Real-Time Collaborative Planning with Big Data: Technical Challenges and In-Place Computing (Invited Paper)

Wenwey Hseush
Yi-Cheng Huang
Shih-Chang Hsu
eBizprise Inc.
{wenwey, ychuang}@ebizprise.com

Calton Pu
College of Computing
Georgia Institute of Technology
calton.pu@cc.gatech.edu

***Abstract*—There is increasing collaboration in new generation supply chain planning applications, where participants across a supply chain analyze and plan on a big volume of sales data over the internet together. To achieve real-time collaborative planning over big data, we have developed an unconventional technology, *BigObject*, based on an *in-place computing* approach in two ways. First, instead of moving (big) data around, move (small) code to where data resides for execution. Second, organize the complexity by determining the basic functional units (objects) for computing in the same sense that macromolecules are determined for living cells. The term "in-place" indicates that data is in residence in memory space and ready for computing. BigObject is an in-place computing system, designed for storing and computing multi-dimensional data. Our experiment shows that in-place computing approach outperforms traditional computing approach in two orders of magnitude.**

***Index Terms*—In-Place Computing, big data, database, real-time system, transformation programming, in-memory computing**

I. Introduction

Over the past decade with the growth of the internet and mobile technologies, the clock speeds of businesses have been accelerating gradually. To remain competitive, businesses and their supply chains need to be responsive and adaptive to the markets. Technologies enable the possibility for partners around the world to collaborate together and achieve common business goals in a real-time manner. Such real-time collaboration is critical when business becomes competitive.

Our work was motivated by a real-time collaborative planning problem, where a group of participants analyze and plan sales over internet in collective efforts. Such corporate-wide or supply-chain-wide collaboration is extremely complex and difficult while both the number of participants and the size of data to be analyzed continually grow. Unfortunately, traditional data technologies such as relational database management systems are inadequate for analyzing such big data, even though they are proven to be efficient to handle transactional applications. On the other end, most commonly adopted approaches to big data computing such as MapReduce [1] in a distributed fashion were originally designed to process a big volume of data, rather than addressing real-time performance issues, especially for applications with data components that present high degree of interdependency.

The main properties that characterize big data are often described as multiple V's [2], with Volume and Velocity as the main properties. In this paper, we add another V, *Valence*, to describe the degree of interdependency among data components of big data applications that are not embarrassingly parallel. The valence of a dataset determines how difficult it is to decompose the dataset into subsets and thus how to divide the problem into sub-problems, which helps to decide an appropriate approach to big data computing and storage. The applications with high-valence datasets include social graph [3] analysis and the collaborative planning problem we are addressing in this paper. The collaborative planning problem presents high valence due to its interactive nature as well as collaboration with multiple views on the shared sales data, where any action or reaction will affect the views of others.

In order to achieve real-time collaborative planning over big data with high valence, we have developed an unconventional technology, *BigObject*, based on an *in-place computing* approach in two ways. First, it moves away from the traditional computing model that requires explicit data retrieval to a data-centric computing model [4], [5] in which computations take place where data resides. The idea is, instead of moving (big) data around, to move (small) code to where the data resides for execution. Second, it organizes the complexity of big data by determining the basic functional units (objects) in a similar way that macromolecules are determined for human living cells. The idea is to express and manipulate big data in a high-level (or macro-level) and efficient way. In the rest of this paper, for simplicity we use the term "big data" to mean "big data with high valence" and use "traditional computing model" to refer to "the computing model that requires data retrieval from relational databases".

The term "in-place" indicates that the data is in residence in memory space and ready for computing. Both code and data reside in the same space, well organized and standing by. The key concern for data-intensive computing is the readiness of

COLLABORATECOM 2013, October 20-23, Austin, United States

DOI 10.4108/icst.collaboratecom.2013.254100

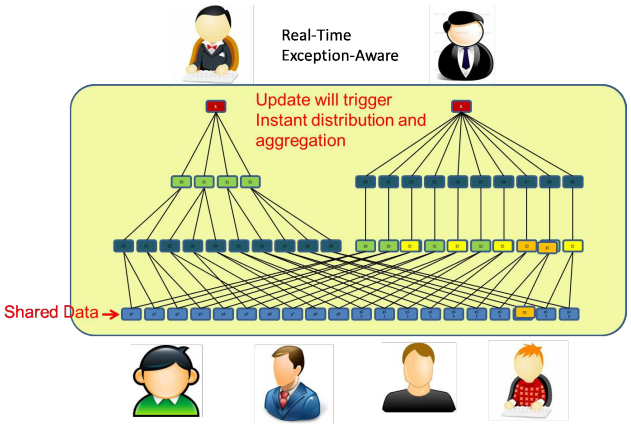

Figure 1. A collaborative planning scenario in business.

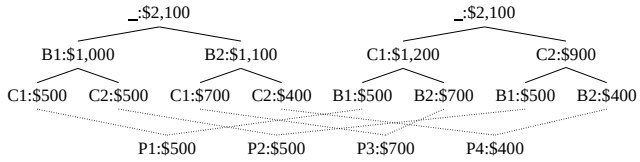

Figure 2. Two views based on the actual sales data in the previous year.

