Jin ZH, Shi H, Hu YX *et al.* CirroData: Yet another SQL-on-Hadoop data analytics engine with high performance. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 35(1): 194–208 Jan. 2020. DOI 10.1007/s11390-020-9536-z

CirroData: Yet Another SQL-on-Hadoop Data Analytics Engine with High Performance

Zheng-Hao Jin[1], Haiyang Shi[2], Ying-Xin Hu[1], Li Zha[3,4], *Member*, *CCF*, and Xiaoyi Lu[2], *Member*, *ACM*, *IEEE*

[1] *Business-Intelligence of Oriental Nations Corporation Ltd., Beijing 100102, China*

[2] *Department of Computer Science and Engineering, The Ohio State University, Ohio 43210, U.S.A.*

[3] *Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China*

[4] *University of Chinese Academy of Sciences, Beijing 101408, China*

E-mail: jinzhenghao@bonc.com.cn; shi.876@osu.edu; huyingxin@bonc.com.cn; char@ict.ac.cn; lu.932@osu.edu

Received July 15, 2019; revised October 14, 2019.

Abstract This paper presents CirroData, a high-performance SQL-on-Hadoop system designed for Big Data analytics workloads. As a home-grown enterprise-level online analytical processing (OLAP) system with more than seven-year research and development (R&D) experiences, we share our design details to the community about how to achieve high performance in CirroData. Multiple optimization techniques have been discussed in the paper. The effectiveness and the efficiency of all these techniques have been proved by our customers' daily usage. Benchmark-level studies, as well as several real application case studies of CirroData, have been presented in this paper. Our evaluations show that CirroData can outperform various types of counterpart database systems in the community, such as "Spark+Hive", "Spark+HBase", Impala, DB-X/Y, Greenplum, HAWQ, and others. CirroData can achieve up to 4.99x speedup compared with Greenplum, HAWQ, and Spark in the standard TPC-H queries. Application-level evaluations demonstrate that CirroData outperforms "Spark+Hive" and "Spark+HBase" by up to 8.4x and 38.8x, respectively. In the meantime, CirroData achieves the performance speedups for some application workloads by up to 20x, 100x, 182.5x, 92.6x, and 55.5x as compared with Greenplum, DB-X, Impala, DB-Y, and HAWQ, respectively.

Keywords CirroData, high performance, SQL-on-Hadoop, online analytical processing (OLAP), Big Data

1 Introduction

Many data processing systems are deployed in modern data centers to handle both operational workloads (i.e., many small transactions with a high portion of updates, known as online transaction processing or OLTP) and analytical workloads (i.e., complex queries traversing large volume of data, known as online/offline analytical processing or OLAP).

Fig.1 represents a typical data processing architecture on modern data centers. In this architecture, production systems keep generating a large volume of data, or Big Data into a layer of OLTP relational databases. These OLTP databases construct the data sources for data analytics jobs from different OLAP systems.

To meet the requirements of different analytics, we usually deploy multiple OLAP systems on top of the same collection of data sources within OLTP databases. The box of data processing cluster in Fig.1 shows that a batch data processing cluster with Apache Big Data Stacks (i.e., ABDS[1,2]), such as Hadoop[3], Spark[4], Hive[5], Impala[6], is usually built up to run customized data analytics jobs over PB-level (petabyte-level) or even larger datasets, which cannot be easily handed on traditional symmetric multi-processing (SMP) or massively parallel processing (MPP) database systems.

Some of the data stored in OLTP databases and the results from batch processing can be stored in a

Regular Paper

Special Section on Applications

This research is supported in part by the Strategic Priority Research Program of the Chinese Academy of Sciences of China under Grant No. XDA19020400.

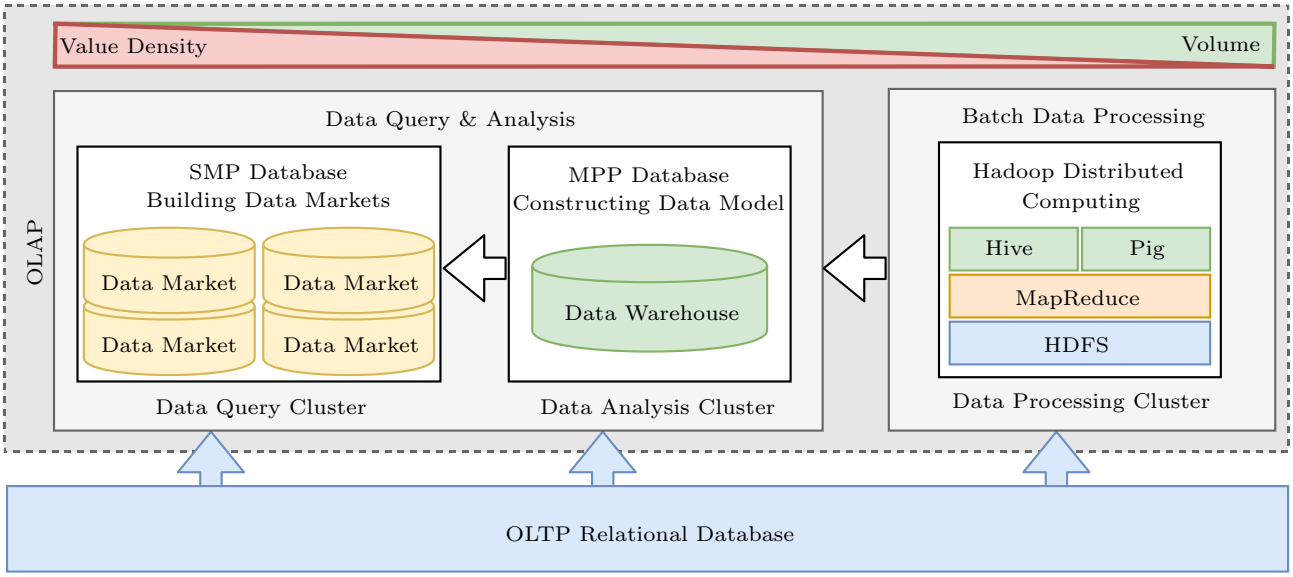


Fig.1. Typical OLTP and OLAP workloads running on modern data centers.

data warehouse system backed by an MPP database, which is shown in the box of data analysis cluster in Fig.1. The data warehouse system can support rich ETL (i.e., extract, transform and load) operations and construct meaningful data models for decision making. The data warehouse system usually cleans the raw data and significantly reduces the size of data to store. To support fast OLAP queries, the data in the warehouse system can be further analyzed and reduced to form up many data markets, which can be stored in SMP database systems, as shown in the box of data query cluster in Fig.1. The value density of the data stored in SMP databases is much higher than that of the partially processed or raw data. Fig.1 summarizes how the data flows from the OLTP side to the OLAP side and is processed in different OLAP systems on modern data centers.

Among these OLAP systems, Apache Hadoop has become the de-facto standard software infrastructure for building cheap and scalable OLAP systems. Even though Hadoop MapReduce seems losing market share due to the increasing popularities of the strong competitors such as Spark and Flink, Hadoop has still established itself as the most reliable and productive software base for OLAP systems. One of the most important reasons for this is the wide adoption of its distributed file system layer, HDFS.

However, achieving high_performance Big Data analytics on Hadoop-based OLAP systems is a nontrivial issue, because there are many system-level performance related factors such as data locality, distributed coordination, task scheduling, and so on, which may become significant bottlenecks. In the meantime, there are not many design articles from the companies to expose their design architecture and technical details. This situation could become a burden to preventing researchers from academia and engineers from industry to exchange their ideas on designing efficient SQL-on-Hadoop OLAP systems.

In this paper, we present a seven-year-old product — CirroData, which is a high-performance OLAP data analytics engine designed with the SQL-on-Hadoop architecture. With achieving high performance and flexibility as the major goals in mind, we have proposed many optimized designs in CirroData, such as distributed SQL plan execution, runtime code generation with LLVM, distributed metadata management, locality-aware and load balanced task scheduling, hybrid row-column partitioning, and so on.

Our evaluation results show that CirroData can outperform various types of counterpart database systems in the community, such as “Spark+Hive”, “Spark+HBase”, Greenplum, Impala, DB-X/Y, and HAWQ [7] in many different representative application scenarios. For instance, CirroData can achieve up to 4.99x speedup compared with Greenplum, HAWQ, and Spark in the standard TPC-H queries. For some application scenarios, CirroData can achieve the performance speedups by up to 20x, 100x, 182.5x, 92.6x, and 55.5x as compared with Greenplum, DB-X, Impala, DB-Y, and HAWQ, respectively.

To summarize, this paper has made the following

contributions.

• We present a real SQL-on-Hadoop database product, CirroData. We share our design and development details to the community about how to achieve high performance and flexibility in CirroData.

• We discuss several optimization techniques in CirroData with details. The effectiveness and the efficiency of all these techniques have been proved by our customers' daily usage.

