Tsinghua Science and Technology

Volume 20 | Issue 1 Article 6

2015

# Banian: A Cross-Platform Interactive Query System for Structured Big Data

Tao Xu
*the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China.*

Dongsheng Wang
*the Department of Computer Science and Technology and Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China.*

Guodong Liu
*Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China.*

TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 07/11 pp62-71
Volume 20, Number 1, February 2015

# Banian: A Cross-Platform Interactive Query System for Structured Big Data

Tao Xu, Dongsheng Wang*, and Guodong Liu

**Abstract:** The rapid growth of structured data has presented new technological challenges in the research fields of big data and relational database. In this paper, we present an efficient system for managing and analyzing PB level structured data called Banian. Banian overcomes the storage structure limitation of relational database and effectively integrates interactive query with large-scale storage management. It provides a uniform query interface for cross-platform datasets and thus shows favorable compatibility and scalability. Banian's system architecture mainly includes three layers: (1) a storage layer using HDFS for the distributed storage of massive data; (2) a scheduling and execution layer employing the splitting and scheduling technology of parallel database; and (3) an application layer providing a cross-platform query interface and supporting standard SQL. We evaluate Banian using PB level Internet data and the TPC-H benchmark. The results show that when compared with Hive, Banian improves the query performance to a maximum of 30 times and achieves better scalability and concurrency.

**Key words:** big data; interactive query; relational database; HDFS; cross platform

## 1 Introduction

Big data is currently a research focus in both academia and industry. To analyze massive amounts of data and obtain valuable information and knowledge, researchers have developed many excellent systems and technologies[1–4]. The GFS[1] and MapReduce[2] developed by Google could process 20 PB of webpages per day in 2007. The HDFS[3] and HBase[4] clusters developed by Facebook[5] scanned 300 million images daily in 2012, amounting to more than 500 TB of data. The search engine system developed by Baidu[6] could handle 100 PB of data per day in 2013.

With the development and popularization of e-commerce and social network[7, 8], data have been showing increasingly high relevance and coupling degree, resulting in the rapid growth of the scale of structured data to PB level and above. Supporting interactive query on such large volumes of data necessitates the development of large-scale storage management ability and rapid analysis and calculation capability. These requirements pose new challenges for both relational database and big data processing technology.

For structured data, relational database is undoubtedly the most classic and popular database system, such as Oracle, MySQL, SQLServer, and DB2. In 1970, Codd[9] first proposed a new model of the relationship, which started the research on the relational method and theory of database. After decades of development, relational database has come to be widely used in various types of information management systems and business application systems[10], and has become an effective storage and analysis tool for data warehouse[11]. With changes

• Tao Xu is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: t-xu10@mails.tsinghua.edu.cn.
• Dongsheng Wang is with the Department of Computer Science and Technology and Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: wds@tsinghua.edu.cn.
• Guodong Liu is with Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: liuguodong@mail.tsinghua.edu.cn.
* To whom correspondence should be addressed.
Manuscript received: 2014-12-03; accepted: 2014-12-25

in information technology, the architecture of relational database has been improved continuously, from centralized database[12], distributed database[13] to parallel database[14], and the storage capacity has been increased from GB level to TB level. At present, parallel database based on Massively Parallel Processing (MPP)[15] architecture can manage hundreds of TB of data. Such database system consists of many loosely coupled processing units, and each unit has its own private computing and storage resources, such as CPU, cache, memory, hard disk, and operating system. The most significant features of MPP database are shared-nothing and multiple copies of data. Therefore, SQL commands can be split and scheduled to different processing units to be executed concurrently so as to achieve better performance. However, the system expenses of relational database for index construction and transaction mechanisms increase sharply when dealing with PB level data. Owing to the high cost and poor scalability, relational database appears weak for processing massive amounts of data and large-scale concurrent applications.

MapReduce is a programming framework proposed by Google and a typical technology for processing big data[2, 16]. With its super-large-scale node scheduling ability and high throughput, MapReduce performs excellently in processing the massive unstructured data[17]. Every computing request starts a job[18] in the MapReduce framework. In order to complete the job, the MapReduce framework needs to perform two kinds of tasks: map and reduce. First, it splits the input dataset into independent blocks and distributes them to different nodes. The job manager initializes several map tasks, and each map task processes one data block and generates an intermediate file after calculation. Then, the MapReduce framework sorts the output file of map tasks, and several reduce tasks are initialized that aggregate the sorting results into the final output file. The framework is responsible for scheduling and monitoring the tasks, and restarting failed tasks. Usually, these tasks are run on the same node as the distributed file system, i.e., the computing nodes and storage nodes are installed together. This allows the framework to effectively schedule tasks on nodes with data storage, which can ensure full use of the cluster network bandwidth. However, the MapReduce framework fails to provide sufficient support for interactive query on structured datasets. Furthermore, the initiation of map and reduce tasks consumes certain system resources and time. In addition, because the complex commands involved must be decomposed into multiple sub-operations and communication among the sub-operations is available only through intermediate files[19], the processing delay is intensified greatly.