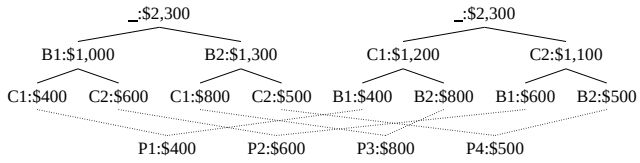

Figure 3. Two views based on the forecasted sales data.

data and its corresponding data structures. The term ”data-centric” has multiple meanings in reference to where data lives, which could mean same location, same machine or the same memory space in different contexts. By definition, in-place computing is data-centric computing as well as in-memory computing, but not vice versa.

64-bit architecture is the key to making in-placing computing technology meaningful and effective. While 2^{32} is quite limited, 2^{64} is considerably unlimited in the sense that a 64-bit address space is large enough to hold the big data in most cases that we see today. With in-place computing in 64-bit address space, it becomes possible for today’s commodity machines to deliver performance approaching the hardware limit (i.e., giga-operations per second) and scalability for the data size growing beyond the swap space size of the running machine.

Due to the high valence of datasets, the collaborative planning problem is best addressed with in-place computing in a single machine in order to avoid the high cost of data shuffling among multiple machines. However, we do not wish to over-promise the capability of the technology. When used in cohesion with a single commodity machine, it is impractical to handle high terabytes or petabytes of data within a reasonable timeframe. It is our planned work in the future to investigate the distributed in-place computing model in order to process data at that scale.

Our experiments show that one hundred millions of sales records in collaborative planning can be computed in a few seconds simply using one core, one CPU, one machine with 64GB RAM. We aim to compute one billion of records of data in seconds on a commodity machine equipped with multiple CPU’s and multiple cores.

II. Real-World Problem – Real-Time Collaborative Planning

We have faced challenges for supply chain planning in China, which is a large yet expansively developing country with a very complicated market environment due to various and diverse cultures, climate conditions, living styles and the ways people conduct business across the country. To be responsive to the market, an enterprise needs to deal with rapidly changing demand-supply situations all year long, and therefore requires its demand chain partners (i.e., channels) and supply chain partners (i.e., suppliers) to work closely and collaboratively to react to market changes. It is important to understand the market trends, project future demand and ensure that the supply can meet the demand. The demand/supply planning, used to be an annual or semi-annual activity conducted solely by executives, has now become a daily activity collectively achieved by both executives and middle-level managers. The business goal of such large-scale collaborative planning is to reach a consensus for sales forecasting.

Let us consider the collaborative planning problem as a game played by a set of participants (i.e., players) who act and react on shared data (i.e., game board) according to a set of rules (i.e., game rules) to determine the legitimate actions and states, and collectively achieve a common goal. The complexity of a collaborative planning is determined by the number of participants, the size of shared data and the degree of interdependencies among data.

As an actual scenario of a real world collaborative planning application, a brand manufacturer with about 200,000 product items (product dimension) and around 10,000 stores (channel dimension) plans the demand and supply based on the sales forecasting of the next 52 weeks and the historical sales from the past two years. Each product has a unique ID and many attributes such as category, sub-category, brand, color, material, etc. Each store also has a unique ID and many attributes, area, city, type, etc. The planning process includes many different manager roles such as the product manager, category manager, sales manager, area manager and brand manager as well as different types of sales data. Fig. 1 illustrates a collaborative planning game, where all managers share forecasted sales data in different hierarchical views. Managers, both executives and middle-level managers, can view and update the data under their roles.

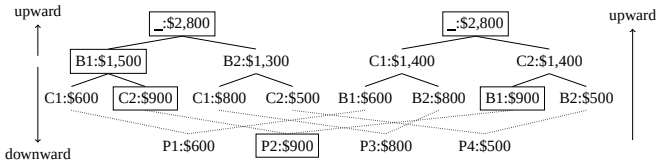

Figure 4. The result after raising the value of B1 in the left view. Alerts are raised for exceptions shown in boxed nodes.

A Simplified Planning Example

To illustrate how to perform a collaborative planning process, we present a simplified example with (1) two managers, brand manager and category manager, and (2) two datasets, actual sales data from the previous year and forecasted sales data. Assume four products, P1, P2, P3 and P4, where P1, P2 belong to brand B1, and P3, P4 belong to brand B2, while P1, P3 are of category C1 and P2, P4 are of category C2. The business goal of the collaborative planning process is for the brand manager and the category manager to reach a consensus for sales forecasting, and discover if any forecasted sales (for product, brand or category) goes beyond expectation.

Fig. 2 shows two views based on the actual sales dataset (i.e., P1, P2, P3 and P4 are shared). Fig. 3 shows two views based on the forecasted sales dataset (i.e., P1, P2, P3 and P4 are shared). The brand manager is interested in both views on the left, showing the (actual and forecasted) sales data grouped by the product's brand first and then by the product's category. That is, the hierarchy of the left views is in a top-down order of *brand* and *category*. Meanwhile, the category manager is interested in both views on the right, showing the (actual and forecasted) sales data grouped by the product's category first and then by the product's brand. That is, the hierarchy of the right views is in a top-down order of *category* and *brand*. Sales numbers shown in this example are all in thousands of dollars.