• We present both benchmark-level and real application-level studies of CirroData with insightful performance numbers taken from multiple real cluster deployments.

The rest of the paper is organized as follows. Section 2 presents the overall architecture and key components for CirroData. Section 3 discusses the system optimization in CirroData. Section 4 and Section 5 describe our detailed evaluations with TPC-H benchmark queries and several real application workloads respectively. Section 6 discusses related studies. Finally, we conclude the paper in Section 7.

2 CirroData Design Overview

This section presents the system architecture and key components of CirroData.

2.1 Bird's-Eye View of CirroData

Fig.2 describes the overall software architecture of CirroData. As a production OLAP system, CirroData has to support all kinds of requirements from both application execution and system maintenance perspectives. CirroData includes five core engines (i.e., front end engine, distributed query plan engine, distributed scheduling engine, query engine, and distributed storage engine) and three common services (i.e., cluster state management, monitoring & controlling tools, and metadata management).

The front end engine is responsible for authenticating user accesses, token dispatching, session management, handling JDBC requests, etc. The distributed query plan engine is responsible for SQL parsing, semantic checking, query planning and optimization, etc. The distributed scheduling engine is in charge of the dispatch of query execution plans, tracking task executions, data distribution across computing nodes, etc. The query engine is responsible for converting the logic execution plans to physical plans and reserving the required resources to complete the executions. The execution results in query engines will be returned back to upper layer components by the distributed scheduling engine. The distributed storage engine provides data storage service and supports high-performance read/write mechanisms, including efficient data organization, encoding, partitioning, and compression, etc. The cluster state management service provides the health status report of the whole cluster and it uses ZooKeeper to synchronize the status of each node across the cluster. Monitoring and controlling tools provide utilities for cluster deployment, node startup

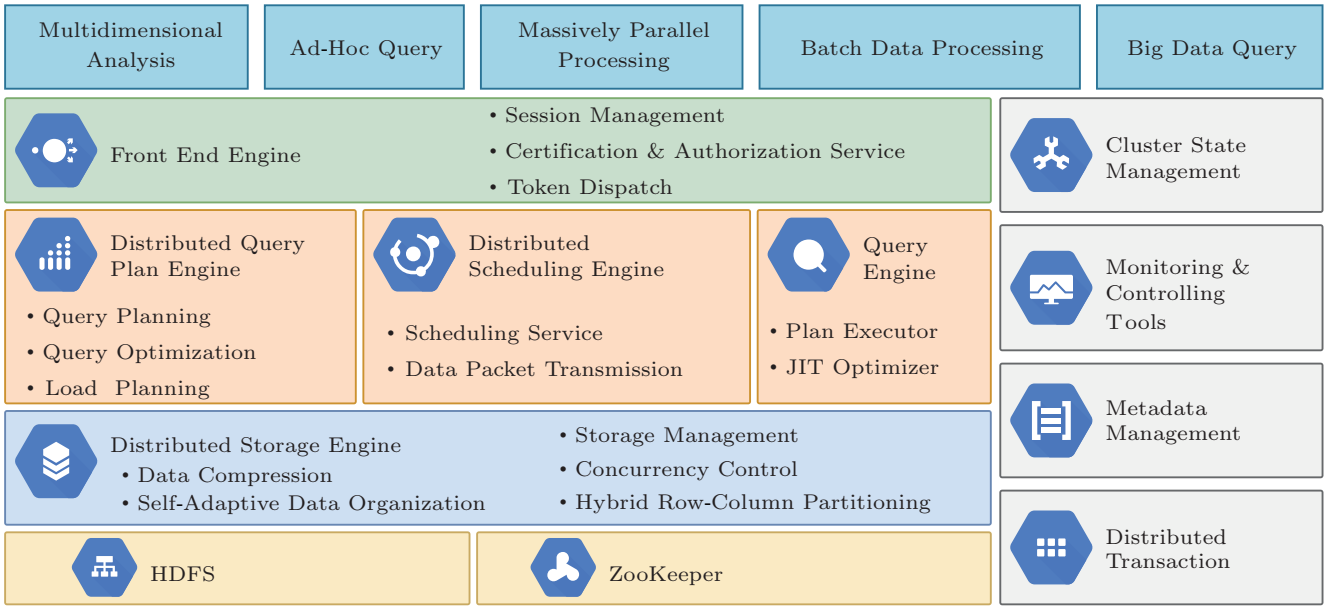


Fig.2. Software architecture of CirroData.

and shutdown, running status monitoring, resource usage monitoring, etc. The metadata management service provides metadata storage and management service. In this paper, we cannot explain all the details in each of these components due to the space constraints. We will cover some key components and optimizations in Subsection 2.2 and Section 3.

Fig.3 represents the SQL query execution flow on the five core engines in CirroData. It gives an overview of how an SQL query is executed in CirroData with the cooperation among different components. In addition to the core engines, we also highlight one important service in CirroData, i.e., metadata management, because it is providing critical support for the whole system running efficiently.

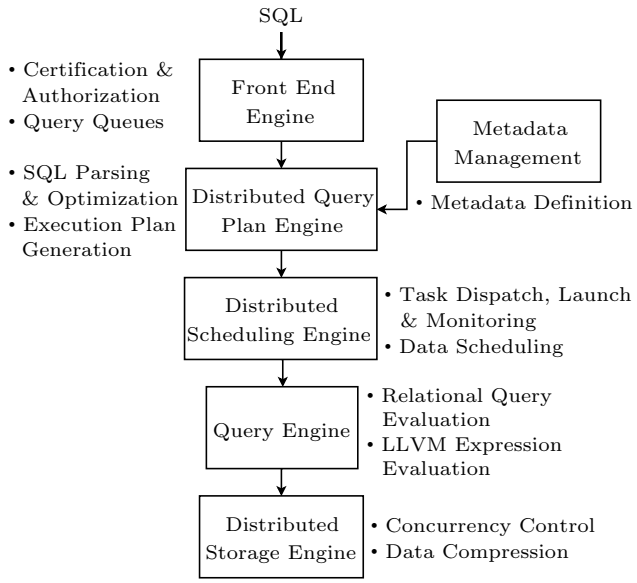


Fig.3. SQL execution flow on CirroData engines.

2.2 Key Components

Distributed and efficient data storage and query execution are the two most important aspects of designing high-performance OLAP systems. Thus, we describe these two key components in the following subsections.

2.2.1 *Distributed Storage Engine on HDFS*

To avoid data moving back and forth among systems, one key design principle of CirroData is to store all the data in a shared distributed storage engine, which is backed by HDFS. We decide to use HDFS as the backend storage engine, which is mainly because HDFS can provide good support on storage scaling, high availability, high throughput, and acceptable I/O latency. These are all important factors for designing high-performance OLAP systems. In addition, on top HDFS, CirroData can design efficient and independent workload scheduling and balancing mechanisms by taking advantages of data locality information in HDFS. In particular, CirroData instruments the HDFS block placement policy to determine where the blocks are replicated. With this locality information, CirroData can ensure fast local I/O whenever possible with the short-circuit reads in HDFS.

To achieve higher I/O performance, CirroData uses the `libhdfs` JNI library instead of the normal HDFS Java filesystem APIs to access the storage layer. This is mainly because the core execution engines in CirroData are implemented with C++ programming language, which enables CirroData to efficiently work with `libhdfs`.

To provide fast and efficient data storage, CirroData proposes a hybrid row-column data organization scheme based on Parquet[①]. The logical unit of the hybrid row-column partition is *row group*, which consists of a chunk of continuous data for each column in the table. By combining the advantages of row partitioning and column partitioning, hybrid row-column partition performs well in data compression and I/O optimization. Moreover, hybrid row-column partitioning can also improve query performance with the MinMax indexing technique. For example, range queries would be significantly accelerated if there are two fields in each row group indicating the *min* and *max* values of the data range in the row group. By being compared with these MinMax indexes, CirroData can prevent many unnecessary data accesses by skipping some row groups. CirroData supports multiple compression schemes for this hybrid row-column data organization, which can achieve around 3:1–20:1 compression ratio.

2.2.2 *YARN-Bypassed Distributed Query Execution*

The distributed query execution in CirroData enables query statements to be executed across multiple compute nodes efficiently. A typical workflow is summarized as follows. 1) The distributed query plan engine converts an incoming query statement into a distributed query execution plan, which consists of a series of sub query plans. These sub query plans are executed on compute nodes across the cluster, which may include local computations and necessary data movements via reading remote tables on other compute nodes. 2) The distributed query plan engine typically determines

[①]http://parquet.apache.org/, Nov. 2019.

an optimal set of compute nodes to process these sub query plans based on a defined cost model. 3) The distributed scheduling engine then dispatches the logical query plans to query engines, which reside on compute nodes. The query engines optimize and execute the query plans locally (shown in Fig.4). 4) The local query engine acknowledges distributed scheduling engine once a query execution is completed.