Therefore, the study of structured big data is a crossing between the fields of big data and relational database[20]. By combining HDFS with the splitting and scheduling model, Banian effectively integrates large-scale storage management with interactive query and analysis.

Our contributions in this paper are as follows:

- We design a practical system that is efficient at managing and analyzing PB level structured data.
- We package the scheduler and query engine modules into middleware, which can work on other Hadoop-based systems with minimal changes to achieve the ability of interactive query.
- We implement a cross-platform query interface, through which clients can execute online join query directly between separate deployments of Banian or between Banian and any other relational database.

## 2 Related Work

The processing of structured big data warrants effective integration of massive storage with fast query and analysis. One line of research is incorporating MapReduce on the basis of MPP database, such as Greenplum[21] and Teradata[22]. These systems can be used to deal with SQL commands and MapReduce tasks simultaneously. SQL can directly use the output of MapReduce and serve query results as the input for MapReduce. However, owing to incompatibility between traditional storage architecture and the large-scale distributed system such as HDFS, as well as the complexity in the ETL (Extraction, Transformation, and Loading) of data, these systems exhibit low scalability.

Another major research approach is SQL on Hadoop, which provides an SQL interface on the HDFS and MapReduce foundation along with application oriented storage and query optimization[23–25]. Hive[26, 27] is the most typical example of SQL on Hadoop. It is used to map files onto a database table and provide an SQL query interface. Hive can run the data analysis logic reflected by SQL statements on HDFS by converting SQL statements into a series of MapReduce tasks. Logically speaking, Hive is only an interface, and thus, cannot improve the high latency of MapReduce.

Dremel[28] is an interactive data analysis system proposed by Google. It can execute ad-hoc queries on PB level datasets in seconds. To achieve such high performance, Dremel depends mainly on three aspects: using a new model that supports nested data; combining multi-level execution trees and the columnar storage format; and ability of managing large-scale clusters. As the report engine[29] of Google BigQuery, Dremel makes up for the disadvantages of MapReduce. However, Dremel is mainly meant for nested data optimization, and it does not perform very well in processing fact tables and dimension tables.

Spark[30] originated from the cluster computing platform at AMPLab, UC Berkeley. It supports users in performing in-memory computations on large-scale clusters using Resilient Distributed Datasets (RDDs). RDDs are capable of dealing with a variety of computing paradigms involving multi-iterative batch processing, data warehouse, flow processing, and graph computing. Spark improves system performance by storing data in the memory of these applications, and it can complete interactive query on a 1 TB dataset in 5–7 s. To achieve fault tolerance efficiently, Spark imposes restriction rules on shared memory using coarse-grained transformations. However, for applications of non-cyclic models, Spark cannot improve the performance, and it does not support fine-grained and asynchronous data processing.

Impala[31] is an MPP SQL query engine developed by Cloudera. It provides an interactive query interface directly on massive Hadoop data stored in HDFS or HBase. However, for join queries and complex queries involving the forwarding of intermediate results, Impala does not improve performance compared with Hive. The query processing will even fail if the size of the intermediate results goes beyond the memory capacity. Furthermore, Impala provides an interface for HiveQL (SQL-like) but does not support standard SQL.

BlinkDB[32] proposed by UC Berkeley is a large-scale parallel processing engine capable of running interactive SQL commands on PB level datasets. It allows users to improve the query response time by weighting the data accuracy, which is controlled to be within the allowable error range. To achieve this goal, BlinkDB uses two key ideas: an adaptive optimization framework and a dynamic sample selection strategy. The former is established with time prolongs using original data and is used for maintaining a set of multi-dimensional samples. The latter is used for selecting an example of an appropriate size based on the accuracy or response time requirement of a query. BlinkDB greatly shortens the search time via sampling. However, it is unsuitable for queries requiring high accuracy owing to the sample error.

In general, the scalability and compatibility of MPP database is not up to the requirements of large-scale data processing, while the performance of MapReduce is not adequate for responding to interactive query in a timely fashion. The study of structured big data needs to combine the advantages of the two approaches.

## 3 System Architecture

Figure 1 shows the architecture of Banian, which is divided into three main layers according to logic functions: the storage layer, scheduling and execution layer, and application layer. These layers are packed into middleware, which can work on other systems with minimal changes.

The storage layer is constructed using HDFS. It stores and manages PB level data with features such as high scalability, compatibility, and fault-tolerance. In the Banian architecture, the storage layer contains three important interfaces as well: (1) the interface used for providing the data block distribution information of the file to the scheduler module through NameNode; (2) read/write interface of local data to the query engine module; (3) the read/write interface of HDFS to the ETL module. Notably, we do not make any changes to the interfaces of the storage layer, but only use the standard API functions provided by HDFS. The other modules read/write data from/to HDFS actively by calling these API functions. Thus, Banian can run on newer versions of HFDS without requiring changes to its code. Furthermore, it maintains good compatibility with other HDFS-based systems.

