Big Data Research 2 (2015) 102–109

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

Parallel Techniques for Large Data Analysis in the New Version of a Futures Trading Evaluation Service [☆]

Xiaoyun Zhou [a,*], Xiongpai Qin [b], Keqin Li [c]

[a] Computer Science Department, Jiangsu Normal University, Xuzhou, Jiangsu, 221116, China
[b] Information School, Renmin University of China, Beijing, 100872, China
[c] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

A R T I C L E I N F O

Article history:
Received 15 November 2014
Received in revised form 11 March 2015
Accepted 13 March 2015
Available online 22 April 2015

Keywords:
Large data analysis
Parallel processing
Futures trading evaluation

A B S T R A C T

A futures trading evaluation system is used to help investors analyze their trading history and find out the root cause of profit and loss, so that investors can learn from their past and make better decisions in the future. To analyze trading history of investors, the system processes a large volume of transaction data to calculate key performance indicators (KPI) as well as time series behavior patterns, and concludes some recommendations with the help of an expert knowledge base. This work is based on our early work of parallel techniques for large data analysis for futures trading evaluation service. In our early work, we have used the query rewriting technique to avoid joining between fact table and dimension table for OLAP aggregation queries, and used a data driven shared scanning of data method to compute KPIs for one customer. However, the query rewriting technique cannot eliminate joining for queries which aggregate on an intermediate level of the hierarchy of a dimensional table, so we propose a segmented bit encoding of dimensional table method which can eliminate the joining operation when the query aggregates on any level of the hierarchy of any dimensional table. Furthermore, our previous method perform badly when concurrency is high, so we propose an inter customer data scan sharing scheme to improve system performance in highly concurrent situations. We present our new experimental results.

1. Introduction

With the fast development of economy, more and more investors put money into futures markets with the expectation of making profit. A futures market has the characteristics of high-risk and high-return due to its high leverage ratio. A large number of non-professional investors participate in futures trading. Some of them make profits, and most of them lose money because of their different capabilities. For these participants, what they are most concerned about is to receive some recommendations from experts according to their past trading transactions. We design a futures trading evaluation system to help them. The system is a profit making capability assessment software, which is developed by the authors and GT Futures Brokerage Company.

The system processes historical transaction data of a specific investor, and extracts key performance indicators and trading behavior patterns. Through the analysis of these indicators and patterns, with the help of the knowledge of investment experts, the system gives customers some suggestions on trading capabilities, trading habits, and trading psychology, etc. to help customers identify their shortcoming and improve themselves in the future. In short, according to a customer's historical transaction data, the system generates an assessment report, with the hope that it can help the customer improve his (her) trading capabilities, stop loss, and make profit. The futures trading evaluation system tries to discover some valuable information, i.e., the customer's behavior characteristics buried in the process of trading, from large volume of historical transaction data.

In the evaluation system that we built, the volume of the data is greater than 200 GB, and it is growing at a pace of 300 to 500 MB per day. The data should be analyzed as timely as possible, so that customers can adjust their trading strategies according to the analytic results.

Our contributions in this paper include: (1) we propose a segmented bit encoding of dimensional tables for star schema data, which can eliminate join operation during aggregating on any level of the hierarchy of any dimension; (2) we exploit inter customer data scan sharing, which can improve report generation throughput greatly; (3) we have conducted a series of experiments to evaluate the proposed techniques.

[☆] This article belongs to BDA-HPC.
[*] Corresponding author.
E-mail addresses: wsp0516@163.com (X. Zhou), qxp1990@ruc.edu.cn (X. Qin), lik@newpaltz.edu (K. Li).

http://dx.doi.org/10.1016/j.bdr.2015.03.004
2214-5796/© 2015 Elsevier Inc. All rights reserved.

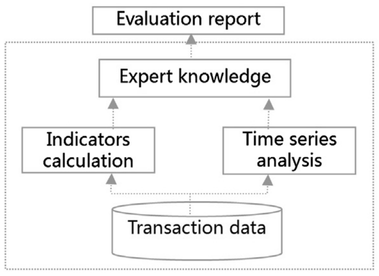

Fig. 1. The structure of the trading evaluation system.

2. Working logic of the futures trading evaluation system

GT Company has accumulated ten years' experience in futures brokerage. Its investment research department has gathered a group of futures market analysts. The core of the futures trading evaluation system is a knowledge base, which has solidified knowledge of these investment analysts.

The concept architecture of the system is shown in Fig. 1. The data for the evaluation system comes from production systems (they can also be downloaded from the China Margin Monitoring Centre Co., Ltd.). The data include daily status of *Holding* contracts, *Open* transactions, *Close* transactions, status of funds, and a number of ancillary information such as customers, varieties, and exchanges.

The indicator computing module calculates 87 key indicators including degree of risk, profit and loss, etc., from the data set of the reporting period. At the same time, the system analyzes 45 time-series patterns from the customer's trading transactions. These indicators and patterns are sent to the expert knowledge base for further processing. Finally an evaluation (assessment) report is generated, and it is sent to the users in the PDF file format.