To achieve high-performance distributed query executions, the cost model for the distributed query plan engine in step 2 takes data locality, system load, and the cost for executing sub query plan into account to construct optimized execution plans. In the meantime, query plans in step 3 are compiled to machine code and cached to expedite subsequent executions, in order to alleviate compilation overhead and achieve optimal performance.

Unlike many other SQL-on-Hadoop systems, such as Vortex or VectorH [8], which are still based on in-band or out-of-band YARN scheduling, CirroData chooses to fully bypass the Hadoop YARN layer. The choice of fully bypassing YARN is mainly because CirroData aims to achieve low latency for query executions and fine-grained resource management. However, YARN cannot satisfy these requirements because the latencies of interactions with YARN services are relatively high compared with the direct communications and coordinations among CirroData components. In addition, YARN requires jobs to manage the resource consumptions in containers, which brings additional container launching and reclaiming overhead and coarse-grained resource management. In many OLAP application scenarios, we need to fully and flexibly control different kinds of resources such as memory, CPU, and I/O devices. With bypassing YARN, CirroData can combine the best properties of both MPP-style query execution and HDFS-based scalable data management into one system.

3 System Optimization

3.1 Distributed Coordination and Metadata Management

The master/slave architecture adopted in Apache Hadoop becomes popular in designing distributed data processing systems. Fig.5(a) depicts a typical topology of the master/slave architecture. At the very beginning, CirroData employed the master/slave architecture as well. However, many experiments in real deployments show that the master node easily becomes a bottleneck of the entire system if there exist more than 50–100 slave nodes and many huge tables in the cluster. It is well-known that, in the pure master/slave architecture, the master is a potential bottleneck because of its responsibility of maintaining metadata as well as coordinating slave nodes.

Motivated by this observation, instead of using the

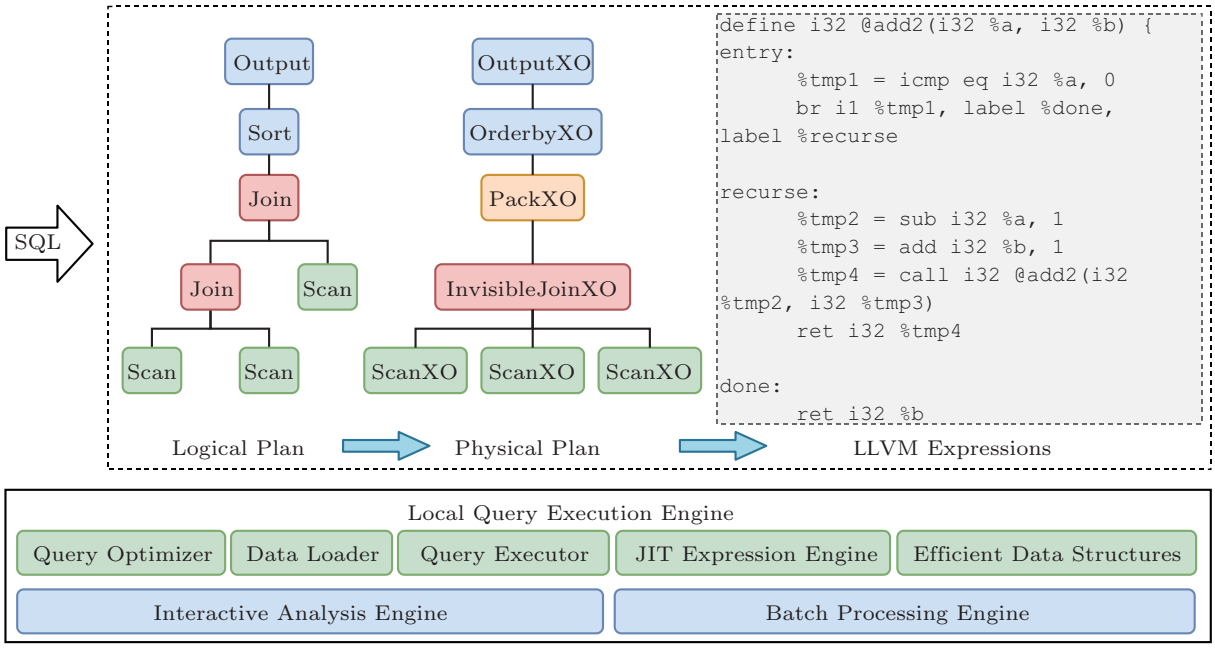


Fig.4. Overview of the CirroData's local query execution engine.

static master/slave architecture, CirroData now adopts a more dynamic and balanced architecture as illustrated in Fig.5(b). The advantages of CirroData's architecture are summarized as follows.

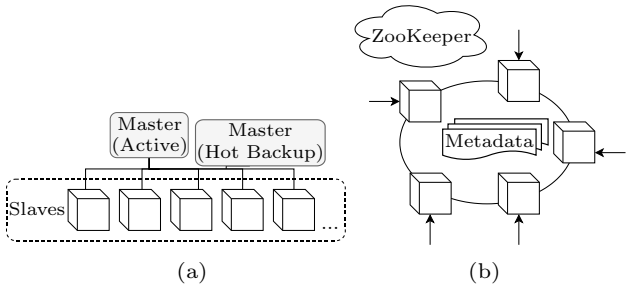


Fig.5. Architecture comparison. (a) Master/slave architecture. (b) CirroData's architecture.

• *High Concurrency.* Any compute node in CirroData can act as a global coordinator to coordinate distributed query processing. Scheduling distributed query processing among all compute nodes eliminates single-node bottlenecks and improves data balance and concurrency.

• *High Reliability.* CirroData's metadata is stored onto HDFS with the N-way replication scheme, which enables metadata to be accessed across the cluster as well as guarantees consistency and reliability.

• *Horizontal Scalability.* CirroData makes use of consistency guarantees in ZooKeeper [9] to achieve near linear scalability. Lightweight status management (e.g., active server list), distributed locking mechanism, etc., are implemented with the help of ZooKeeper's atomic read/write.

3.2 Load-Balanced and Locality-Aware Task Parallelism

Considering the advantage of massive parallel processing (MPP) and its wide employment in modern databases such as Teradata and Greenplum, CirroData simply adopted MPP design concepts to gain high horizontal scalability at the first several versions. The MPP approach splits a query statement into sub query plans which are finally executed by multiple compute nodes in parallel; thus MPP typically delivers both performance and scalability. To be able to handle huge amounts of data, the data in MPP solutions is usually split among compute nodes such that each node processes only its local data and shares nothing with other compute nodes. However, the shared-nothing assumption of MPP is not valid for CirroData, because CirroData uses HDFS as a shared storage layer as presented in Section 2. Simply using shared storage together with MPP is a huge overkill, which results in more complexity, higher network utilization, less scalability, and less parallelism.

Fig.6 depicts the distributed, load-balanced, and locality-aware task parallelism solution currently employed by CirroData. In contrast to conventional MPP

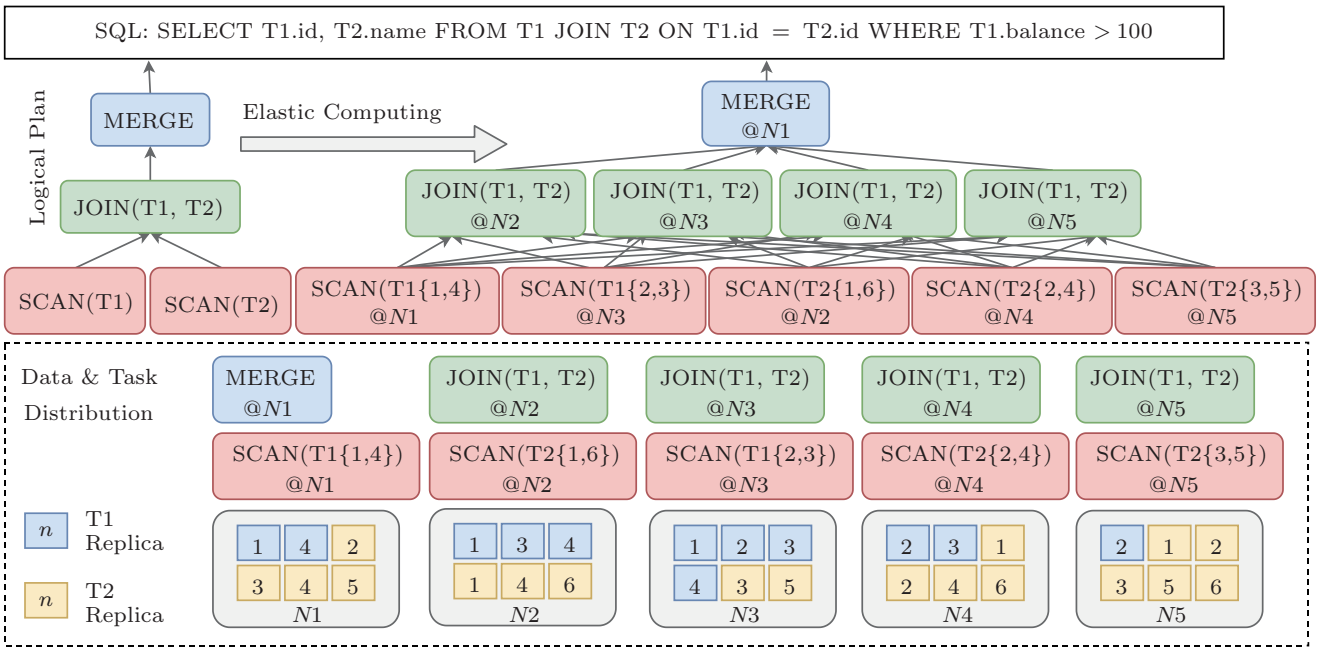


Fig.6. Task parallelism in CirroData.