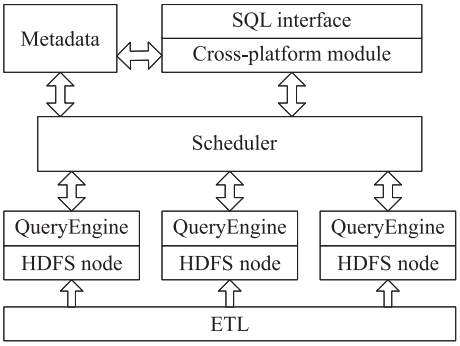


**Fig. 1 Banian architecture.**

The scheduling and execution layer is the core component of Banian. It contains three modules: scheduler, query engine, and metadata server. The scheduler receives SQL commands from the application layer. Then, by adopting the splitting and scheduling technology of parallel database, the commands are split and scheduled to the sub-nodes for concurrent execution. To ensure consistent scheduling of the commands and data, the database table information is firstly parsed into file information through the metadata server. Then, the file information is further analyzed into the position information of data blocks by HDFS NameNode. Finally, the scheduler generates an operation list for local execution at each sub-node. We implement the scheduler using a symmetrical structure. The application layer can send a user's query to any node. The scheduler daemon on this node becomes the worker node for this request, and it is responsible for command split, resource allocation, and result convergence. Section 4 will introduce the workflow and design principle of the scheduler.

The metadata server preserves metadata related to the system. The most important data is the database table structure. To ensure the system scalability, all related works are completed using the metadata server, while HDFS is excluded from the construction and maintenance of database tables. When a file is loaded into a database table, the corresponding relationship between the file and the table is recorded. The metadata server will inform the scheduler about the files needed to be queried and the method of parsing the file content through the table structure when splitting SQL commands. To speed up command distribution, the metadata server maintains a fast lookup table for caching data block information. According to the normal workflow, the scheduler needs to query twice, once each to the metadata sever and the HDFS NameNode, to obtain the file information and data block information. Using the fast lookup table, the scheduler can directly send the operation list to the query engine on the corresponding sub-node in the case of cache hit.

The query engine is deployed on each sub-node. It is responsible for receiving and executing the operation list allocated by the scheduler. Guaranteed by a consistent scheduling strategy, the query engine reads local data directly during execution. Such a design is conducive for optimizing concurrent tasks, reducing data transmission among sub-nodes, and alleviating network pressure. Given that only a local data queue needs to be maintained, the system cost is reduced greatly. During execution, intermediate results are stored in the memory. After completion of the operation list, the query engine sends the final results to the worker node. This is an important distinction with MapReduce. Furthermore, the query engine needs to maintain a regular heartbeat connection with the worker node. Once this connection fails, the worker node will restart the task of the failed node on another node that has a copy of the relevant data blocks.

In the era of big data, business applications request execution of join query on different platforms and even datasets of different regions. In terms of transverse compatibility and scalability, Banian provides a unified cross-platform query interface in the application layer. To achieve cross-platform join query, Banian allocates a data structure called Location for each platform, which is stored in the global table. Section 5 will introduce the workflow and design of the application layer.

In addition, Banian provides a distributed structure-oriented ETL interface. The ETL interface offers multi-dimensional strategy choices (including data formats, file organization structure, compression strategy, etc.) to support the dynamic balancing of upper applications in terms of ETL cost, storage efficiency, and analysis performance.

## 4 Splitting and Scheduling

To improve query performance, the scheduling and execution layer must split SQL commands into subtasks as much as possible and schedule the subtasks to different sub-nodes for concurrent execution.

Figure 2 shows the complete workflow of the scheduling and execution layer in processing SQL commands.

(1) Grammatical and lexical analysis is conducted by the execution and analysis units to generate the task tree after receiving SQL commands.

(2) Traverse each entry on the task tree, query metadata server according to table information, and obtain the corresponding file information.

(3) Transform tasks into file operations, i.e., task tree into operation tree. Query the fast lookup table, and go to Step 5 in the case of cache hit.

(4) Traverse each entry on the operation tree, query HDFS NameNode according to file information, and

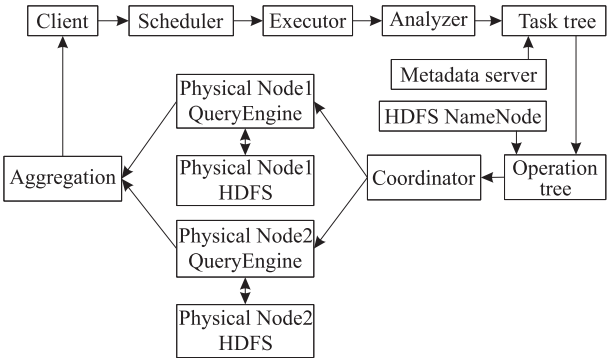


**Fig. 2 Workflow of scheduling and execution layer.**

obtain the corresponding data block position.

(5) According to the data block position, all entries in the operation tree at the same sub-node are integrated into an operation list. The coordinator unit sends the operation list to the query engine on the corresponding sub-node.