The value at each leaf node is the (actual or forecasted) sales amount for a product, and the value at each intermediate node is the (actual or forecasted) sales amount for a brand or a category. Both the brand manager and category manager are interested to know if any forecasted sales (product, brand or category) is deviated from the actual sales in the previous year by 30%. They wish to be notified when such exceptions are caught.

Expecting the sales of brand B1 to rise 50% in reaction to an advertising campaign, the brand manager updates the forecasted sales of B1 from 1 million to 1.5 million dollars (as seen in the left view of Fig. 4). This update will immediately affect the forecasted sales for every product that belongs to B1, every category and the total sales. The changes will be propagated through the views in Fig. 4 as follows.

1) **Left View Upward** The total forecasted sales (at root node) is changed to 2.8 million from 2.3 million.
2) **Left View Downward** By proportion, the sales of C1 is changed to 600 thousands from 400 thousands and the value of C2 is changed to 900 thousands from 600 thousands respectively. Next, the sales of P1 is changed to 600 thousands from 400 thousands and the value of P2 is changed to 900 thousands from 600 thousands respectively.
3) **Right View Upward** The sales of B1 under C1 is changed to 600 thousands from 400 thousands and the value of B1 under C2 is changed to 900 thousands from 600 thousands respectively. Next, the sales of C1 is changed to 1,400 thousands from 1,200 thousands and the value of C2 is changed to 1,400 thousands from 1,100 thousands respectively. Finally, the total sales (root node) is changed to 2.8 million from 2.3 million.
4) **Left View Exceptions** The total sales, the sales of B1, the sales of C2 under B1 and the sales of P2 are deviated from the actual sales (Fig. 2) by 30%. They are shown in boxed nodes.
5) **Right View Exceptions** The total sales and the sales of B1 under C2 are deviated from the actual sales (Fig. 2) by 30%. They are shown in boxed nodes.

Although the above scenario looks simple, the computation is complex and time-consuming when the volume of data is huge and a considerable number of participants are involved. It is a challenging task to design a collaborative planning application with real-time response.

Collaborative Planning Application

Let us return to the game model of collaborative planning of supply chains. The planning (game) is conducted by having different mangers share and adjust weekly forecasted data from different viewpoints. In our application, the sales manager sees a view with a hierarchy in the order of area, city, store, category, sub-category, brand and product, while the product manager sees a view with a hierarchy in the order of category, sub-category, brand, product, area, city and store. A category manager or a brand manager, who reports to the product manager, sees a partial view of what the product manager sees, and an area manager, who reports to the sales manager, sees a partial view of what the sales manager sees. However, no matter what views they see, all of them share the same product-store data (for both actual sales and forecasted sales), which are located at the leaf nodes of the view hierarchies. The sales manager and product manager have options to create or change a view hierarchy by adding, removing or switching attributes.

There are three major types of computing operations in our planning application described as follows.

- **Aggregation** The data values of intermediate nodes in a hierarchy are determined by an aggregation rule, which is not necessarily the summation function. When a view is created, the data values of intermediate nodes will be calculated based on an aggregation rule upward starting from the leaf nodes. When a data value is changed, either at a leaf node or at an intermediate node, all intermediate nodes above the changed node will be re-calculated automatically.

- **Distribution** Besides aggregation, when a manager changes the data value at an intermediate node or the root node, the system will automatically distribute the changes downward the hierarchy according to a distribution rule (e.g., even distribution, proportional distribution or weighted distribution). Once the changes are distributed to the leaf nodes, the system will automatically trigger the aggregation process for all view hierarchies that share the same product-store data at the leaf level. The distribution process is not applied to actual sales data, since they are historical and irreversible.
- **Exception Detection** While facing a large amount of data in a hierarchy, it is very difficult for a manager to examine data in the hierarchy manually. It is helpful for managers to set up a set of exception criteria rules, which describe whether the data patterns or relations go beyond their expectation. These exception criteria rules will be triggered to check against the data once distribution and aggregation operations are completed. The managers expect the system to catch exceptions in real time and raise alerts. For example, the sales manager wishes to be alerted for forecast accuracy exceptions when the actual sales is deviated from the forecasted one for some percentage. Based on the exception criteria formula, the application will compare the forecasted data against the actual sales and raise necessary exceptions.

Challenges

A collaborative planning application can be designed based on the traditional computing model. However, it is impractical to build such a planning application with reasonable response time. The challenges are listed below.

- **Large Data Movement** The operations, aggregation, distribution and exception detection, require access to the entire tables of actual sales and forecasted sales. Retrieving large size of data from the database will take excessive I/O and network overhead.
- **Bounded Virtual Memory** Even the 64-bit address space is considerably unlimited, the (virtual) memory that can be allocated from the heap space in a program is bounded to the size of swap space. It is not scalable without adding more RAM or increasing the swap space.
- **Paging Anomaly** The main-memory data structures allocated from the heap space to hold retrieved data are also in virtual memory, which are subject to swapping. It is possible that swapping occurs immediately after data is retrieved from the database into memory, leading to severe I/O performance problems.
- **Large Data Update** Both distribution and exception detection operations require large data update in a database resulting in severe performance problems. SQL update and insert are expensive operations.
- **Juggling** Being aware of memory space too small to fit all data at once, application developers are forced to design algorithms (i.e. out-of-core or external-memory algorithms [6], [7]) to process data with small amount of memory and later merge the results.
- **Relationality** It is common to spend a lot of time and effort tuning relational databases for big-data analytics-type applications, which often need to scan through a large portion of data from several tables. Relational databases are not reliable in terms of consistent performance. Performance could be further degraded due to compound SQL statements or large multi-way joins. The recent development of NoSQL [8]–[10] is to address such performance issues.