We have adopted widely used indicators, but how to put indicators into groups and how to use them is the key of the futures trading evaluation system, and depends on experts' knowledge. Since the focus of this paper is to demonstrate some parallel techniques for large data analysis, and due to the requirements of confidentiality, we do not present further details of the knowledge based evaluation algorithm.

3. Parallel techniques for large data analysis

In this section, the scalable data processing architecture is firstly presented, followed by the data encoding scheme for star schema and corresponding query processing algorithms.

3.1. The scalable parallel data processing architecture

In the whole cluster architecture, different nodes are responsible for different jobs as follows.

3.1.1. Different nodes for different jobs

In order to support the increasing volume of data, we use cluster computing to do the job of data analysis. Cluster nodes are divided into three categories, including front-end processing nodes, data processing and analysis nodes, and data loading nodes (see Fig. 2).

Front-end processing nodes are responsible for pre-processing parameterized queries, and handing them over to the data processing and analysis nodes for further processing. We adopt a *Scatter-Gather* style of data processing method. When one of the front-end processing nodes receives an evaluation request, it decomposes the request into parts, and distributes the parts to data analysis and process nodes for real computing. Partially aggregated results are

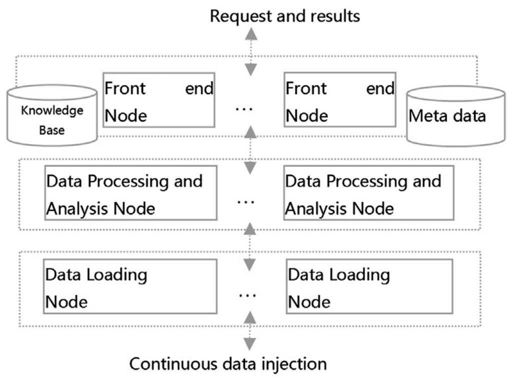

Fig. 2. The parallel data processing architecture.

Table 1
Nodes and functions.

Node	Functions
Front-End Node	Query Pre-processing
	Task Assignment
	Result Merging
	Meta Data Management
Data Processing and Analysis Node	Local Calculation of Indicators
	Local Calculation of Time Series Patterns
Data Loading Node	Data Extraction
	Data Transformation
	Data Splitting and Loading

merged in the front-end processing node later, and sent to the expert knowledge base to generate the evaluation report. The report is returned to the client using a URL, which can be used to download the generated PDF file.

Meta data about data distribution are stored in front-end nodes, which are used by the query dispatcher to select appropriate data processing and analysis nodes for parallel query processing. All front-end nodes could access a shared database engine which stores the meta data. To support highly reliability of the system, the hot standby technology can be used to protect the database.

Calculation of indicators can be expressed using SQL aggregation queries. Most SQL aggregation queries could be executed in a *Scatter-Gather* manner.

Each data processing and analysis node manages a subset of data, and it is responsible for local calculation of indicators and local time-series analysis, and returning partial results to the front-end processing nodes to merge. For example, the holding profit of a customer can be calculated on data nodes, and the partial results are merged on front end nodes to get the final result.

Since data analysis touches large volume of historical data, and the data are seldom updated (a data *append* is not an *update*, and new data are imported into the system by the data loading nodes), we decided to use non-transactional database storage engines to manage the data on processing and analysis nodes. MySQL is used as the underlying storage engine. By modifying the code base, the transaction management burden is avoided. Compared to database systems with fully transaction management support, the storage engine becomes much lighter, and data accessing speed is increased.

Data loading nodes are responsible for splitting, transforming, and loading new data. We adopt a data distribution scheme that is similar to GFS (Google File System) [1], where each data partition is replicated to at least three nodes for high fault-tolerance. Table 1 summarizes functions of the three types of nodes.

The system architecture is inspired by MapReduce (the Google's large-scale data processing platform). The difference is that we use database engines for data management, rather than a file sys-

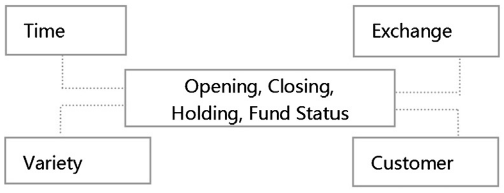

Fig. 3. Star schema of the data.

tem. The storage engine achieves higher structured data processing speed after transaction capabilities are removed.

The architecture makes the system very easy to scale. As the data volume grows, more data processing and analysis nodes are added to the system for acceptable performance. Currently, we have built an evaluation system based on a cluster consisting of 24 PCs, with 2 machines for data loading, 16 machines for data processing and analyzing, and 6 machines for front-end processing.

Data loading nodes use the above mentioned simple mapping scheme to load data into data processing and analysis nodes. When the number of nodes increases, the meta data about mapping has to be modified manually, so that data distribution could be correctly carried out. In future research, we will focus on methods that support automatic system scaling out without manual intervention, just like the consistent hash technique used in Amazon's Dynamo [2].