(6) The query engine initiates the workflow after receiving the operation list and directly reads local data for further execution (Steps 4 and 5 ensure that all data objects of each operation are available in local storage).

(7) After completing all commands in the operation list, query engine sends the results to the aggregation unit.

(8) The aggregation unit collects all results and sends them to the application layer.

The above workflow illustrates that the scheduler is the key path for the execution of SQL commands and the core module of Banian. As shown in Fig. 3, the scheduler is a logical unit as opposed to a physical module. It is composed of the scheduler daemons on each physical node. Being different from the metadata server and HDFS NameNode, the scheduler has no central node, and all physical nodes have equal status. Any scheduler daemon can receive an SQL command and become the worker node for task splitting and scheduling, and collecting results for said command. This architecture ensures system scalability and reliability. Because there is no central node, the cluster scale can be extended infinitely (the number of nodes is limited by the HDFS architecture actually), and the computing ability will maintain a linear growth rate with increasing cluster size. The model of multiple worker nodes improves query concurrency significantly. The evaluation results in Section 6 show that the processing procedure cannot be executed in parallel only for 2%–4% of the total query time. That is to say, Banian can be very effective (linear approximation) in reducing the query response time by increasing the number of nodes.

Failure detection[33] is important in a large-scale distributed system. During the running of Banian, the scheduler is responsible for monitoring the health and task completion condition of each node. In the case of a node going off-line owing to hardware failure, network error, software failure, or other causes, the scheduler will inform all nodes to ensure that subsequent queries avoid the failed node.

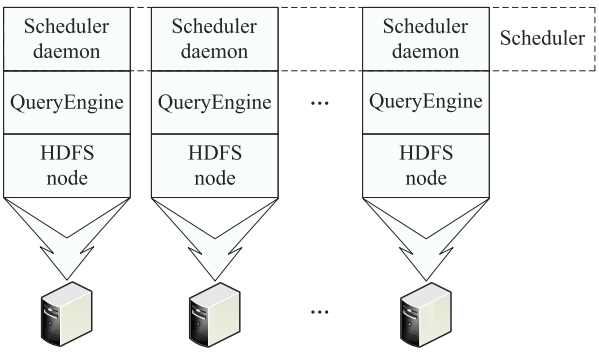


**Fig. 3 Scheduling deployment.**

## 5 Cross-Platform Query

Big data applications often need to access datasets on different platforms that may even be cross-domain. For structured data, the time cost of data extraction and loading cannot meet the real-time requirement. As shown in Fig. 4, the different platforms involved interconnect via LAN or Internet, resulting in a distributed and heterogeneous network topology. In this network topology, the data sources are dynamic, heterogeneous, and autonomous.

We implement a cross-platform query interface using which the clients can directly execute online join query between discrete deployments of Banian or between Banian and any other relational database

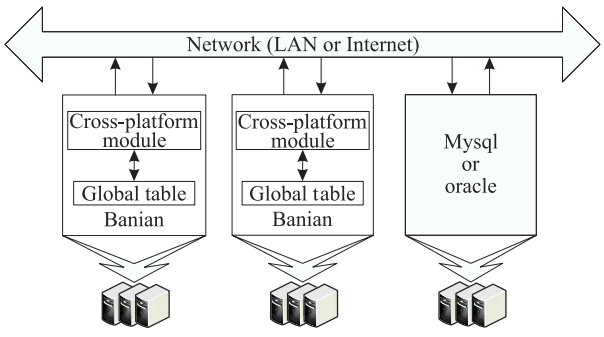


**Fig. 4 Cross-platform network topology.**

(such as MySQL and Oracle). The cross-platform query interface contains three main components: SQL interface, cross-platform module, and global table.

The SQL interface provides a command shell for users and forwards query commands to the cross-platform module. If a request command involves several datasets on different platforms, the cross-platform module queries the global table and gets the information of Location (a data structure). Then, it splits the command according to the variable tagname of Location, sends the sub-command to the slave platform as master, and receives the result.

The global table stores the configuration information of all platforms using a data structure called Location.

```
struct Location{
    char *tagname;
    char *host;
    int  port;
    int  authority;
    char *username;
    char *password;
}
```

Let us look at the cross-platform query workflow by taking the execution of a join query. We deploy two Banian systems called banian1 and banian2, with the following values of Location structure:

```
banian1_Location{
    tagname='banian1';
    host='166.111.134.49';
    port=2276;
    authority=1;
}
banian2_Location{
    tagname='banian2';
    host='166.111.134.50';
    port=2276;
    authority=1;
}
```

We create database db1 on banian1 and add table weblog, and create database db2 on banian2 and add table userinfo. Table 1 shows the specific SQL command. The cross-platform module splits the SQL command into sub-commands C1 and C2 (as shown in Table 1).

(1) Send command C1 to banian2_Location.host, wait for banian2 to return the result of C1.

(2) Store the result of C1 and execute command C2.