TABLE I
MYSQL `GROUP BY/ROLLUP` TIME

# Records	1M	5M	10M	50M	100M
Time	6.03	36.12	72.86	549.92	1520.64

TABLE II
MYSQL `UPDATE` TIME

# Records	1M	5M	10M	50M	100M
Time	11.25	61.91	128.83	677.06	1353.50

Relational database designers understood the performance issues of large data movement and have supported stored procedures as well as some built-in functions to run inside the database servers. Simple aggregation operation such as summation can be implemented using SQL `ROLLUP` [11] operator.

To have an idea of how relational databases perform, we conducted experiments with SQL `ROLLUP` operator using several open-source databases. Since all tested databases gave similar results, we only show the results of MySQL [12] database for illustration purposes. Table I shows the time in seconds to perform aggregation by `GROUP BY` with `ROLLUP` operator using the following SQL statement:

```
SELECT C.area, M.store, P.category, M.product,
    SUM(M.sales) AS s_sum
FROM M, P, C
WHERE M.product=P.product AND M.store=C.store
GROUP BY C.area, M.store, P.category, M.product
    WITH ROLLUP
```

Listing 1. Aggregation is accomplished by SQL `ROLLUP` operator.

Relational databases spend a considerable amount of time for large data writes such as `INSERT` or `UPDATE`, which are needed by both distribution and exception detection. Table II shows the time in seconds to perform `UPDATE` for MySQL.

Apparently, when the data size is large, such as 100 million, it is difficult for a relational database to meet the real-time requirement for a collaborative planning application. This leads us to move away from relational databases and develop BigOject, a general-purpose computing system, as the foundation of our collaborative planning application.

III. BIGOBJECT

BigObject is an in-place computing system for multi-dimensional data. Data are organized in *big objects*, which

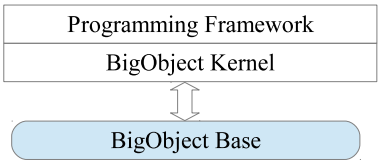

Figure 5. Architecture of BigObject.

are in place in 64-bit memory space and in place to execute an expression or an algorithm. Similar to macromolecules in living cells, big objects are also well organized that all necessary data and data structures are self-contained, having no need to allocate memory from the heap space. This is an important requirement to guarantee scalability for data size growing beyond the swap space size of the running machine.

In order to implement an efficient in-place computing model, BigObject supports three critical mechanisms, (1) in-memory mechanism, (2) transformation mechanism, and (3) data-centric computing mechanism. This first describes how big objects reside in 64-bit memory space persistently. The second describes how to create or transform big objects from raw data or other big objects. And the third describes a programming framework to run code directly on big objects.

Similar to Unix/C architecture, the architecture of BigObject consists of three layers, *BigObject base*, *kernel*, and *programming framework*, as shown in Fig. 5. BigObject base is a persistent data space as well as a computing space, in which big objects are born, live, and work. Like Unix kernel, BigObject kernel provides a set of functions to create and manipulate big objects. The top layer of the architecture is a programming framework, which supports language constructs and API for developers to write code with big objects.

BigObject base can be treated as an in-memory database if a query language is provided. Like most NoSQL systems [8], [9], it is possible to implement SQL for BigObject. However, our intention is to stay away from the computing approach that relies heavily on data retrieval during computation. The programming framework is not designed for query purposes, but used to program real algorithms for the computing purpose. Efficiency (i.e., performance) and expressiveness (i.e., high-level way to manipulate big data) are two key design principles for BigObject.

A. Big Objects

BigObject organizes data in multi-dimensional space. A dimension is composed of a set of members, each of which has certain properties called *attributes*. For example, age and gender are common attributes in customer dimension. Data to be analyzed, such as actual sales or forecasted sales, are called *measures*. Multi-dimensional data are naturally defined in relational databases with star schema. For example, Table III and IV define two dimensions, *channel* and *product*, and their attributes, *area* and *category*. Table V is a measure table of sales records, each row indicating the sales amount for a product sold at a store.

TABLE III
A CHANNEL DIMENSION TABLE

channel.store	*channel.area*
S1	NY
S2	NY
S3	CA

TABLE IV
A PRODUCT DIMENSION TABLE

product.sku	*product.category*
P1	Food
P2	Food
P3	Cloth

TABLE V
A SALES MEASURE TABLE

channel.store	*product.sku*	sales_value
S1	P1	6
S1	P2	5
S1	P3	7
S2	P3	8
S3	P1	4
S3	P2	9

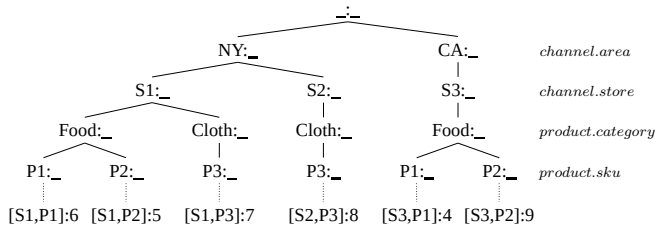

Figure 6. A tree object *actual-sales* with hierarchy: *channel.area* $\prec$ *channel.store* $\prec$ *product.category* $\prec$ *product.sku*.