As for network topology, front-end processing nodes, data processing and analysis nodes, and data loading nodes are put in different network segments respectively. The three networks exchange data through an uplink core switch.

3.1.2. Data partitioning and data distribution

We adopt a timestamp-based partitioning method to partition the data among the data nodes. After the data are partitioned, they are distributed to adjacent nodes using the round-robin strategy. For example, when data are partitioned into P_1 and P_2, P_1 is stored on the nodes N_1, N_2, and N_3; P_2 is stored on the nodes N_2, N_3, and N_4. After data loading is completed, the meta information is registered into the database engine shared by the front-end nodes, so that newly loaded data could be used. The data partitioning method can avoid the situation that a customer's transaction data is accumulated on just one node. Thus the scheme can mobilize all data processing and analysis nodes to perform data computing in parallel, which can improve system performance.

We observe that the data volume of dimension tables including the customer dimension, the varieties dimension, the market (exchange) dimension, and the time dimension, is far smaller than that of the fact tables. In our system, the data volume of the fact tables is more than 200 GB, while the data volume of all dimension tables is less than 50 MB. So we store dimension tables on the database engine which is shared by front-end nodes.

3.2. Data encoding and query processing

3.2.1. The star schema

The data schema of the futures trading evaluation system is a typical star-schema. Fig. 3 shows the star schema used in our data processing system. The tables of time, exchange, variety, and customer are dimension tables, which store data about time information of *year–quarter–month–day*, different exchanges, different varieties, and customers respectively. There are several fact tables, including opening, closing, holding, and fund status, which store *Open* transactions, *Close* transactions, *Holding* contracts, and fund status of the customers every day.

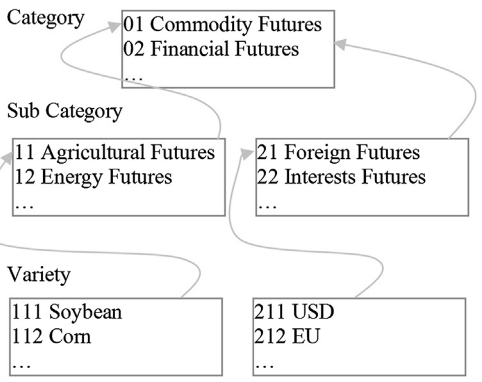

Fig. 4. The hierarchy of the variety dimensional table.

3.2.2. Query rewriting to eliminate the joining and the shortcomings

In our previous work [12], we have used the query rewriting technique to eliminate the join between the fact table and dimensional tables. The technique is illustrated by an example as follows.

The queries for indicator calculation and that for time-series analysis can be expressed in parameterized SQLs, which join the fact table and dimension tables.

For example, to calculate holding profit distribution over different varieties (different futures), the SQL query joins two tables, i.e., the *cust_hold* table and the *vari_info* table (the *cust_hold* table stores holding contract information, and the *vari_info* table stores varieties information). The SQL statement of the query (Q_1) is:

select vari_code, vari_name, sum(hold_profit)
from cust_hold h, vari_info v
where h.cust_no = ?
and h.vari_code = *v.vari_code*
group by v.vari_code, v. vari_name.

The SQL query groups some customer's holding profit by varieties.

Joining operations make the system performance declined sharply because of massive data exchange between data nodes when distributing data onto a large cluster. The parameterized SQL queries are rewritten by a query rewriting module, and the join operation is eliminated. After query rewriting, the above mentioned Q_1 is transformed into:

select vari_code, sum(hold_profit)
from cust_hold h
where h.cust_no = ?
group by h.vari_code.

The SQL query has eliminated the join operation between tables. However, there is a major shortcoming for the rewriting technique, i.e. it cannot deal with queries which aggregate on an intermediate level of the hierarchy of a dimension. In the new version of evaluation system, we are concerned about some more indicators, which are calculated by aggregating the data on an intermediate level of the hierarchy of the dimensional table. For example, for the *Variety* dimensional table, there is a concept hierarchy of three levels, i.e. "Category → Sub Category → Variety". Some sample data is provided in Fig. 4. Not only we are interested in profit distribution on different Varieties, but also we are concerned about the profit distribution on different sub categories.

In a relational data schema which conforms to BNF (Backus Naur Form) 3, the dimension of *Variety* is comprised of three tables, which are linked together by foreign keys. In the table of Category (the first level of the hierarchy of the *Variety* dimension), there are different categories such as "Commodity futures", "Financial futures" etc. In the table of Sub Category (the second level of the hierarchy of the *Variety* dimension), commodities such as

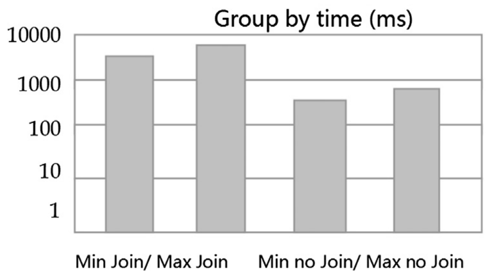

Fig. 5. GROUP BY times.