Note that clients should add the platform name as a prefix and underline the connection with the database table when sending SQL commands. Banian will parse the prefix string of each table and match it with the variable tagname of Location.

**Table 1 Cross-platform query command.**

| | |
|---|---|
| SQL command | Select * from banian1_db1.weblog join banian2_db2.userinfo on banian1_db1.weblog.sourceip = banian2_db2.userinfo.sourceip where banian1_db1.weblog.time >1401552000 and banian2_db2.userinfo.zipcode = 100084. |
| C1 | Select banian2_db2.userinfo.sourceip from banian2_db2.userinfo where banian2_db2.userinfo.zipcode = 100084. |
| C2 | Select * from banian1_db1.weblog where banian1_db1.weblog.sourceip = {result} and banian1_db1.weblog.time >1401552000. |

The variable authority records access authority of the platform. If access is barred, the variable is set to 0. If there are no application tools for accessing MySQL or Oracle, the username and password variables of Location should be assigned. Then, the cross-platform module can execute SQL commands remotely.

## 6 Evaluation

In this section, we evaluate the performance and scalability of Banian and compare the results with those of Hive. The workloads are derived from a cooperative institution (1.2 PB of Internet data) and the TPC-H benchmark[34, 35]. Firstly, on a cluster containing 100 nodes, we run 5 sets of SQL commands on a 1.2 PB dataset to evaluate the query latency of big data processing. Secondly, using a 1 TB dataset generated by TPC-H, we execute the 22 SQL commands of TPC-H to comparatively analyze the results of the queries with a high degree of complexity. Finally, by increasing the cluster scale sequentially between 10 and 100 in steps of 20, we test the lateral scalability of Banian and Hive.

### 6.1 Evaluation setting

We built an experimental platform with 100 servers (each with 8 CPU cores (2.66 GHz), 32 GB ram, and 16 TB hard disk) connected by gigabit Ethernet and installed Banian and Hive on the platform.

We prepared two workload sets: datasets D1 and D2. Dataset D1 (1.2 PB) comprises website access information collected from Internet and is reserved in form of log file. It is transformed into a large fact table having 60 billion rows in the column-store mode after being imported into the database through the ETL interface. The fact table consists of 240 columns, and it

records all website access information, including URL, SourceIP, TargetIP, time, type, version, browser, and operating system, etc. Dataset D2 is generated by the TPC-H benchmark and its size is 1 TB in total. It is composed of 8 base tables related to each other.

### 6.2 Performance evaluation

We load dataset D1 into Banian and Hive. Firstly, it is uniformly distributed across the 100 nodes by lines. Then, the data is transformed into the column-store format of parquet and rcfile on the local disk of each node. To analyze system performance comprehensively, we design 5 SQL commands (Q1–Q5), including single-column query, multi-column sequencing, group-by test, full-text retrieval, and double-table join query. Table 2 shows the specific SQL commands. During this evaluation, Q1–Q5 are executed 50 times each, and the average value is considered. Before each test, the memory is flushed. Figure 5a shows the query time.

As for the single-column query Q1, a simple integer value judgment is executed. On average, each sub-node processes 5 GB of data with the minimum query time (Banian 57.31 s and Hive 174.84 s). Q4 is a single-column query as well. However, because the column url occupies the most bytes and the like operation is the most time consuming, the query time is maximum (Banian 421.64 s). Analysis of the execution processes of Q1–Q5 on Banian suggests that the query time in the column-store mode depends largely on the number of columns and column bytes involved, as well as the query complexity.

Q5 on Hive shows the longest query time (1364.86 s). This is mainly because the double-table join query contains many comparison conditions and intermediate processes, which correspondingly elevates the system cost and IO operations of MapReduce.

**Table 2 SQL commands for performance evaluation.**

| | |
|---|---|
| Q1 | Select max(time) from weblog where time >1 401 552 000. |
| Q2 | Select count(*) from weblog where time >1 401 552 000 group by targetip. |
| Q3 | Select sourceip from weblog where time >1 401 552 000 and targetip = 166.111.4.100 order by time limit 1000. |
| Q4 | Select count(*) from weblog where url like '%tsinghua%'. |
| Q5 | Select weblog.url from weblog join userinfo on weblog.sourceip = userinfo.sourceip where weblog.time >1 401 552 000 and userinfo.zipcode = 100084. |

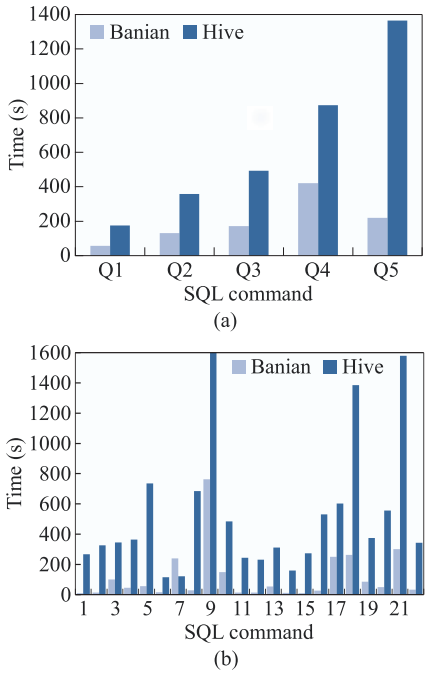


**Fig. 5 Query time of Q1–Q5 on Banian and Hive using 1.2 PB dataset D1 (a), and query time of 22 SQL commands of TPC-H benchmark on Banian and Hive using 1 TB dataset D2 (b).**

