scan instance previously stored in preparation step.

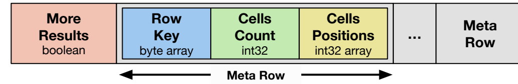

Figure 1: Serialized results meta-information

An InternalScanner, a scanner that executes under the scope of a RegionServer, is then called with the prepared Scan instance. Each returned row is processed in order to remove its meta-information. The InternalScanner finishes when either the caching value is hit or there are no more results in current Region that satisfy the Scan parameters. When latter verifies, the following eligible Regions are the target of next iterations.

Two important tasks are performed to each cell of rows in optimization phase. First, each column is replaced by its index in respect with the schema known by both client and server. Secondly, the row key is removed from all cells but first. In the end of execution, all cells got rid of column names and repeated row keys.

The meta-information of optimized results is built following the format illustrated in Figure 1. It shows a *boolean* first field, `More Results`, that represents the existence or nonexistence of more results in current Region, preventing another RPC call to this end. This field is always `false` if InternalScanner is exhausted. Followed by this field, for each row, three fields are included: `Row Key` is the row key where cells belong to, `Cells Count` is the number of row's cells and, finally, `Cells Positions` is a list of indexes for each cell in schema. This meta-data is serialized using Protocol Buffers and then sent to client. In contrast, results are reduced only to their timestamps, type of cell and values, serialized and sent through the direct connections created in preparation step. In the end, client reads the meta-information from the endpoint co-processor's response and use it to collect and rebuild the final results from its connection to the RegionServer.

Conclusion. In the last step, all allocated resources, specially the created connection and stored Scan instance in preparation step, are freed.

3. EVALUATION

Prepared Scan was evaluated using the industry standard benchmark Yahoo! Cloud Serving Benchmark[2] and compared with the native scan and GZip-enabled scan.

In this evaluation, we use Apache HBase 1.0.0 and Apache HDFS 2.6.0 with one Master and two RegionServer nodes, respectively allocated with NameNode and DataNodes. Each node has an Intel Core i3-2100 processor and 8GB RAM. Each RegionServer has 4GB of *heap memory.* Clients run in a dedicated node, which hardware specification is the same as mentioned nodes. All nodes are connected with 1Gbps network bandwidth.

In order to support this operation, we extended the original YCSB code replacing the native scan with Prepared Scan. Each client prepares a scan once at the beginning of the execution, since the used schema does not change throughout the benchmark process.

We create a single table in HBase with one million rows, using YCSB, each one with ten key-value pairs containing a row key which size ranges between 5 and 10 bytes and 1-byte column. Finally, each value is sized in 100 bytes. The whole key-space is equally partitioned based on number of rows.

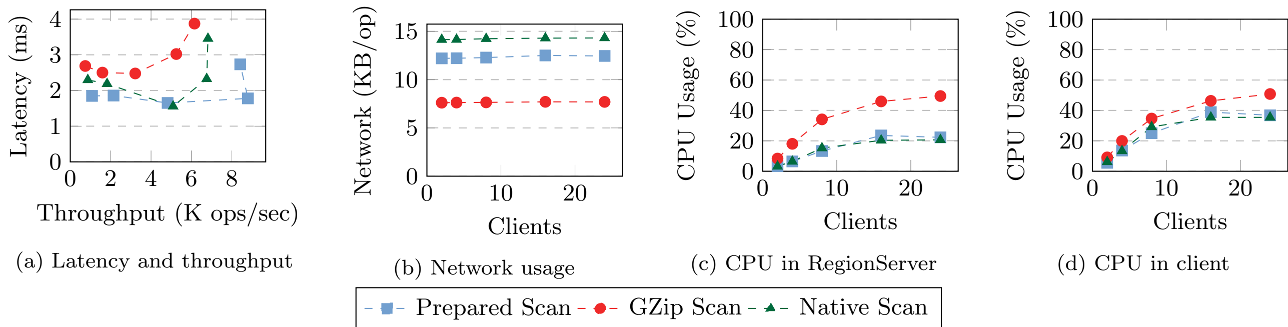

Figure 2: Results for 10 rows scan with minimal columns

3.1 Results

This evaluation aims to compare the network bandwidth usage by the Prepared Scan and the native scan with and without compression algorithms enabled. To do so, we collected performance and resource load metrics using Dstat.

We ran scan operations for 2, 4, 8, 16, and 24 concurrent clients. Figure 2 shows the results scan with 10 rows.

Figure 2a shows that the native scan hits its maximum throughput with 16 clients. In general, its latency is always between the latency of GZip Scan and Prepared Scan. In its turn, GZip Scan starts with higher latency and keeps it until its maximum throughput. Prepared Scan keeps the lowest latency when compared with previous two types of scan. It starts to increase throughput notably after 8 clients, showing up an increase of 29% in throughput compared with the native scan.

For the purpose of this work, we also present here network usage. The network usage per scan operation is depicted in Figure 2b. The native scan is the operation that uses most of the available network bandwidth (1Gbps) to return results to client. In contrast, GZip Scan decreases the amount of data up to 46%, which is the best result. Also, Prepared Scan saves up to 13% of returned data compared to the native scan without compression.

The last metric is CPU usage, which is also important to consider in compression algorithms. As expected, Prepared Scan uses more CPU than the native scan without compression, since there is an associated cost to pre-process of results in RegionServer and rebuilt them in client side. However, GZip Scan is clearly the operation that uses more CPU to compress all data returned to client.

4. CONCLUSION

In this paper we presented a new operation for Apache HBase that aims to improve the efficiency of retrieving structured data from this NoSQL database. It allows HBase to be aware of the structure of data and discard the meta-information that is not necessary due to redundancy, such as column names and row keys, in each key-value pair, decreasing network bandwidth usage up to 20% and increasing throughput up to 29%. GZip Scan is also good to decrease network bandwidth consumption. However, it requires a lot of CPU to perform as good as Prepared Scan.

5. ACKNOWLEDGMENTS

The research leading to this publication was funded by the European Commission's H2020 under grant agreement number 732051, CloudDbAppliance project. This work is also financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia as part of project UID/EEA/50014/2013.

6. REFERENCES

[1] R. Cattell. Scalable SQL and NoSQL Data Stores. *SIGMOD Rec.*, 39(4):12–27, May 2011.
[2] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.
[3] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *SOSP'07*, 2007.
[4] P. Gomes, J. Pereira, and R. Oliveira. An object mapping for the Cassandra distributed database. In *Inforum*, 2011.
[5] A. Hive. http://hive.apache.org.
[6] L.-Y. Ho, M.-J. Hsieh, J.-J. Wu, and P. Liu. Data partition optimization for column-family nosql databases. In *IEEE International Conference on Smart City/SocialCom/SustainCom*, 2015.
[7] A. Lakshman and P. Malik. Cassandra - A Decentralized Structured Storage System. In *LADIS'09*, 2009.
[8] L. Lin, V. Lychagina, W. Liu, Y. Kwon, S. Mittal, and M. Wong. Tenzing: A SQL Implementation On The MapReduce Framework. 2011.
[9] M. Mior, K. Salem, A. Aboulnaga, and R. Liu. NoSE: Schema Design for NoSQL Applications (to appear). In *IEEE 32nd International Conference on Data Engineering (ICDE)*, 2016.
[10] R. Vilaça, F. Cruz, J. Pereira, and R. Oliveira. An Effective Scalable SQL Engine for NoSQL Databases. In J. Dowling and F. Taïani, editors, *Distributed Applications and Interoperable Systems*, volume 7891 of *Lecture Notes in Computer Science*, pages 155–168. Springer Berlin Heidelberg, 2013.