"Agricultural Product", "Energy Futures" etc belong to the category of "Commodity Futures". "Foreign Exchanges" and "Interest Rate" etc. belong to the category of "Financial Futures". In the table of Variety (the third level of the hierarchy of the *Variety* dimension), "Soybean" and "Corn" belong to the sub category of "Agricultural Product". And "USD" and "EU" etc. belong to the sub category of "Foreign Exchanges". (*Please notice the difference of Variety dimension and the Variety table*.)

If we aggregate some measures on the second level of the hierarchy of the *Variety* dimension, the SQL would be:

select s.sub_cat_name, sum(hold_profit)
from cust_hold h, vari_info v, sub_cat s
where h.cust_no = ?
and h.vari_code = v.vari_code
and v.sub_code = s.sub_code
group by s.sub_cat_name.

For such a SQL query, which joining three tables, and aggregate on the intermediate level of the hierarchy of a dimensional table, query rewriting technique cannot transform it into a query that completely eliminate the joining.

Joining elimination has demonstrated superior performance in our previous work. Through the elimination of join operation, the performance of *GOUPY BY* operation is improved greatly. The time is reduced from about 3–5 seconds to about 380–440 mill-seconds, as shown in Fig. 5, where the y-axis uses a logarithmic scale.

Min Join and *Max Join* denote the minimum and maximum time to perform a Group-By using joining. *Min no Join* and *Max no Join* denote the minimum and maximum time to perform a Group-By without joining.

We propose an encoding method to encode hierarchical information of dimensional tables into the fact table to speedup queries which aggregate on any level of the hierarchy, by eliminating joining.

3.2.3. Segmented bit encoding of dimensional tables and query processing

We use the dimension of Variety to illustrate how to encode the hierarchical information of a dimensional table, and how to rewrite the query to leverage the encoded information.

In the *holding* fact table, the information in each row is as follows.

Variety key	Time key	Customer key	Exchange key	Holding Information

The "variety key" is a foreign key, which is linked to the Variety table in the *Variety* dimension. We try to encoding three levels of hierarchy into a bit string to replace the "Variety key" in the fact table.

Firstly, according the cardinality of the Category table, i.e. there are at most C categories in our system, then we can encode each category into $\lceil \log_2^C \rceil$ bits (Cat_{bits}). Secondly, according the max number S of the sub categories under each category (we do not use the cardinality of the whole Sub Category table to reduce space consumption), we encode the sub categories under each category into bits in the same manner (Sub_{bits}). Thirdly, according to the max number V of the varieties under each sub category, we encode each variety into bits (Var_{bits}).

Cat_{bits}, Sub_{bits} and Var_{bits} are concatenated together as a new variety_key to replace the foreign key in the fact table of cust_hold. Suppose Cat_{bits} is 4 bits long, Sub_{bits} is 4 bits long and Var_{bits} is 8 bit long. The soybean under agricultural product (which is under commodity futures) can be encoded into "0001.0001.00000001", the points inside the bit string are there for reading convenience only.

Hierarchical information in other dimensional tables are also encoded and used to replace their foreign keys in the fact tables respectively.

SQL queries which aggregate on intermediate levels of the hierarchy of any dimension can be easily rewritten to eliminate the join. For example, the SQL in the end of Section 3.2.2 can be rewritten as follows:

SQL	*select s.sub_cat_name, sum(hold_profit)* *from cust_hold h, vari_info v, sub_cat s* *where h.cust_no = ?* *and h.vari_code = v.vari_code* *and v.sub_code = s.sub_code* *group by s.sub_cat_ name*
rewritten	*select s.varity_key, sum(hold_profit)* *from cust_hold h* *where h.cust_no = ?* *group by variety_key & "0000.1111.00000000"*

The trick is that by *grouping by* variety_key & "0000.1111.00000000", we not only correctly aggregate on the second level of the hierarchy of the *Variety* dimensional table, but also eliminate the need to join between the fact table and dimensional tables.

Not only group by can be transformed into bit string operations just like the example above, but also various predicate in the where clause of a SQL can be transformed to operate on bit strings.

For example, equation predicate such as "*Variety.vari_name = 'soybean'*" can be transformed into "*cust_hold.variety_key =* '0001.0001.00000001'". Queries containing equation predicates, range predicates, LIKE predicate, as well as IN predicate are transformed into a query which operates only on the fact table, specifically on the replaced foreign keys, which have encoded hierarchical information of the dimensions.

Worthy to mention is that, when data is loaded into the data warehouse, the data should be encoded. This is a practical strategy in that data warehouse follows the usage pattern of "*write once, query many times*", thus it is worthy of paying some cost while loading data, the overhead will finally pay off during data warehouse using.