designs, the CirroData design gives more fine-grained data slicing and distributed query plan scheduling. The most frequently used data slicing policy is using hash distribution which is able to align tables efficiently to improve the performance of some query patterns. As shown in Fig.6, with the knowledge of data distribution across compute nodes, the distributed query execution engine is able to schedule and dispatch sub query plans taking into account where the data is located. A query plan consists of several relational operators, such as *scans*, *joins*, *sorts*, *merges*. As illustrated in Fig.6, join and merge operations require data shuffling to guarantee correct results. Shuffling introduces extra CPU and network overhead, thus the distributed query plan engine often generates a plan considering: 1) the co-location of data, 2) existing data distribution, 3) system load of involved compute nodes, and 4) minimizing resource consumption by generating an appropriate number of sub query plans (e.g., joins). Overall, the task scheduling and execution design in CirroData has three optimizations compared with conventional MPP designs: 1) efficient data slicing policies, 2) locality optimization, and 3) elastic resource scheduling.

3.3 Runtime Code Generation with LLVM

Just-In-Time (JIT) query compilation has been used to eliminate the effects of interpretation overhead. On receiving a query for the first time, the query engine compiles (a part of) the query into a routine that gets subsequently executed. The Low Level Virtual Machine (LLVM) [10] has been proven to be one of the most efficient compilers to process JIT compilation in many modern databases (e.g., HyPer [11, 12], Impala [6], MemSQL [13], Vitesse DB [14]). LLVM is a collection of widely-used modules and libraries for building compilers. The LLVM libraries provide a modern target-independent optimizer working on LLVM intermediate representation (LLVM IR), along with assembly and machine code generation support for various hardware platforms (e.g., x86, ARM, PowerPC). The use of LLVM in code generation can keep the benefit of being architecture-independent, while still maintaining tight control over the generated code. Moreover, using general and specific optimizations in the LLVM optimizer results in noticeable performance improvement in query executions.

CirroData takes advantage of LLVM-based runtime code generation to produce query-specific versions of functions to improve efficiency in utilizing CPU and cache. Relational algebra trees in CirroData are compiled into efficient machine code using LLVM compiler backend.

Fig.7 presents the workflow of runtime code generation in CirroData. The workflow consists of three steps in general: 1) parsing query statements into abstract syntax trees (AST) by applying lexical and syntactic analysis, 2) generating and optimizing intermediate representations (IR) according to the data structure of ASTs, and 3) caching IRs and executing queries by calling Just-In-Time (JIT) functions. The performance benefit comes from JIT compilation in step 2 (i.e., IR

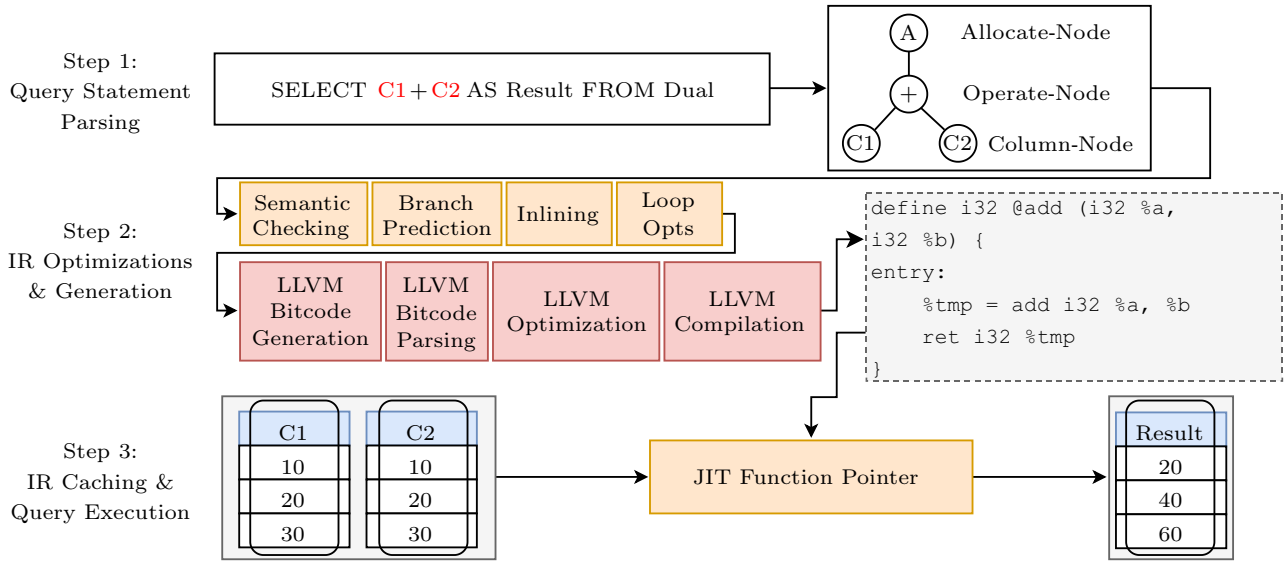


Fig.7. Workflow of runtime code generation in CirroData.

generation and optimization), which tries to maximize data locality and predictable branches, and prefers inlined functions rather than having a lot of function calls and method dispatches.

Fig.7 shows a representative query in which runtime code generation speeds up the execution significantly. The add operation summing up two columns must be called for every record in every data file scanned. For a data file consisting of billions of records or more, the efficiency of the add operation will therefore be critical to the query performance of CirroData. In the meantime, since runtime information is not known at compile time (e.g., it is unknown if the add operation is to handle integers, strings or floats), the add operation can only be designed in a general-purpose manner, which is sub-optimal. With code analysis and runtime information, JIT optimizer and compiler can make the add operation to be inlined and parameterized into some specific type, like integers. Therefore, the performance of the example query gains significant improvement. Overall, runtime code generation in CirroData results in large query speedups by eliminating branches, unrolling loops, propagating constants, inlining functions, etc.

4 Evaluation with TPC-H Query Workload

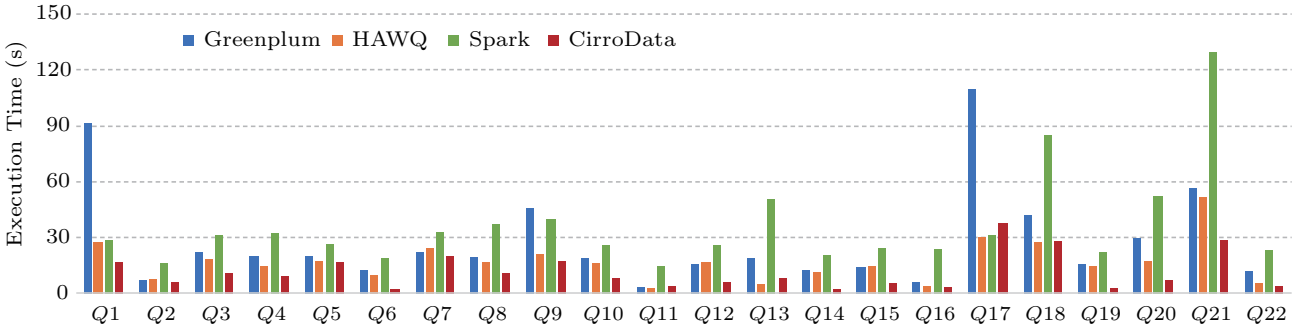
In this section, we conduct our experiments on the standard TPC-H benchmark with a scale factor of $SF30$. In these experiments, we compare the execution time of CirroData with Greenplum (version 5.12.0), HAWQ (version 2.3.0.0), and Spark (version 2.2.0). Our cluster for the TPC-H benchmark consists of five nodes, each of which is equipped with 256 GB memory. One of the five nodes acts as namenode/master, and the other four nodes work as datanodes/segments/workers.