For PB level structured data-oriented queries, on average, Banian is 2–7 times faster than Hive, especially for complex queries involving multiple tables. The parallel splitting and scheduling technique is significantly advantageous compared with the MapReduce mechanism.

### 6.3 TPC-H evaluation

We load dataset D2 into Banian and Hive, and run a suite of business oriented ad-hoc queries (22 SQL commands) from the TPC-H benchmark on our experimental platform. The 1 TB of data including 8 tables ranging in size from 20 GB to 400 GB is evenly distributed on the 100 nodes. The results are shown in Fig. 5b.

The 22 SQL commands are far more complex than most OLTP transactions. For example, command SQL5 needs to query 6 tables and execute 8 statements, and command SQL20 involves 5 tables and 9 statements. Particularly, there are fixed running sequences between these determine statements.

Therefore, Hive using MapReduce will initialize multiple Map and Reduce tasks to handle these determine statements, and the query delay is intensified greatly because of the increasing IO of intermediate files. For SQL5 and SQL20, Banian improves query performance by up to 30 times compared with Hive.

In general, Banian shows excellent performance (over 5–30 times faster than Hive) when executing highly complex SQL commands.

### 6.4 Scalability evaluation

We split dataset D1 and each node retains 12 TB of data. The table size increases from 120 TB to 1.2 PB as the cluster size increases from 10 nodes to 100 nodes.

Figure 6 shows the query time of Q1–Q5 on Banian and Hive. We select two coordinates, 10 and 100, and calculate the increase in processing time resulting from cluster scale expansion. We use the variable $y$ to represent the increasing time, and the variable $y$ {Q1–Q5} has values of {1.14 s, 6.6 s, 8.46 s, 1.36 s, 23.21 s} and {38.71 s, 69.93 s, 90.15 s, 58.33 s, 332.08 s} for Banian and Hive, respectively.

A smaller value of the variable $y$ indicates better scalability. When the number of nodes increases from 10 to 100, the execution time of Q1–Q4 on Banian remains almost the same. As shown in Fig. 6, they are flat lines. However, the increase in the execution time of Hive is relatively obvious. Q5 is a double-table join query, and the $y$ of Banian is greater than those of other commands because of the forwarding of intermediate results. However, it is still far below the $y$ of Hive. The system management and communication overheads rise as the cluster size increases. The fine scalability of Banian reduces the range of this increase.

We decompose the execution processing of Q1–Q5 on Banian into three main steps: schedule, calculation, and result_converged. Figure 7 shows the time taken by each step. Calculation is the execution processing of all commands in the operation list on local data. If data size of each sub-node is fixed, so will be the time of calculation, regardless of the cluster scale. Schedule is the processing of splitting and scheduling, and result_converged is the processing of result aggregation. The variable $y$ is derived mainly