Both dimensions and measures are organized into table objects, with a structure similar to table in relational databases. In our example, *sales* object is a table object.

Another kind of object in BigObject is tree object. A tree object has a hierarchical structure that is suitable for performing analytical computation efficiently. Fig. 6 shows a tree object derived from *sales* object, where hierarchical attributes from top to bottom are *channel.area* $\prec$ *channel.store* $\prec$ *product.category* $\prec$ *product.sku*. We use $a_1 \prec a_2$ to denote that the level of attribute a_1 is higher than the level of a_2 in the tree hierarchy. For example, the root of *sales* object has two child nodes holding data: NY:_ and CA:_ as shown in Fig. 6.

The data structures of the table objects and tree objects look like any normal data structures we often see in C/C++ or Java. However, they are implemented to hold the following properties:

- Big – capable of maintaining a large size of data elements, addressable in one big space (e.g. 64-bit address space).
- In-place – ready to compute, no query, no explicit data retrieval.
- Persistent – viable to keep the state, which outlives the process that created it.
- Relocatable – movable from one space to another, in same machine or different machines.

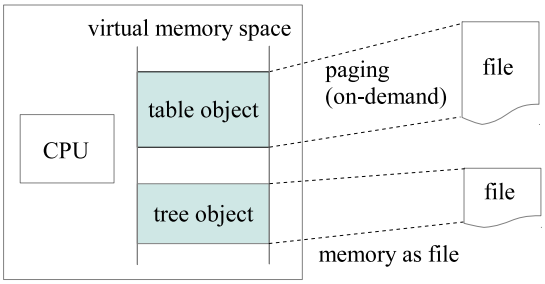

Figure 7. Memory-mapped big objects.

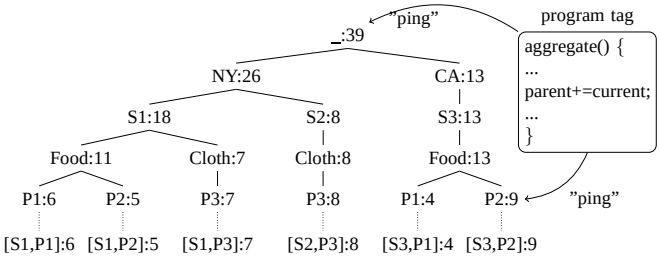

Figure 8. Aggregation using a program tag.

B. In-Memory Mechanism

Big objects are implemented using the memory mapping technique. Memory mapping that implements demand paging became available for application developers in POSIX-compliant Unix since the 1980s. This allows programs to allocate virtual memory space backed by a memory-mapped file. As shown in Fig. 7, a memory-mapped file is a segment of virtual memory assigned a direct mapping with a file, a block device or any resource that can be opened for random access

The adoption of memory-mapped files resolves the performance problems such as data movement, juggling and swapping that we discussed in the previous section. As a storage unit, the memory space used in a big object is automatically mapped and saved in the memory-mapped file. As a computing unit, algorithms can be directly applied on the memory structure of a big object transparently without any explicit data retrieval and additional memory allocation.

The beauty of the memory-mapping approach is that a big object is simply a file, in which the content of the object can be preserved even if the system is powered off, or move around in different machines. It can also serve as a paging device when the big object is computing. In other words, memory-mapped big objects meet the desired four properties (big, in-place, persistent, and relocatable). More importantly, the size of a memory-mapped file can be bigger than the size of the swap space, which supports a scalable solution to hold and manipulate data in 64-bit address space (or whatever the address space limitation of OS, 42-bit or 48-bit for Linux).

C. Transformation Mechanism

Transformation is a common technique used in science, which converts a problem into some form easy to manipulate. The **mapper** in MapReduce is an example of transformation, which transforms a key-value domain into another. However, transformation doesn't come free. It always incurs cost, more or less depending on the underlying implementation. The idea of transformation mechanism is to transform data into a data structure suitable for efficient computation, assuming the cost of transformation can be tolerated.

BigObject adopts a transformation mechanism, *Transformative Join* (*Trans-Join* in short), by using a similar semantics of join from relational algabra [13]. Trans-join transforms a big object into a tree object according to a desired hierarchy. It is a binary operator which takes a big object as the first operand and an attribute of a dimension as the second operand, and creates a tree object. A trans-join operation "lifts" the hierarchy of the tree object one level up by grouping the leaf nodes by the attribute. For each group of leaf nodes that share the same attribute value, an intermediate node is created as the new parent node of the group. As a result, a new level of intermediate nodes are inserted between the leaf nodes and the original parent nodes.