In the case of Greenplum, we test both row-oriented and column-oriented storage types, which are both supported by Greenplum. For the experiments on the row-oriented storage type, $zlib(5)$ compression algorithm is employed to compress the tables. On the other hand, the tables are not compressed during the experiments on the column-oriented storage type. Our results of Greenplum experiments show that the column-oriented storage type outperforms the row-oriented for all the TPC-H queries. Therefore, we only include the results for column-based Greenplum in Fig.8.

In the meantime, tables are configured as $ORIENTATION = PARQUET$, $COMPRESSTYPE = SNAPPY$ for the experiments with HAWQ. While there are many configurations available for Spark, we choose a small but optimal set of configurations for our experimental cluster.

Fig.8 gives the execution time of Greenplum, HAWQ, Spark, and CirroData for all TPC-H queries. Among these four databases, CirroData outperforms the other three in sequential scan queries (i.e., $Q1$ and $Q2$), index scan queries (i.e., $Q4$ and $Q14$), hash join queries (i.e., $Q3$, $Q7$, $Q8$, etc.), sort queries (i.e., $Q5$, $Q6$, $Q12$, etc.) and most of nested loop joins (except for $Q11$ and $Q17$). Quantitatively, compared with the other three databases, CirroData gains a speedup of 1.28x–1.64x for sequential scan queries, 1.54x–4.99x for index scan queries, 1.30x–4.99x for hash join queries, 1.21x–4.60x for nested loop join queries, and 2.47x–4.60x for sort queries.

For a traditional MPP architecture based system, such as GreenPlum, its computation and storage en-



	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20	Q21	Q22
Greenplum	90.96	7.41	22.48	19.79	19.79	12.81	22.71	19.48	45.64	18.86	3.35	15.53	18.85	12.87	13.68	6.16	109.33	42.05	16.07	29.95	56.51	12.09
HAWQ	27.57	8.00	18.44	14.31	**17.76**	9.58	24.62	17.24	21.07	16.26	**2.72**	16.81	**4.85**	11.47	14.60	**4.30**	**30.23**	**27.51**	14.73	17.42	51.89	**5.19**
Spark	28.44	16.57	31.66	32.46	26.85	18.89	32.86	36.84	39.89	26.33	14.36	26.18	50.28	20.48	24.31	23.51	**31.67**	85.26	22.67	52.33	129.35	22.91
CirroData	**16.8**	**5.8**	**10.7**	**9.3**	**17.2**	**2.4**	**20.1**	**10.8**	**17.4**	**8.3**	4.2	**6.3**	8.7	**2.3**	**5.6**	**3.9**	37.6	**27.9**	**3.2**	**7.2**	**28.3**	**4.0**

Fig.8. Performance comparisons of Greenplum, HAWQ, Spark, and CirroData on TPC-H query workload.

gines are tightly coupled. The data stored in this type of systems is usually distributed by hashing, which can cause limited horizontal scaling capability. To improve the scalability of GreenPlum-like systems, the community has proposed enhanced designs on top of it, such as HAWQ. HAWQ also chooses to use HDFS as its storage layer (like GreenPlum-on-HDFS), which can decouple the computation and storage layer. However, HAWQ is based on the master/slave architecture as depicted in Fig.5(a). The master is responsible for SQL execution plan generation, task execution scheduling and tracking, etc., which easily makes itself become a system bottleneck. Compared with them, CirroData fully decouples the computation and storage layer, and the main computation engine in CirroData is stateless while HDFS-based storage engine can scale horizontally. With these advantages, we see that CirroData outperforms both GreenPlum and HAWQ in most of the above-mentioned TPC-H queries.

In the Hadoop community, Spark-based OLAP solutions are also very popular. These solutions typically are also based on HDFS with good scalability. But their computation engines and scheduling mechanisms are mainly designed for batch processing workloads. Thus, their execution latencies are usually very high. One of the major performance bottlenecks in these systems is their data shuffling mechanisms [15, 16]. Due to the high latencies in these systems, they cannot provide the desired high performance for SQL query executions. The numbers shown in Fig.8 also demonstrate this analysis. Compared with Spark, CirroData can achieve much better performance in almost all the queries (except $Q17$).

We see that for some queries such as $Q11$, $Q13$, $Q17$, and $Q18$, CirroData's performance is on par with other systems but not fully optimized. This is because the execution engine of CirroData could potentially still have some bottlenecks for some operations. For instance, we have found that if the execution of some queries must scan the data by row, it can cause performance degradation and this situation is still being optimized. We will keep optimizing CirroData and comparing its performance with other systems by using TPC-H queries in our future work.

5 Case Studies with Real Applications

In this section, we demonstrate the performance benefits of CirroData by presenting several real application case studies, each differing from each other based on the goal to be achieved. The cluster setups involved in each case study will be introduced in the corresponding subsections.

5.1 Case Studies with Join Workloads

Multi-table based joins are very common and important demands in many of our customers' daily data analytics workloads. The goal of experiments in this subsection is for demonstrating the performance benefit of CirroData on join workloads. We choose the join workloads in company-A as examples. Company-A is one of the largest telecom companies in China and it has around several hundreds of millions active users on their systems.

Fig.9 shows two typical use cases of join operations in company-A. Fig.9(a) presents a two-table based join to analyze the trend of a particular type of user activity between two different months. Since company-A has a huge user base, the query executed in Fig.9(a) needs to join two huge tables with 110 million rows and 120 million rows, respectively. Similarly, Fig.9(b) shows another typical join query scenario in company-A, which involves three huge tables. We deploy our CirroData system on one of their clusters and run these queries with their real datasets. The cluster has 43 nodes, and each node has a 32-core 2.60 GHz Intel® Xeon® CPU, 128 GB memory, a 1 TB HDD, and a 10GigE network. Among these nodes, two nodes are deployed as namenodes, three nodes run ZooKeeper services, and the rest 38 nodes are datanodes. As shown in Fig.9, our CirroData can give very good performance for both types of joins. The full executions have been optimized to around 7.8 seconds and 11.3 seconds, respectively.

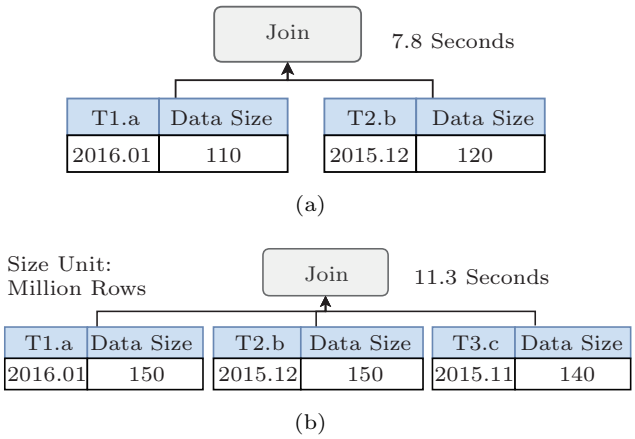


Fig. 9. Join performance of CirroData. (a) Two tables. (b) Multiple tables.

To further compare the performance of our Cir-

roData with company-A's existing deployed database system (which is Greenplum as shown in Fig.10 and Fig.11), we choose a 16-node partition from one of their production running clusters. The configuration of each node is similar to the above-mentioned cluster nodes. Fig.10(a) shows the numbers for two-table joins with an increasing number of records, from 70 million records to 200 million records for each table. As we can see from Fig.10(a), our highly optimized CirroData is able to achieve around 2.5x speedup for 200×200 million records join, compared with their current Greenplum database. In Fig.10(b), we keep each table to have 70 million records while we increase the number of tables to be joined in the queries. As we can see, CirroData can outperform Greenplum by achieving around 3x speedup. Here, the joins are hash-based joins. As described in Subsection 3.2, the distributed tasks for hash joins can be executed in a more balanced manner, which is one of the key features delivered by CirroData. In addition, as mentioned in Subsection 3.3, the expressions in join queries are compiled into more efficient machine code using LLVM compiler backend. Thirdly, CirroData organizes the data in a cache-friendly manner, which can lead to a more efficient execution pipeline. Lastly, the design of hash function and hashtable structure in CirroData has also been heavily optimized for these typical workloads. All of these optimizations contribute to the large performance gains as we see in Fig.10.

5.2 Case Studies with Aggregation and Drill-Down Workloads

Among our customers' workloads, there are also a lot of demands on high-performance aggregation and drill-down operations. The goal of experiments in this subsection is for demonstrating the performance benefit of CirroData on aggregations and drill-downs. We choose the aggregation and drill-down workloads in both company-A and company-B as examples.