3.2.4. Intra customer data scan sharing and low concurrency

In the new version of evaluation service, we still use the iterative algorithm in our previous work [12] to calculate the indicators. For integrity of this paper, we briefly present the idea here.

Since we have also using data scan sharing between customers (generating evaluation reports for different customers), the data scan sharing for indicator calculation for one customer is called intra customer data scan sharing here.

The calculation process of indicators is as follows. Firstly, each indicator is initialized. Secondly, each indicator is pre-processed;

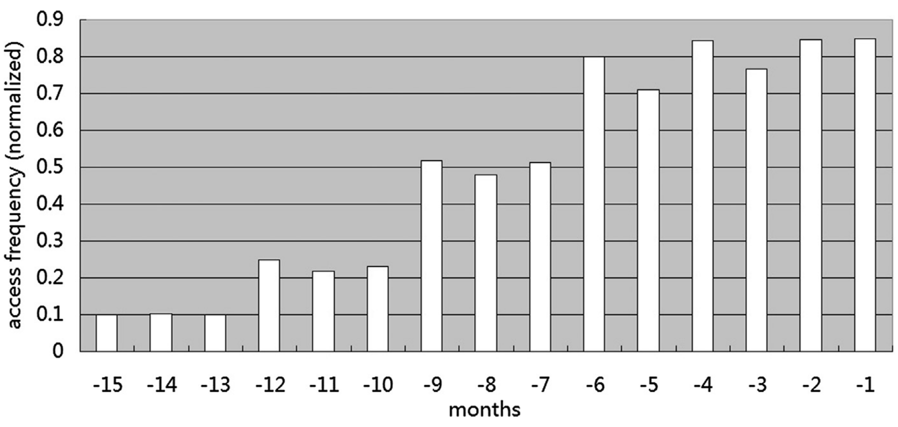

Fig. 6. Access frequencies of data across different months.

when the cursor produces one tuple (one row of data), each indicator uses the tuple to calculate its own internal value. Lastly, all indicators are post-processed and the calculation is complete.

The calculation of holding profit distribution over different exchanges is used as an example to illustrate the iterative algorithm.

During the initialization stage, a hash map to store holding profit of different exchanges is created. At the stage of preprocessing, nothing is done. When consuming one tuple from the SQL cursor of holding contract table, the *exchange_no* is used to lookup the hash map. If the holding profit of some *exchange_no* could be found, then the profit in current tuple is added to the value in the hash map; if not, then a new hash map entry having the form of $< exchange\_no, holding\_profit >$ is created and inserted into the hash map. During the post-processing stage, nothing is done to the hash map. When the front-end node merges partial results from data processing and analysis nodes, it looks up in the *Exchanges* table to replace the *exchange_no* with *exchange_name*, and produces the final holding profit distribution over exchanges.

Intra customer data scan sharing can compute indicators in one pass of data scanning. However, in a highly concurrent situation where assessment reports of different customers need to be generated, the throughput of the system is low because there is no coordination between data accessing between customers (please see our later experiment results).

We have analyzed the access log of the evaluation system, and found that most customers are interested in evaluation of trading in the recent year, especially in most recent 6 months. The access frequencies of data across different months is depicted in Fig. 6 (the data has been normalized).

We design an inter customer data scan sharing scheme with the expectation that system throughput could be improved in highly concurrent situations.

3.2.5. Inter customer data scan sharing for higher concurrency

When many users access the data warehouse concurrently, there are some opportunities that we can exploit to improve query performance. When these queries need some common data blocks, disk I/Os should be shared between them.

3.2.5.1. Clockwise shared scanning For high performance of concurrent access, we adopt a data driven query execution strategy which is based on clock wise shared scanning of data blocks.

The data blocks on each node are organized into a virtual circle. The scanning thread will prepare the data, block by block in a circular manner. Fig. 7 shows a high level idea of the algorithm.

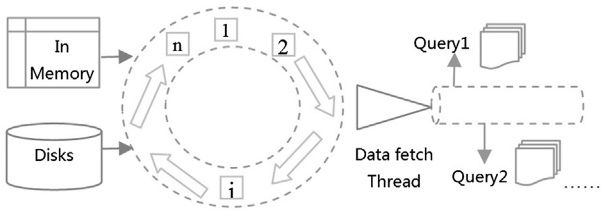

Fig. 7. Clock wise shared scanning.

After a block of data is fetched, it is fed into the virtual pipeline. The pipeline contains some data blocks, and the data blocks will be consumed by the queries. Each query maintains a private in memory table (hash) to store local aggregation results.

When some new query comes, the data fetch thread may be currently preparing data block of number i, the number of the data block is recorded in the query's in memory data structure. When the data fetch thread reach the data block number i again, the query has seen all data blocks, it can finish.