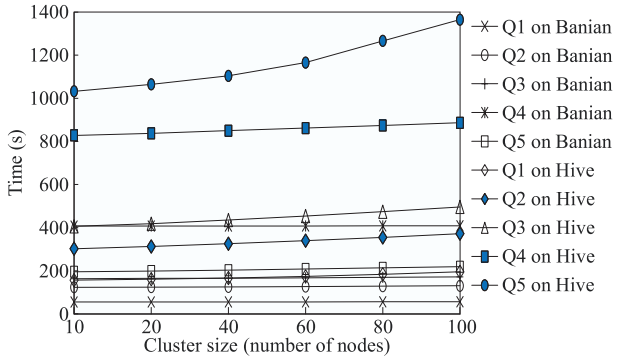


**Fig. 6 Query time of Q1–Q5 on Banian and Hive for cluster sizes of 10, 20, 40, 60, 80, and 100 in sequence.**

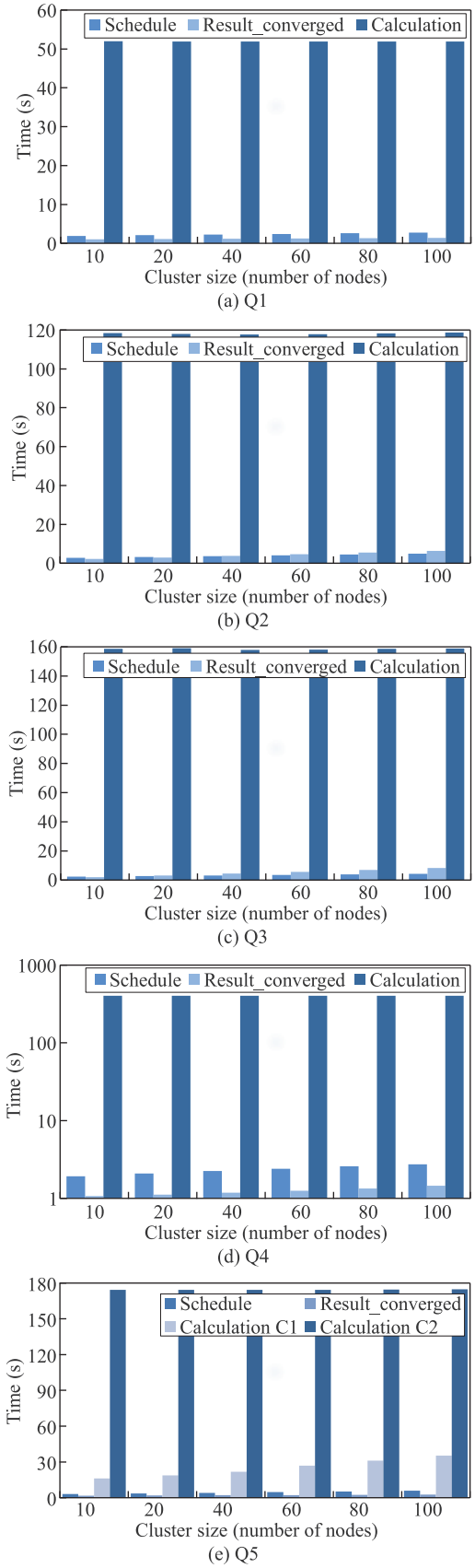


**Fig. 7 Time taken by schedule, calculation, and result_converged for Q1–Q5.**

from the steps of schedule and result_converged, which cannot be executed in parallel. As shown in Fig. 7, schedule and result_converged account only for 2%–4% of the total query time.

Q1 and Q4 are simply statistical queries, and the cost of schedule and result_converged is the least (3–4 s). Q2 and Q3 take 5–12 s because the results collected from all sub-nodes need to be recalculated. Q5 must query the table userinfo before calculation and send the result to all sub-nodes maintaining the weblog table. Therefore, it takes the longest time (21–44 s).

## 7 Conclusions

Drawing on the advantages of relational database and big data processing, we construct a structured big data-oriented management platform called Banian by combining HDFS with the splitting and scheduling engine of parallel database. This platform supports the storage of PB level data and interactive cross-platform query.

This paper systematically introduces Banian and describes the associated system architecture, as well as the design principle and working flow of each module. Banian is tested using two groups of datasets and the results are compared with those of Hive. The test results suggest that the performance of Banian is 5–30 times better than that of Hive with regard to complex SQL statements involving multiple tables or conditions. As for simple SQL statements, the performance of Banian is 3–10 times better than that of Hive. Banian employs a symmetrical structure having a loose coupling degree and shows higher scalability and compatibility. At the same time, owing to the adopted splitting and scheduling algorithm with consistent calculation and data, data transmission among sub-nodes is reduced, and the system concurrency is improved. Therefore, the overall system performance is enhanced significantly.

To achieve higher processing performance and scalability, Banian does not support the partial update and deletion of table data, and its support for transaction consistency is not very strong. Therefore, it is not yet a full-fledged replacement for parallel database. In the future, the above-mentioned weaknesses of Banian will be addressed with further research efforts.

### Acknowledgements

This work was supported by the National High-Tech Research and Development (863) Program of China (No. 2012AA012609).

### References

[1] S. Ghemawat, H. Gobioff, and S. T. Leung, The Google file system, *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 29–43, 2003.

[2] J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters, *Commun. of ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, The Hadoop distributed file system, in *Proceedings of IEEE Conference on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–10.

[4] HBase project, http://hbase.apache.org/, 2014.

[5] D. Borthakur, J. Grap, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, et al., Apache Hadoop goes realtime at facebook, in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2011, pp. 1071–1080.

[6] K. Yu, Large-scale deep learning at Baidu, in *Proceedings of the 22nd ACM International Conference on Conference on Information & Knowledge Management*, 2013, pp. 2211–2212.

[7] C. Budak, D. Agrawal, and A. El Abbadi, Structural trend analysis for online social networks, in *Proceedings of the VLDB Endowment*, vol. 4, no. 10, pp. 646–656, 2011.

[8] L. Pu, J. Xu, B. Yu and J. Zhang, Smart cafe: A mobile local computing system based on indoor virtual cloud, *China Communications*, vol. 11, no. 4, pp. 38–49, 2014.

[9] E. F. Codd, A relational model of data for large shared data banks, *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[10] L. Bellatreche and K. Y. Woameno, Dimension table driven approach to referential partition relational data warehouses, in *Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP*, New York, NY, USA, 2009, pp. 9–16.