Fig.11 shows the experiments we have done with company-A's real aggregation queries with their real datasets. Fig.11(a) shows the numbers of single-dimension aggregation with increasing data sizes from 100 million rows to 800 million rows. As we can see, CirroData can speed up the performance of single-dimension aggregation by 4x compared with Greenplum. Here, the cardinality of the aggregated result is 230. In CirroData, for this type of low-cardinality aggregation, we first fully pre-aggregate the records lo-

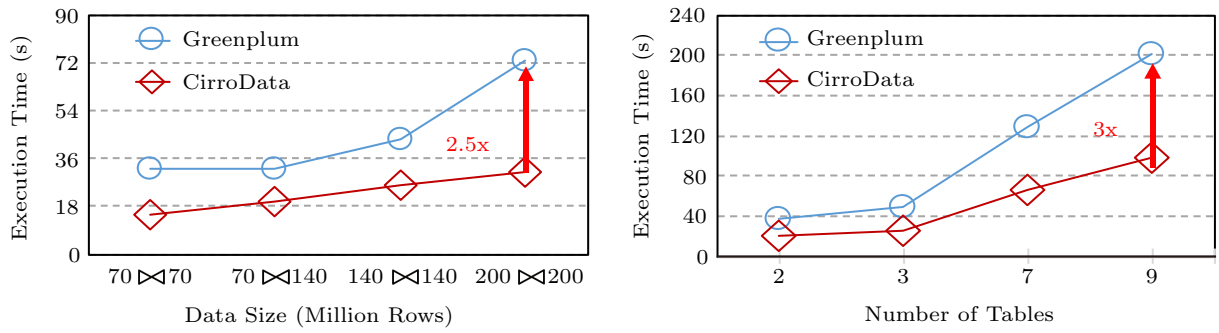


Fig.10. Join performance between Greenplum and CirroData. (a) Joining two tables. (b) Joining multiple tables.

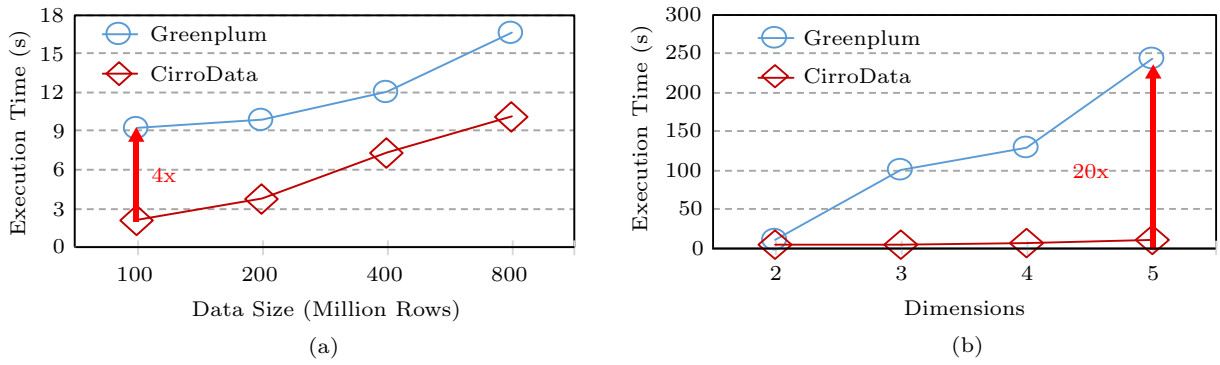


Fig.11. Aggregation performance between Greenplum and CirroData. (a) Single dimension. (b) Multiple dimensions.

cally to deduplicate the repeated records, which can significantly reduce the data size on each node since the aggregated records are much smaller than the original data. For high-cardinality aggregation, we do not spend much time on pre-aggregation since the pre-aggregated data size will be similar to the original one. After pre-aggregation, data will be sent to different nodes based on the hash mechanism, and then the system can run aggregation in parallel. To detect whether an aggregation operation has low cardinality or high cardinality, CirroData has an algorithm to probe it by running a small sampling program. In this way, CirroData can adaptively optimize the executions for both low-cardinality and high-cardinality aggregation operations. This is why we can see around 4x performance speedup in Fig.11(a).

Then, we run another set of experiments for evaluating the performance of multi-dimensional aggregations with a fixed number of records (i.e., 200 million). Fig.11(b) shows the numbers for these experiments. Interestingly, we see that with five dimensions, our CirroData can achieve 20x performance speedup compared with Greenplum. Here, in addition to the optimizations mentioned above, the benefit is also coming from our well-designed task scheduling mechanism for executing distributed computation operations in a load-balanced manner.

Company-B is a branch of another telecom company in China. We choose company-B's aggregation and drill-down workloads from company-B's daily data analysis queries and datasets. CirroData is employed on a four-node cluster and each node in this cluster has four 4-core 1.87 GHz CPUs, 8 GB memory, four 300 GB HDDs, and a 1GigE network. Before using our CirroData on their clusters, their applications were run over DB-X database system.

Performance comparisons of aggregation and drill-down analyses are depicted in Fig.12. The performance comparisons are conducted on a real dataset which consists of more than 160 million rows of records. With optimizations on distributed query processing as well as hybrid row-column based storage partitioning, CirroData outperforms DB-X by 75x–100x on processing aggregation and drill-down analyses.

Here, the DB-X is an SMP database running on a single server, while our CirroData is running over four nodes. Hence, CirroData indeed uses 4x resource compared with DB-X, but we want to highlight that the performance gain is as high as 75x to 100x. Another technical reason for the performance gain is that DB-X is based on pure row-based data storage, while CirroData is using the hybrid row-column based storage partitioning technique. The hybrid row-column storage scheme is much faster than the row-based storage scheme for these aggregation and drill-down workloads.

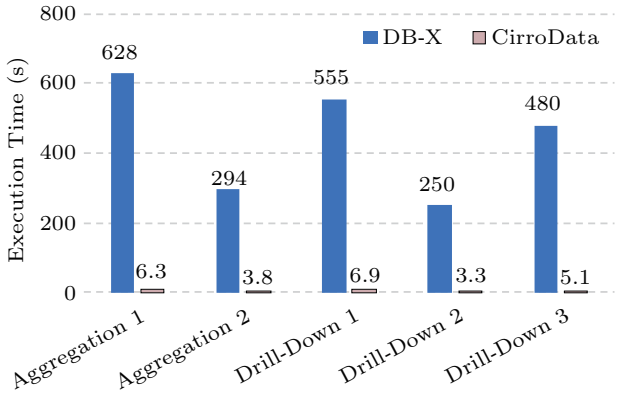


Fig.12. Aggregation and drill-down performance between DB-X and CirroData.

5.3 Case Studies for Comparing with Spark-Based Data Analytics Frameworks

In this subsection, we study query performance on real financial datasets from company-C. Company-C is one of the big financial companies in China. The goal of experiments in this part is for illustrating the performance advantages of CirroData when we compare it with Spark-based data analytics frameworks, such as "Spark+Hive" and "Spark+HBase", which are widely used in many financial companies. "Spark+Hive" means running Hive on top of Spark, on which users can directly execute SQL. "Spark+HBase" means running Spark-based analytics jobs on the data stored in HBase. The cluster we used in these tests has five nodes. Each node has 256 GB memory, 32-core 2.30 GHz CPU, a 1 TB HDD, and a 1GigE network.

Figs.13(a)–13(c) show that CirroData gains up to 2.1x, 8.4x, 6.0x speedups in querying on a single table, two tables, and four tables, respectively, compared with "Spark+Hive". Meanwhile, compared with "Spark+HBase", the speedups that CirroData can achieve reach up to 6.7x, 22.5x, and 38.8x when processing data on a single table, two tables, and four tables, respectively.

Here, the "Spark+HBase" scheme performs the worst which is mainly because HBase is not a good storage engine for processing complex queries with a large number of data accesses. HBase is mainly designed for point query and some range queries with the help of additional index techniques [17]. Compared

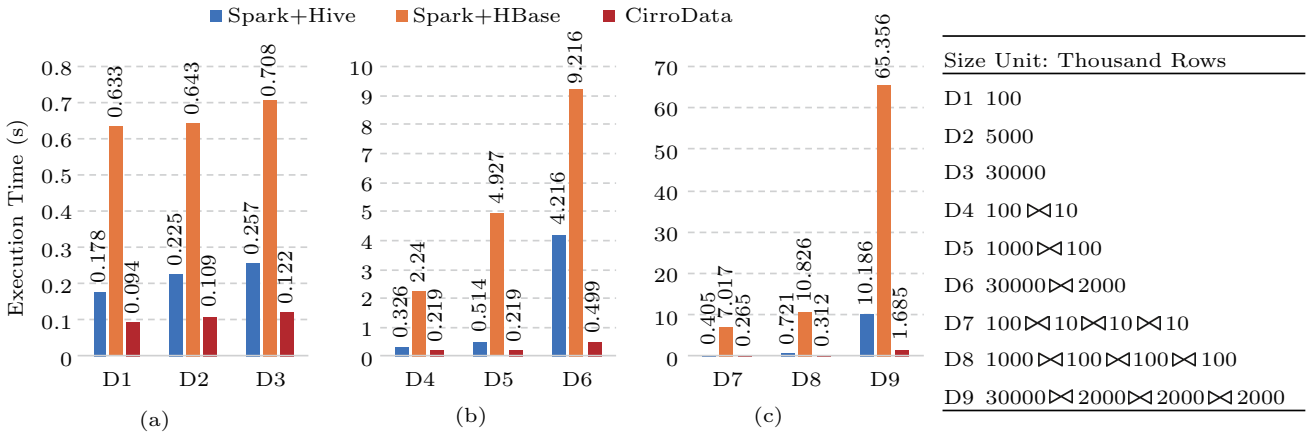


Fig.13. Query performance comparisons among "Spark+Hive", "Spark+HBase", and CirroData. (a) Single table. (b) Two tables. (c) Four tables.

with "Spark+Hive", CirroData has a better distributed query execution plan, and the tasks are scheduled in a more load-balanced manner. The tasks are executed in a pipeline fashion. CirroData can significantly reduce the I/O operations for the intermediate data exchanging, due to efficient data broadcasting and hash distributing among the tasks.