Queries are run in their threads, but scheduled by a scheduler. The scheduler coordinates the whole thing of invoking data fetch thread, and executing every waiting query. The algorithm of the scheduler is as follows.

```
Algorithm 1: Scheduling of Queries
01, $i = 0$;
02, wait for ready of current data block $i$;
03, Check to see whether there are some new queries, if so put the query into
the running queue and set its start block number to $i$;
04, put the data block into the virtual pipeline;
05, for (int $j = 0$; $j <$ running queue. length; $j++$)
06, {
07,         tell the query of number $j$ to run;
08,         feed the data block into the query;
09, };// queries now are running concurrently
10, for (int $j = 0$; $j <$ running queue. length; $j++$)
11, {
12,         wait query $j$ to finish work on the data block;
13, };
14, $i++$;
15, $i = i$ % number of data blocks;
16, check to see whether there are some queries in the running queue that
could finish, if so finish them;
17, goto step 2;
```

Note: the current data block will be prepared by the data fetch thread.

When there is enough memory, some memory will be used as data buffer to accommodate data blocks. When the requested data block is resident in the buffer, it can be accessed with low latency; otherwise the data block should be read from disks.

Usually executing queries in memory is much faster than retrieving records from disks. To accelerate disk I/O operations, it is preferable to fetch data from disk by sequential reading. A thread called data fetch thread is uniquely in charge of preparing next data block. If the block is not in memory, it will read the block from the disk.

The data fetch thread works in a looking forward manner, when the current data block is processed by all queries, it is told to follow up with a new data block. When the queries are running, the data fetch thread is not in an idle state, it will go ahead to see whether the data blocks that will be accessed next are ready. If the data block is in the memory, it is OK. If not, the data blocks should be read from disks and placed in the buffer. Some blocks in the buffer will be evicted.

There are two parameters that tune the running behavior of the data fetch thread, one is the number of data blocks that should be checked in each round of preparing – $Num_{\text{look ahead}}$, another is the threshold of access frequency that a data block should be evicted from buffer for new coming data – $AF\_Threshold_{\text{evict}}$.

We vary the number of $Num_{\text{look ahead}}$ and $AF\_Threshold_{\text{evict}}$ to choose the best ones for higher performance. The data fetch thread prepares data blocks, and fetches data blocks from disks when necessary. The algorithm of the data fetch thread is as follows.

```
Algorithm 2: Data Fetch Thread
01, put the current data block in the virtual pipeline, and tell the ready status of the block to the scheduling thread;
02, look ahead to see whether $Num_{\text{look ahead}}$ of data blocks are ready in the memory buffer;
03, if some of the data blocks are not ready, select some in memory data blocks for eviction;
04, read the data blocks from disks into memory buffer;
05, wait for the scheduling thread to wake itself up;
06, goto step 1;
```

3.2.5.2. Combing inter customer and intra customer data scan sharing To combine inter customer and intra customer data scan sharing, firstly data blocks are fetched from disks and put in memory. When memory buffer is used up, new data request will evict some data blocks in memory.

The current data block is fed into indicator calculation threads for different customer (one for each customer). In each indicator calculation thread, data is filtered according to *cust_id*, and fed into the iterative indicator calculation algorithm.

The data fetching thread fetches and feeds data in a continuous manner, each indicator calculation thread should take care of ending the calculation when all data in the report period is accessed (please refer back to Section 3.2.5.1).

3.2.5.3. Sliding window access frequency based data block memory eviction Which data blocks should be resident in the memory buffer depends on their access frequency. To deal with hotspot shifting, i.e. with the time elapsing, some new hot data will emerge and some old one will cold down. The access frequency should be computed by using a sliding time window.

The access frequency is not a single number, but a series of access frequencies in recent consecutive time slots. The data structure to store the access frequencies includes the *start* timestamp of the first time slot, and the access frequencies array. There is no need to store the *start* timestamp of each time slot, based on the size of a timeslot, these timestamps can be calculated. The size of the access frequencies array $Size_{\text{af-array}}$ and the time span of a single time slot $TimeSpan_{\text{slot}}$ are tunable.

The optimal values for the two parameters are application dependent. In our experiments with SSB (Star Schema Benchmark) data set and workloads, the two parameters are set to 8 and 30 minutes respectively.

Access frequencies of all data block are maintained in a global data structure. Fig. 8 depicts a 3 slot access frequency structure.

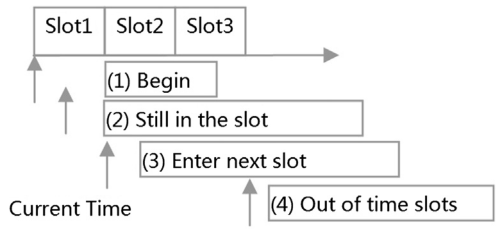

Fig. 8. Sliding window access frequencies of a data block.

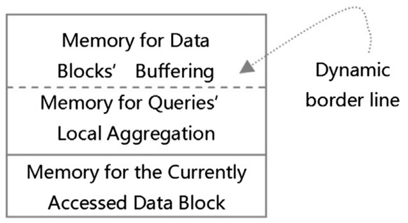
Memory for Data Blocks' Buffering
Memory for Queries' Local Aggregation
Memory for the Currently Accessed Data Block
Dynamic border line

Fig. 9. Memory allocation for different use.