For example, the tree object *actual-sales* in Fig. 6 is trans-joined from the *sales measure* table object in Table V based on hierarchy *channel.area* $\prec$ *channel.store* $\prec$ *product.category* $\prec$ *product.sku*. This process is similar to using `GROUP BY` with `ROLLUP` operator on columns *channel.area*, *channel.store*, *product.category*, and *product.sku* by joining *sales measure*, *channel*, and *product* tables in SQL.

Trans-join operator differs from the `JOIN/GROUP BY/ROLLLUP` operators in that it physically constructs an object with tree structure rather than outputs a table in relational model [11]. Since trans-join can be used to construct a tree object with the desired level of hierarchy, which allows us to create the most suitable tree object for efficient computing tasks. In our collaborative planning application, each different hierarchical view, i.e., tree object, can be easily constructed by using trans-join. The data is shared among all tree objects at the leaf level and the aggregated values in intermediate nodes are computed and analyzed accordingly.

D. Data Centric Computing Mechanism

One of the key ingredients of in-place computing is data centric computing, in which computations take place where the data resides. Instead of using query languages such as SQL to retrieve necessary data out of databases, BigObject adopts a simple programming framework, which allows (1) a piece of program to attach to a tree object node for execution and (2) expressing and evaluating an arithmetic expression of big objects. This framework together with compiler optimization not only provides a high-level expressive power, but also delivers good performance. The first feature in the programming framework is realized by a language construct called *Program Tags*, and the second feature is done by *Macro Expression*.

Program Tag

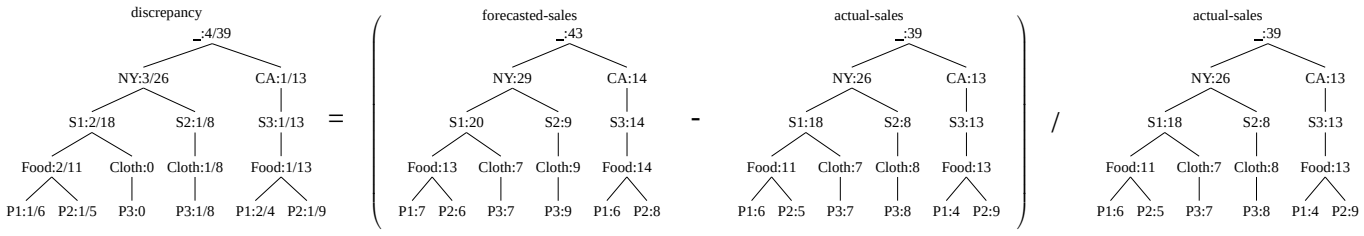

Figure 9. Illustration of macro expression evaluation for equation 1.

```
void discrepancy(BO<int> D, BO<int> F, BO<int> S)
{
    F.aggregate();
    S.aggregate();
    D = (F - S) / S;
}
```

Listing 2. A function consists of a macro expression

A program tag is a piece of program containing statements and variables, which is used to implement a node-level function for a tree object. Once a program tag is defined, it can be attached to some nodes of a tree object and then evaluated node-by-node in a tree traversal order. For example, to implement aggregation operation for a tree object, we can "ping" each node in the tree object with a program tag as shown in Fig. 8. In this aggregation example, each node simply adds its value to its parent's node in a depth-first traversal order. Program tags can be designed to behave differently in different levels and can be parameterized at invocation. The programming framework allows developers to define program tags to perform desired computation on each node in a high-level way.

In our collaborative planning application, both distribution and aggregation operations are implemented using program tags.

Macro Expression

Similar to matrix arithmetic expression, BigObject programming framework supports the application developers to write and evaluate an arithmetic expression over big objects, where operators are either pre-defined or overloaded. The following equation is a macro expression to calculate the discrepancy between actual sales and forecasted sales:

$$D = (F - S)/S \tag{1}$$

, where S is the *actual-sales* object, F represents the *forecasted-sales* object and D is the *discrepancy* object holding the result. It indicates forecast inaccuracy. Fig. 9 shows an example of how this equation is evaluated. BigObject programming framework allows us to program this equation in a high-level way as shown in Listing 2. Therefore, the exception detection in our collaborative planning application can be easily implemented using macro expressions.

TABLE VI
OBJECT CREATION TIME

# Records	1M	5M	10M	50M	100M
Trans-join	0.38	1.87	3.75	19.28	39.25

For efficiency, program tags and macro expressions are compiled to C++ code first by a language compiler and then the generated C++ code is further compiled into an executable library by a C++ compiler. Compiled executables are dynamically linked and sent to BigObject system for execution.

E. Concurrency Control

In order to address the concurrency issues, we adopt a simple read-write lock concurrency control by locking views while share data is being updated. Recoverability is not a concern since this is a planning job rather than a transaction that requires durability.

IV. PERFORMANCE EVALUATION

BigObject is a data-centric approach, different from the traditional computing approach that requires data retrieval from databases. Without the overhead of data retrieval, it is anticipated that a BigObject-based application should perform more efficient than an application using relational databases. It may not be easy to compare BigObject to relational databases since there is no benchmark program that can run both. Instead, we compare two applications that take different approaches but deliver the same results with the same datasets.