5.4 Case Studies for Comparing with MPP-Based Data Analytics Systems

In this subsection, the major goal is to show the performance benefits of CirroData when we compare it with several MPP-based data analytics systems. Table 1 presents a detailed performance comparison on a real insurance dataset from company-*D*. Company-*D* is one of the big insurance companies in China. To fully evaluate query performance, we choose several typical query statements: 1) insert, 2) select with different condition clauses, 3) update, 4) delete, and 5) multiple rounds of delete and insert. The tests are run on a five-node cluster. Each node has an Intel® Xeon® 2.30 GHz CPU with 32 cores, a 5 TB SAS HDD, 128 GB memory, and a 10GigE network.

For insert queries with no more than 100 million rows, CirroData achieves up to 4.6x, 1.2x, and 1.9x speedups compared with Impala, DB-Y, and HAWQ, respectively. In the meantime, CirroData outperforms Impala, DB-Y, and HAWQ by up to 22.8x, 5.9x and 5.9x, respectively, in inserting 1 000 million rows. In inserting 4 000 million rows, CirroData performs 9.9x better than DB-Y, while Impala and HAWQ are not able to complete the queries.

For performing select queries without group by clauses, CirroData reduces the execution time by up to 99.3% and 99.4% compared with Impala and HAWQ, respectively, though performing a little worse than DB-Y. However, CirroData outperforms DB-Y by up to 2.6x in dealing with select queries with group by clauses. Meanwhile, CirroData improves the join performance by up to 182.5x, 2.8x, and 55.5x compared with Impala, DB-Y, and HAWQ, respectively. Note that Impala and HAWQ are not able to complete joining large data sizes in our experiments.

For performance comparisons on updating and deleting operations, CirroData improves the execution time by up to 92.6x and 29.1x, respectively, compared with DB-Y. Another important observation is that CirroData is able to deal with up to 4 000 million rows for both updating and deleting efficiently, while DB-Y cannot handle these cases.

In the experiments of running multiple rounds of delete and insert queries, we first delete one million rows from the existing table and then insert the one million deleted rows into the same table in each round. The result shows that CirroData reduces the execution time by 64.4% compared with DB-Y. Note that for the update and delete experiments, both Impala and HAWQ cannot support running these tests due to missing the corresponding functions.

5.5 Case Study for Scalability of CirroData

The goal of experiments in this subsection is for illustrating the scalability of CirroData. These tests are actually run on company-*A*'s cluster as described in Subsection 5.1. Fig.14 shows the execution time changes in a cluster scaled from 5 nodes to 10 nodes. We observe that, by doubling the scale, the execution time for aggregation and drill-down analyses have been

Table 1. Performance Comparison with Other MPP-Based Data Analytics Systems

Query Statement		Data Size (Million Rows)	Impala	DB-Y	HAWQ	CirroData
Insert		1	5.30	3.01	5.820	3.046
		10	67.34	17.77	17.780	14.803
		100	277.57	171.86	134.420	139.160
		1000	11 268.34	2 938.25	2 913.050	495.000
		4000	–	6 976.37	–	704.507
Select		1	0.88	0.02	0.215	0.880
		100	46.50	0.24	60.150	0.340
	Group by 2 columns	100	6.86	0.42	63.820	1.554
	Group by 3 columns	100	6.06	1.31	68.200	4.633
		4000	–	0.04	–	4.386
	Group by 2 columns	4000	–	51.48	–	19.951
	Group by 3 columns	4000	–	66.56	–	25.204
		100 ⋈ 100	5 788.54	45.82	1 759.080	31.713
		100 ⋈ 1000	–	93.75	–	34.085
		1000 ⋈ 4000		248.67	–	116.737
		1000 ⋈ 1000 ⋈ 1000	–	236.43	–	106.729
Update		1		1.56		1.485
		10	–	222.49	–	3.724
		100	–	576.03	–	6.220
		1000	–	–	–	45.490
		4000	–	–	–	671.237
Delete		1		47.56		1.722
		10	–	93.27	–	3.432
		100	–	169.06	–	12.814
		1000	–	986.66	–	33.964
		4000	–	–	–	350.651
Delete and Insert		1000	–	7.72	–	2.749

reduced by 42%–59%. Thus, CirroData has near linear scalability in these experiments. These numbers also imply the advantages of performing task scheduling, execution, and coordination, and metadata and data management in a fully distributed manner in CirroData.

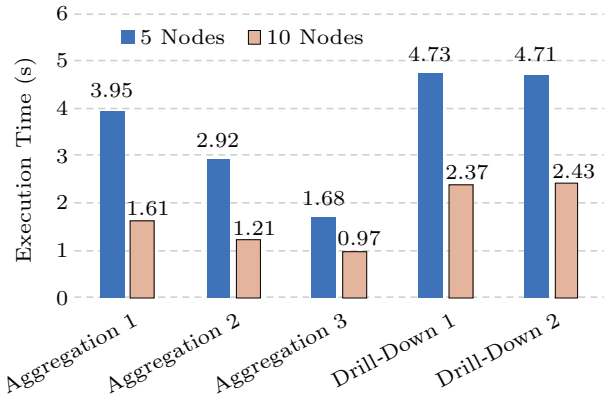


Fig.14. Scalability of CirroData.

6 Related Work

A lot of work focusing on designing high-performance and scalable database systems has been done in the past. In this section, we discuss some of the work related to the approaches taken in CirroData.

With the popularity of large-scale real-time analytic applications in both industry and academia, various vendors and academic groups have built database systems in two directions. MemSQL [13], IBM dashDB②, HyPer [18], and Peloton [19] have proposed single-engine solutions for both OLTP and OLAP requests. On the other hand, Hive [20], Impala [6], VectorH [8], and IBM Wildfire [21] have adopted SQL-on-Hadoop architecture to process OLTP and OLAP separately in two different sub-engines.

There are many studies focusing on optimizing distributed query processing. H-Store [22] has presented a system model based on the coordination of multiple single-threaded engines to provide more efficient execution of distributed transactions in main memory database systems. A Markov model based approach proposed by [23] demonstrates to be helpful for automatically selecting appropriate optimizations to process distributed transactions. Accordion [24], E-Store [25], and the design in [26] have proposed solutions to improve the elasticity of distributed transaction

②http://www.ibm.com/analytics/us/en/technology/cloud-data-services/dashdb, Nov. 2019.

and query processing. Other studies like Granola [27], SAP HANA [28, 29], Calvin [30], HAWQ [7], MemSQL [13], Impala [6] illustrate fine-grained system architectures in designing distributed database systems.

Motivated by some of the existing work, the homegrown CirroData system is designed on top of HDFS with a combination of many optimized techniques as described in Section 2 and Section 3, such as distributed coordination and metadata management, efficient data storage on HDFS, YARN-bypassed query execution, load-balanced and locality-aware task parallelism, runtime code generation with LLVM, and so on. The major design goal of CirroData is trying to support our customers' daily data analytics requirements on top of the SQL-on-Hadoop architecture with high performance.

7 Conclusions

In this paper, we presented a real database product — CirroData, which is a high-performance SQL-on-Hadoop system designed for OLAP workloads. We have shared some of our design and implementation experiences to the community about how to achieve high performance in CirroData. The effectiveness and the efficiency of CirroData have been proved by our customers' daily usage. With in-depth evaluations with TPC-H benchmarks and several real application case studies, we demonstrated that CirroData can outperform various types of counterpart database systems in the community, such as "Spark+Hive", "Spark+HBase", Impala, DB-X/Y, Greenplum, HAWQ, and others. For some workloads (e.g., join), we can see CirroData can achieve up to 182.5x performance speedup compared with other systems.

As part of future work, we plan to explore more advanced technologies on modern data center environments, such as high-speed networks, GPGPU, non-volatile memory (NVM), and NVM express based SSD. We believe these technologies will help to design a high-performance and scalable database system.