Upon startup of a data node, the access frequency structure for each data block is initialized. With the time going on, if current timestamp is still in the current slot, then every access of the data block will be added to the slot.

If the time is beyond the current slot and enters the next slot, the current slot will be changed, later access of the data block will be added to the new current slot. When current time is beyond the scope that could be recorded in the three slots, i.e. Current time stamp − start time stamp $> 3 * TimeSpan_{\text{slot}}$, the frequency of the slot 1 will be thrown away, slot 2 and slot 3 will replace slot 1 and slot 2 respectively (using a memory copy operation), and a new slot 3 is created. The start timestamp of the first slot is changed accordingly. The total access frequency of a data block is calculated by summing up the frequencies store in the slots.

Memory on each node is limited. It is partitioned into three major parts, the memory for currently accessed data blocks, the memory for data blocks' buffering, and the memory for queries' private aggregation (Fig. 9).

Since the number of running queries is changing, the memory need of these queries is also changing. There is a need to evict some data blocks from the buffer to provide the memory for queries' private use. The eviction algorithm, which is a variant of LRU algorithm, is simply based on access frequency.

4. Experimental results

4.1. Generation times of evaluation reports

We carried out experiments on a high-end server and a cluster respectively. The high-end server is an HP Proliant DL580 G enterprise server, and it is equipped with two Intel Xeon E7330, 8 GB of RAM, and 500 GB SAS hard disk. The cluster consists of 24 Dell755 PC entry level servers. Each server is equipped with an Intel Core 2 Duo E6550, 2 GB RAM, and a 250 GB SATA hard disk.

Note: ***Cluster*** denotes our previous work of intra customer data scan sharing; ***Cluster**** denotes our current work with inter customer data scan sharing added.

The report generation time on the HP ProLiant server and the cluster are shown in Fig. 10 (the selected reporting period is from December, 2008 to December, 2009). The data volumes of the customers (C1 to C8) are different due to their different trading habits, so their report generation time are different too.

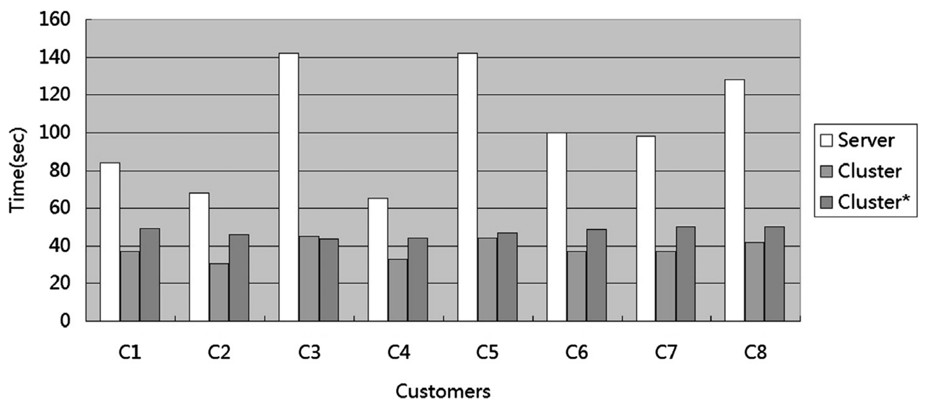

Fig. 10. Report generation time on a high-end server and a cluster.

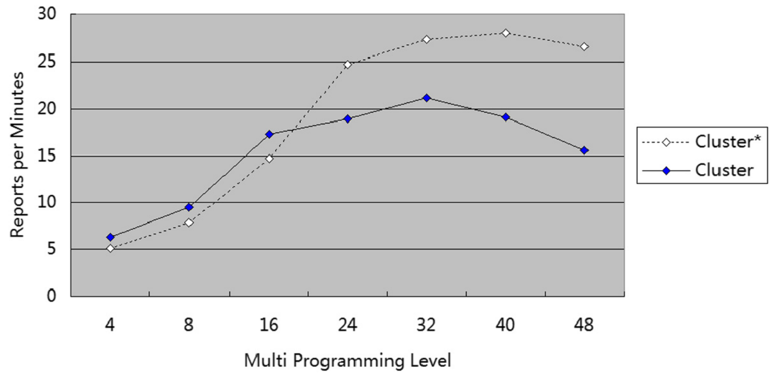

Fig. 11. Throughputs of *Cluster* and *Cluster**.

Using the high-end server, the report generation times are between 1 minute and 2.36 minutes; while using the cluster, the times are from 30 seconds to 50 seconds.

We can see that report generation times of *Cluster** are worse than *Cluster* for most of the customers, but by a small margin. This can be explained by the reason that when sharing data scan among customers, some data loaded into memory is not relevant to queries of a specific user. The loading of this data is completely a waste. In our following up work, we will try to reduce such waste by using some index techniques to filter out irrelevant data in blocks.