[11] J. Han, J. Y. Chiang, S. Chee, J. Chen, Q. Chen, S. Cheng, W. Gong, M. Kamber, K. Koperski, G. Liu, et al., DBMiner: A system for data mining in relational databases and data warehouses, in *Proceedings of the 13th Biennial Conference of the Canadian Society on Computational Studies of Intelligence: Advances in Artificial Intelligence*, 1997, pp. 326–336.

[12] Y. C. Tay, N. Goodman, and R. Suri, Locking performance in centralized databases, *ACM Transactions on Database Systems (TODS)*, vol. 10, no. 4, pp. 415–462, 1985.

[13] D. Bell and J. Grimson, *Distributed Database Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1992.

[14] D. DeWitt and J. Gray, Parallel database systems: The future of high performance database systems, *Communications of the ACM*, vol. 35, no. 6, pp. 85–98, 1992.

[15] L. Antova, A. El-Helw, M. A. Soliman, Z. Gu, M. Petropoulos, and F. Waas, Optimizing queries over partitioned tables in MPP systems, in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 2014, pp. 373–384.

[16] Y. Chen, S. Alspaugh, D. Borthakur, and R. H. Katz, Energy efficiency for large-scale mapreduce workloads with significant interactive analysis, in *Proceedings of the 7th ACM European Conference on Computer Systems*, 2012, pp. 43–56.

[17] Y. Meng, Z. Luan, and D. Qian, Differentiating data collection for cloud environment monitoring, *China Communications*, vol. 11, no. 4, pp. 13–24, 2014.

[18] M. Zaharia, D. BorthaKur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, Job scheduling for multi-user mapreduce clusters, Technical Report UCB/EECS-2009-55, EECS Department, University of California, Berkeley, USA, April 2009.

[19] I. Elghandour and A. Aboulnaga, ReStore: Reusing results of MapReduce jobs in pig, in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012, pp. 701–704.

[20] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin, MapReduce and parallel DBMSs: Friends or foes? *Communications of the ACM*, vol. 53, no. 1, pp. 64–71, 2010.

[21] Greenplum Inc., Greenplum Database: Powering the data-driven enterprise, http://www.greenplum.com/resources, 2014.

[22] Y. Xu, P. Kostamaa, and L. Gao, Integrating hadoop and parallel DBMs, in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, 2010, pp. 969–974.

[23] F. N. Afrati and J. D. Ullman, Optimizing multiway joins in a map-reduce environment, *IEEE Transactions on Knowledge & Data Engineering*, vol. 23, no. 9, pp. 1282–1298, 2011.

[24] H. Herodotou and S. Babu, Profiling, what-if analysis, and cost-based optimization of MapReduce programs, in *PVLDB*, 2011, pp. 1111–1122.

[25] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin, HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads, in *PVLDB*, 2009, pp. 922–933.

[26] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, Hive — A petabyte scale data warehouse using Hadoop, in *IEEE 29th International Conference on Data Engineering (ICDE)*, 2010, pp. 996–1005.

[27] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, Hive: A warehousing solution over a map-reduce framework, in *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.

[28] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, Dremel: Interactive analysis of webscale datasets, in *Proceedings of the VLDB Endowment*, vol. 3, nos. 1–2, pp. 330–339, 2010.

[29] M. Li, L. Andrey, T. Sasu, and Y. Antti, MPTCP incast in data center networks, *China Communications*, vol. 11, no. 4, pp. 25–37, 2014.

[30] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2012.

[31] Impala project, http://impala.io/, 2014.

[32] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica, BlinkDB: Queries with bounded errors and bounded response times on very large data, in *Proceedings of the 8th ACM European Conference on Computer Systems*, New York, NY, USA, 2013, pp. 29–42.

[33] R. Lin, B. Wu, F. Yang, Y. Zhao, and J. Hou, An efficient adaptive failure detection mechanism for cloud platform based on volterra series, *China Communications*, vol. 11, no. 4, pp. 1–12, 2014.

[34] W. W. Hsu, A. J. Smith, and H. C. Young, I/O reference behavior of production database workloads and the TPC benchmarks — An analysis at the logical level, in *ACM Transactions on Database Systems (TODS)*, vol. 26, no. 1, pp. 96–143, New York, NY, USA, 2001.

[35] L.Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, et al., Bigdatabench: A big data benchmark suite from internet services, in *HPCA*, 2014.

**Tao Xu** is currently a PhD candidate in Department of Computer Science and Technology of Tsinghua University. He received his MEng degree from Tsinghua University in 2005. His research interests include massive storage, large-scale distributed system, and stream computing.

**Guodong Liu** is currently a research assistant in Tsinghua National Laboratory for Information Science and Technology of Tsinghua University. He received his BS degree from Shanghai Jiao Tong University in 2011. His research interests include large-scale data processing system, massive storage management, and distributed system.

**Dongsheng Wang** received his PhD degree from Harbin Institute of Technology in 1995. He is now a professor in Department of Computer Science and Technology at Tsinghua University. He is a senior member of China Computer Federation, member of IEEE. His research interests include computer architecture, high performance computing, storage and file systems, and network security.