References

[1] Fox G C, Qiu J, Kamburugamuve S, Jha S, Luckow A. HPC-ABDS high performance computing enhanced Apache big data stack. In *Proc. the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2015, pp.1057-1066.

[2] Qiu J, Jha S, Luckow A, Fox G C. Towards HPC-ABDS: An initial high-performance big data stack. http://grids.ucs.indiana.edu/ptliupages/publications/nist-hpc-abds.pdf, June 2019.

[3] Shvachko K, Kuang H, Radia S, Chansler R. The Hadoop distributed file system. In *Proc. the 26th IEEE Symposium on Mass Storage Systems and Technologies*, May 2010, Article No. 9.

[4] Zaharia M, Chowdhury M, Franklin M J, Shenker S, Stoica I. Spark: Cluster computing with working sets. In *Proc. the 2nd USENIX Workshop on Hot Topics in Cloud Computing*, June 2010, Article No. 5.

[5] Thusoo A, Sarma J S, Jain N, Shao Z, Chakka P, Anthony S, Liu H, Wyckoff P, Murthy R. Hive — A warehousing solution over a Map-Reduce framework. *Proceedings of the VLDB Endowment*, 2009, 2(2): 1626-1629.

[6] Kornacker M, Behm A, Bittorf V *et al.* Impala: A modern, open-source SQL engine for Hadoop. In *Proc. the 7th Biennial Conference on Innovative Data Systems Research*, January 2015, Article No. 5.

[7] Chang L, Wang Z W, Ma T *et al.* HAWQ: A massively parallel processing SQL engine in Hadoop. In *Proc. the 2014 ACM SIGMOD International Conference on Management of Data*, June 2014, pp.1223-1234.

[8] Costea A, Ionescu A, Raducanu B *et al.* VectorH: Taking SQL-on-Hadoop to the next level. In *Proc. the 2016 International Conference on Management of Data*, June 2016, pp.1105-1117.

[9] Hunt P, Konar M, Junqueira F P, Reed B. ZooKeeper: Wait-free coordination for Internet-scale systems. In *Proc. the 2010 USENIX Annual Technical Conference*, June 2010, Article No. 14.

[10] Chris L, Adve V. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proc. the 2nd IEEE/ACM International Symposium on Code Generation and Optimization*, March 2004, pp.75-88.

[11] Neumann T. Efficiently compiling efficient query plans for modern hardware. *Proceedings of the VLDB Endowment*, 2011, 4(9): 539-550.

[12] Neumann T, Leis V. Compiling database queries into machine code. *IEEE Data Eng. Bull.*, 2014, 37(1): 3-11.

[13] Shamgunov N. The MemSQL in-memory database system. In *Proc. the 2nd International Workshop on in Memory Data Management and Analytics*, September 2014, Article No. 1.

[14] Tan C K. Vitesse DB: 100% PostgreSQL, 100X faster for analytics. The 2nd South Bay PostgreSQL Meetup, 2015. https://www.meetup.com/postgresql-1/events/221039792/, Nov. 2019.

[15] Lu X, Liang F, Wang B, Zha L, Xu Z. DataMPI: Extending MPI to Hadoop-like big data computing. In *Proc. the 28th International Parallel and Distributed Processing Symposium*, May 2014, pp.829-838.

[16] Liang F, Lu X. Accelerating iterative big data computing through MPI. *Journal of Computer Science and Technology*, Mar. 2015, 30(2): 283-294.

[17] Gugnani S, Lu X, Qi H L, Zha L, Panda D K. Characterizing and accelerating indexing techniques on distributed ordered tables. In *Proc. the 2017 IEEE International Conference on Big Data*, December 2017, pp.173-182.

[18] Kemper A, Neumann T. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *Proc. the 27th International Conference on Data Engineering*, April 2011, pp.195-206.

[19] Pavlo A, Angulo G, Arulraj J *et al.* Self-driving database management systems. In *Proc. the 8th Biennial Conference on Innovative Data Systems Research*, January 2017, Article No. 14.

[20] Thusoo A, Sarma J S, Jain N, Shao Z, Chakka P, Zhang N, Antony S, Liu H, Murthy R. Hive — A petabyte scale data warehouse using Hadoop. In *Proc. the 26th IEEE International Conference on Data Engineering*, March 2010, pp.996-1005.

[21] Barber R, Garcia-Arellano C, Grosman R *et al.* Evolving databases for new-gen big data applications. In *Proc. the 8th Biennial Conference on Innovative Data Systems Research*, January 2017, Article No. 2.

[22] Kallman R, Kimura H, Natkins J *et al.* H-Store: A high-performance, distributed main memory transaction processing system. *Proceedings of the VLDB Endowment*, 2008, 1(2): 1496-1499.

[23] Pavlo A, Jones E P, Zdonik S. On predictive modeling for optimizing transaction execution in parallel OLTP systems. *Proceedings of the VLDB Endowment*, 2011, 5(2): 85-96.

[24] Serafini M, Mansour E, Aboulnaga A, Salem K, Rafiq T, Minhas U F. Accordion: Elastic scalability for database systems supporting distributed transactions. *Proceedings of the VLDB Endowment*, 2014, 7(12): 1035-1046.

[25] Taft R, Mansour E, Serafini M, Duggan J, Elmore A J, Aboulnaga A, Pavlo A, Stonebraker M. E-store: Fine-grained elastic partitioning for distributed transaction processing systems. *Proceedings of the VLDB Endowment*, 2014, 8(3): 245-256.

[26] Mahajan K, Chowdhury M, Akella A, Chawla S. Dynamic query re-planning using QOOP. In *Proc. the 13th USENIX Symposium on Operating Systems Design and Implementation*, October 2018, pp.253-267.

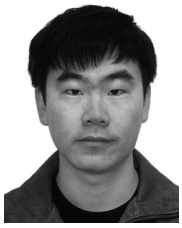
[27] Cowling J A, Liskov B. Granola: Low-overhead distributed transaction coordination. In *Proc. the 2012 USENIX Annual Technical Conference*, June 2012, pp.223-235.

[28] Färber F, May N, Lehner W *et al.* The SAP HANA database — An architecture overview. *IEEE Data Eng. Bull.*, 2012, 35(1): 28-33.

[29] Lee J, Kwon Y S, Färber F *et al.* SAP HANA distributed in-memory database system: Transaction, session, and metadata management. In *Proc. the 29th International Conference on Data Engineering*, April 2013, pp.1165-1173.

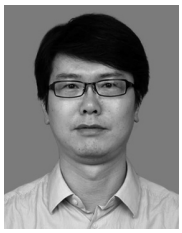
[30] Thomson A, Diamond T, Weng S C, Ren K, Shao P, Abadi D J. Calvin: Fast distributed transactions for partitioned database systems. In *Proc. the 2012 ACM SIGMOD International Conference on Management of Data*, May 2012, pp.1-12.

Zheng-Hao Jin is the chief architect of Business-Intelligence of Oriental Nations Corporation, Beijing. He received his Master's degree in computer software and theory from Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, in 2001. His current research interests include distributed DBMS, Big Data, cloud computing, and data science.



Haiyang Shi is a Ph.D. student in the Department of Computer Science and Engineering, The Ohio State University, Ohio, supervised by Dr. Xiaoyi Lu. Before joining OSU, he worked as a software engineer at Weibo and MiningLamp, Beijing. He received his Bachelor of Engineering degree in computer science and technology from Tianjin University, Tianjin, in 2012. His current research interests include Big Data, distributed file system, and high-performance erasure code.

Ying-Xin Hu is a software engineer of Business-Intelligence of Oriental Nations Corporation, Beijing. He received his Master's degree in optical engineering from Nankai University, Tianjin, in 2007. He spent several years working in digital signal processing and high performance computing. Currently, he is engaged in distributed relation database research and development, especially in HTAP architecture, execution engine, and high-performance computing.



Li Zha is an associate professor in Institute of Computing Technology, Chinese Academy of Sciences, Beijing. He received his Ph.D. degree in computer science and technology from Beijing Institute of Technology, Beijing, in 2003. His current research interests include distributed system, Big Data, cloud computing, and database. He is also a member of the Big Data Task Force of the China Computer Federation (CCF).

Xiaoyi Lu is a research assistant professor in the Department of Computer Science and Engineering, The Ohio State University, Ohio. He received his Ph.D. degree in computer science and technology from Institute of Computing Technology, Chinese Academy of Sciences, Beijing, in 2012. His current research interests include high-performance interconnects and protocols, big data analytics, parallel computing models, virtualization, cloud computing, and deep learning systems. He has published more than 100 papers in major international conferences, workshops, and journals with multiple Best (Student) Paper Awards or Nominations. He has been actively involved in various professional activities in academic journals and conferences. He is a member of ACM and IEEE. More details about Dr. Lu are available at http://web.cse.ohio-state.edu/~luxi.