Although we achieve worse response times, however we achieve higher system throughputs, please refer to next section.

4.2. System throughputs

We also measure throughputs of *Cluster* and *Cluster** by vary the MPL (multi programming level) parameter of test client.

The result is as depicted in Fig. 11. When increasing MPL, both throughputs of *Cluster* and *Cluster** increase.

Throughput of *Cluster* reaches its peak when MPL is 32, and after that it drops down quickly. However throughput of *Cluster** reach its peak when MPL is 40 and after that it slowly drops down.

At low concurrencies, *Cluster* outperforms *Cluster**, however *Cluster** outperforms Cluster at higher concurrencies by exploiting inter customer data scan sharing.

5. Related works and discussion

Segmented bit encoding of dimensional information has borrowed ideas from universal relation [3]. However, our scheme doesn't put all dimension information but hierarchy information into the fact table, thus it is more space-saving compared with universal relation. Then the hierarchical information is used by most aggregation queries in our applications.

IBM has proposed BLINK [4] prototype to pre join dimension tables and the fact table to form a single wide table, which results in much simpler query processing. Table scanning is parallelized and constant query response time is achieved. De-normalization of data leads to data redundancy. Our scheme does not incur as much data redundancy as BLINK.

In the domain of scientific research, simulation, internet, e-commerce, as well as the financial data analysis areas discussed in the paper, it is witnessed that the data volume is growing rapidly [5]. Traditional data warehouse technology could not deal with the rapid exploding data effectively.

Google has brought forward the MapReduce technology, which is a parallel computing software framework [6] to deal with very large data sets. In Google, more than 20 PB of data is processed every day using MapReduce. MapReduce has demonstrated its power in the area of big data processing [7].

MapReduce has given us great inspiration, and our scheme fit the MapReduce framework well. In our following up work, we are planning to embed the data encoding scheme and data scan sharing scheme to the Hadoop [8] system (the open source MapReduce implementation). The HadoopDB proposed in [9,10] is used as a reference.

Data scan sharing can improve system throughput greatly. In [11], the authors proposed merging the execution plans of queries for sharing the underlying data scan. Our scheme does not merge query plans, but runs queries separately and consumes common data providing. Although it is much simpler, it is efficient just as demonstrated from experimental results.

6. Conclusion

In this paper, based on our previous work of a futures trading evaluation service, we have proposed segmented bit encoding of dimensional tables and exploited inter customer data scan sharing.

By using the segmented bit encoding of dimensional hierarchical information, we bring the advantage of eliminating joining when aggregating on any level of the hierarchy of any dimension. By sharing data scan among customers, report generation throughput is improved.

Acknowledgements

This work is supported partially by NSF of China under Grant No. 61170013 and NSF of Jiangsu Province under Grant No. BK2012578.

References

[1] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung, The Google file system, in: SOSP 2003, 2003, pp. 29–43.

[2] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, Werner Vogels, Dynamo: Amazon's highly available key-value store, Oper. Syst. Rev. 41 (6) (2007) 205–220.

[3] Tzy-Hey Chang, Edward Sciore, A universal relation data model with semantic abstractions, IEEE Trans. Knowl. Data Eng. 4 (1) (1992) 23–33.

[4] Vijayshankar Raman, Garret Swart, Lin Qiao, Frederick Reiss, Vijay Dialani, Donald Kossmann, Inderpal Narang, Richard Sidle, Constant-time query processing, in: ICDE 2008, 2008, pp. 60–69.

[5] Lisa Kart, Nick Heudecker, Frank Buytendijk, Survey analysis: big data adoption in 2013 shows substance behind the hype, http://www.gartner.com/id=2589121, 2013.

[6] Jeffrey Dean, Sanjay Ghemawat, MapReduce: simplified data processing on large clusters, in: OSDI 2004, 2004, pp. 10–17.

[7] S. Sakr, A. Liu, A.G. Fayoumi, The family of MapReduce and large scale data processing systems, http://arxiv.org/abs/1302.2966, 2013.

[8] Apache Foundation, Hadoop Project, http://hadoop.apache.org/.

[9] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel Abadi, Avi Silberschatz, Alexander Rasin, HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads, in: PVLDB 2009, vol. 2(1), 2009, pp. 922–933.

[10] Daniel Abadi, From HadoopDB to Hadapt: a case study of transitioning a VLDB paper into real world deployments, VLDB Business & Awards, 2013.

[11] Georgios Giannikis, Gustavo Alonso, Donald Kossmann, SharedDB: killing one thousand queries with one stone, in: VLDB 2012, vol. 5(6), 2012, pp. 526–537.

[12] Xiongpai Qin, Huijui Wang, Xiaoyong Du, Shan Wang, Parallel techniques for large data analysis in a futures trading evaluation service system, in: GCC 2010, 2010, pp. 179–184.