We have built a collaborative planning application, which supports collaboration with multiple views and three operations, aggregation, distribution, and exception detection, using BigObject. The computation part is written in C++ and BigObject programming framework, and the GUI part is implemented in Java with ZK, an open-source Ajax [14] Web application framework. Since our goal is to develop an affordable technology, we constrain the hardware used for the planning system and BigObject engine to a commodity machine with 64-bit CPU at 2.3 GHz, 64GB of RAM, and 1.0TB of hard disk. The machine runs 64-bit Ubuntu Linux (version 12.10).

We conducted experiments with different sizes of records. Table VI shows the time in seconds to build a view (hierarchy as in Fig. 6) by transforming a table object into a tree object.

TABLE VII
OPERATION TIME

# Records	1M	5M	10M	50M	100M
Aggregation	0.065	0.294	0.565	2.78	5.49
Distribution	0.090	0.416	0.837	4.13	8.19
Exception	0.072	0.305	0.609	3.00	6.06

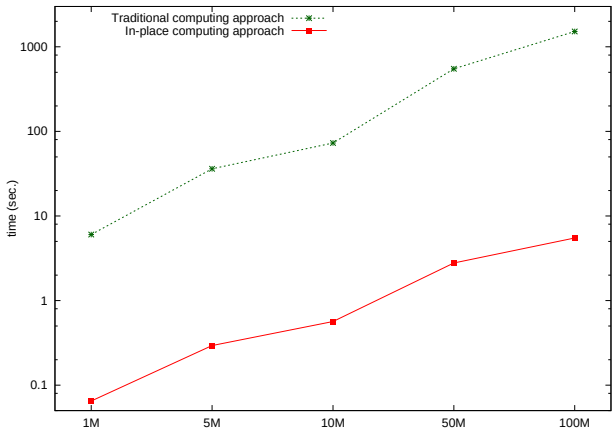

Figure 10. Performance of aggregation between a traditional computing approach and the in-place computing approach.

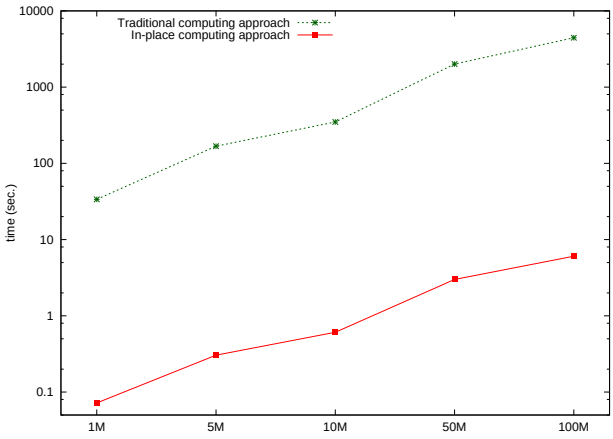

Figure 11. Performance of exception detection between a traditional computing approach and the in-place computing approach.

TABLE VIII
STORAGE SIZE FOR DIFFERENT COMPUTING APPROACH

# Records	1M	5M	10M	50M	100M
A MySQL Table	81MB	410MB	822MB	4.1GB	8.3GB
A Table Object	20MB	93MB	184MB	1.0GB	2.2GB
A Tree Object	20MB	97MB	193MB	964MB	1.9GB

Table VII shows the time in seconds to perform aggregation, distribution, and exception detection for a tree object.

We implemented a second application that supports aggregation and exception detection using the traditional computing model, and experimented the application with several open-source databases, which all deliver quite similar results. For simplicity, we only show the results with MySQL (version 5.5.31) database, in comparison with the first application. The two operations, aggregation and exception detection in this application, are implemented in two different ways. Firstly, the aggregation operation is written in one SQL statement as shown in Listing 1 in the previous section and runs on the database server. Secondly, the exception detection operation is a C++ function, which retrieves the forecasted sales and actual sales data from database into an internal data structure, computes intermediate results if needed, performs the discrepancy formula (see Equation 1) and finally records the exceptions when detected.

By examining the aggregation time for the application using the traditional computing model (as shown in Table I) and the aggregation time for our planning application (as shown in Table VII), it is clear to see that our application outperforms the application using the traditional computing model in two orders of magnitude. When the data size is 100 million (in records), our application takes 5.49 seconds to aggregate, while traditional computing approach takes more than 1520 seconds to do `GROUP BY/ROLLUP` operation. It is about 276 times faster. Fig. 10 shows the performance comparison between our application which is based on in-place computing model and the application based on the traditional computing model.

SQL was originally designed as a query language, rather than a language to implement complex algorithms. It is noted that even SQL is able to perform aggregation using `ROLLUP` operator, it is nontrivial to come up with a single SQL statement to perform distribution operation. A proportional distribution operation needs to first calculate the weight for each child node in the view hierarchy and then distribute the changed amount to the child nodes according to their weights.

Implementing exception detection using pure SQL is a complex operation. An exception detection operation needs to perform an arithmetic expression from multiple data components, calculate the discrepancies, check against the exception criteria and finally record the exceptions in details. Developing a collaborative planning application solely with SQL to accomplish all three major operations (i.e., aggregation, distribution, and exception detection) is a daunting task.

Fig. 11 shows the experiment results of exception detection. It is clear that the in-place computing approach is more efficient than the traditional computing approach. In this experiment, our application takes 6.057 seconds for data size of 100 million (in records), while the application based on traditional computing model takes 4446 seconds. It is about 734 times faster.

We also compare the storage sizes between the two computing approaches. Table VIII shows the size of a table (see Table V) in MySQL, the size of a table object and a tree object in BigObject for different numbers of sales records. A big object is much smaller than a table in MySQL due to concise record structures and string encoding scheme utilized in BigObject [15]–[17]. The string encoding scheme is also helpful in reducing time for trans-join operation.

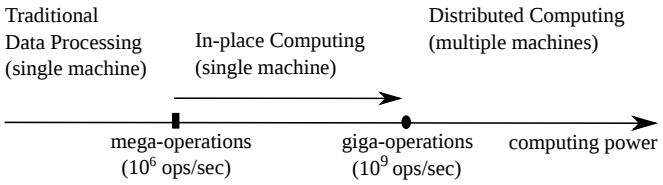

Figure 12. Spectrum for computing models on commodity machines

The performance of applications using the traditional computing model suffers from large volume of data retrieval and update. By using in-place computing technology, the experiment results show that a collaborative planning application can be built to handle big data with acceptable response time using a commodity machine. The question left for the readers is whether to trade the traditional approach based on relational databases for such performance gain.

V. Conclusion

A real-world collaborative planning application is complex. The complexity depends on the number of concurrent participants in the collaboration as well as the data characteristics, volume and valence, which determine the performance of the application. The traditional computing model based on databases is inadequate for processing big data with high valence. Meanwhile, most commonly adopted approaches for big data computing such as Hadoop [18] were not designed specifically for real-time computing and may suffer from performance problems due to large data shuffling among multiple servers. Instead, we take an unconventional approach, in-place computing model to address the issues raised by big volume and high valence. BigObject is an in-place computing system designed for storing and computing multi-dimensional data. It focuses on 'squeezing' (i.e., scaling in) computing performance out of the CPU's of a single machine by trading space (complexity) with time (complexity). The general availability of 64-bit architecture makes this possible and triggers revisits to many software areas to rethink alternatives for potential performance leap.

By building a collaborative planning application based on BigObject, we are able to study the behavior of the in-place computing system and evaluate its performance against the traditional approach. In the application, BigObject demonstrates with significant performance gain and well-organized complexity in both data structure and behavior. Our experiment shows that the application takes seconds to compute one hundred millions of data records using one core, one CPU and one machine. In-place computing approach outperforms the approach based on traditional computing model in two orders of magnitude.

As part of the result from the experiment, we have learned the comfort zone for in-place computing. Fig. 12 shows a spectrum of computing power and the corresponding computing models on today's commodity machines. In-place computing model is capable of delivering computing power between million operations per second (mega-operations) and billion operations per second (giga-operations). It is feasible and practical to process data up to one billion of records (or low terabytes of raw data) within a reasonable timeframe using one commodity machine with multiple cores and multiple CPU's. When processing data more than billions of records, it will require a distributed approach. In-place computing offers an affordable solution for processing data with high valence and volume up to one billion of records.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10.

[2] D. Laney, "3-d data management: Controlling data volume, velocity and variety," *META Group Research Note, February*, vol. 6, 2001.

[3] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow, "The anatomy of the facebook social graph," *arXiv preprint arXiv:1111.4503*, 2011.

[4] Y. Gao, "Data centric computing for internet scale enterprises." in *Fourth ACM/SPEC International Conference on Performance Engineering*, 2013, pp. 347–348.

[5] R. F. Freitas and W. W. Wilcke, "Storage-class memory: The next storage system technology," *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 439–447, 2008.

[6] S. S. Chawathe, "Comparing hierarchical data in external memory," in *Proceedings of the International Conference on Very Large Data Bases*. Citeseer, 1999, pp. 90–101.

[7] Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter, "External-memory graph algorithms," in *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 1995, pp. 139–149.

[8] R. Cattell, "Scalable sql and nosql data stores," *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2011.

[9] M. Stonebraker, "Sql databases v. nosql databases," *Communications of the ACM*, vol. 53, no. 4, pp. 10–11, 2010.

[10] N. Leavitt, "Will nosql databases live up to their promise?" *Computer*, vol. 43, no. 2, pp. 12–14, 2010.

[11] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, "Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals," *Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 29–53, 1997.

[12] P. DuBois, *MySQL*. Pearson Education, 2008.

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

[14] J. J. Garrett *et al.*, "Ajax: A new approach to web applications," 2005.

[15] G. Graefe and L. D. Shapiro, "Data compression and database performance," in *Applied Computing, 1991.,[Proceedings of the 1991] Symposium on*. IEEE, 1991, pp. 22–27.

[16] M. A. Roth and S. J. Van Horn, "Database compression," *ACM Sigmod Record*, vol. 22, no. 3, pp. 31–39, 1993.

[17] D. Abadi, S. Madden, and M. Ferreira, "Integrating compression and execution in column-oriented database systems," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM, 2006, pp. 671–682.

[18] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–10.