Efficient Large-scale Data Analysis Using MapReduce

Rui Kubo, Yoshifumi Fukumoto, and Makoto Onizuka

Abstract

Techniques for efficiently analyzing large-scale collections of data, such as log data or sensor data, are attracting attention as important for improving the operating speed of businesses and enhancing the user experience. This article outlines MapReduce, which is a typical framework for analyzing big data, and introduces our techniques for achieving high-speed analytical processing by extending MapReduce.

1. Introduction

1.1 Background of MapReduce

MapReduce is a framework for efficiently processing the analysis of big data on a large number of servers. It was developed for the back end of Google's search engine to enable a large number of commodity servers*1 to efficiently process the analysis of huge numbers of webpages collected from all over the world [1].

On the basis of this research, software was developed*2 by the Apache project to implement MapReduce, which was published as open source software (OSS) [2]. Since anyone can use OSS, this enabled many organizations, such as businesses and universities, to tackle big data analysis [3] in a way that only a few research institutes could do in the past owing to the complexity and high cost of the analytical system.

In the NTT Group, there is also an increasing number of cases involving big data analysis in a wide range of situations such as back-end systems for search engines, analysis and reporting systems for communications traffic and logs, and advertising recommendation engines.

*1 Commodity server: An ordinary inexpensive personal-computer-based server using commodity components.

*2 The OSS community is actively advancing development by adding functions and fixing bugs.

1.2 Mechanism of MapReduce

MapReduce provides an application programming interface (API) and middleware for developers who want to analyze big data. More specifically, when a developer defines processing that is compliant with two types of APIs, the map and reduce functions, processing is distributed efficiently for execution on multiple servers (**Fig. 1**).

The mechanism of MapReduce ensures that the complexity involved in developing a system that works on numerous servers can be concealed from the developer. More specifically, this approach has the following advantages.

- Scheduling: The servers that execute the processing defined by the map and reduce functions are selected automatically by the middleware. The middleware selects a nearby server in which a chunk of the data is stored to be the map function execution server. This reduces the volume of data transfers and enables efficient processing.
- Synchronized processing: data transfers between servers for the map and reduce functions are synchronized.
- Fault tolerance: To ensure that processing can continue overall even when several servers have failed, data backups and intermediate processing results are stored automatically. If a failure actually occurs, servers restart the processing using the backups and intermediate processing results.

The scheduling and synchronization processing become more complex as the scale of the data and

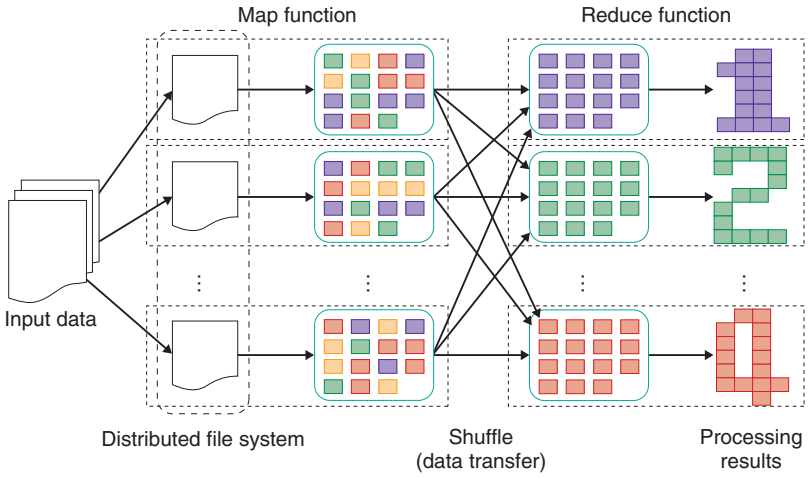


Fig. 1. Flow of MapReduce processing.

servers increases; thus, server breakdowns often occur. Since such complexities are concealed from the developer by MapReduce, he or she can focus on developing big data analysis methods.

We have been researching and developing techniques for achieving high-speed analytical processing by extending MapReduce. Increased efficiency is extremely important for achieving cost advantages such as improving the system's response time and reducing the number of servers. Below, we introduce the features and suitable usage situations for three techniques that we have developed: PJoin, Map Multi-Reduce, and MRFusion.

2. New techniques

2.1 PJoin

There are many developers who have used structured query language (SQL) when querying a database. Data manipulations that are frequently used with SQL include selecting, projecting, aggregating, and joining. Such data manipulations are frequently used in various situations. However, if databases that were designed some time ago are now given big data to handle, problems arise: the processing time is long and much time is necessary to obtain results. Therefore, there is a demand for techniques that produce results even more efficiently, even with big data.

PJoin can be used to join data efficiently. For example, if we analyze the log of an e-commerce site, we can obtain a best-seller product ranking for each profile by joining user data (profiles of all the users, such as their ages and genders) and purchase history data (purchase histories of all the products), as shown in **Fig. 2**. PJoin makes it possible to quickly obtain such results even more efficiently.

Previously proposed techniques*3 [4] that can be used to perform joining with MapReduce have drawbacks. One drawback is constraints on the size of input data and another is the large volume of communications*4 generated between servers, which forms bottlenecks, so the results cannot be obtained efficiently.

In the preprocessing phase, PJoin creates data that represent reference information between data to be joined*5 and rearranges the input data to enable efficient joining. This overcomes the drawbacks of the previously proposed techniques and makes it possible to quickly obtain results efficiently, even when joining big data (**Fig. 3**).

Since a feature of PJoin is to perform quick and efficient joining by performing preprocessing, it is most suitable for use in situations such as querying repeated joins with large chunks of data.

2.2 Map Multi-Reduce

Map Multi-Reduce can be used to efficiently

*3 Typical techniques are Memory-backed Join, Map-side Join, and Reduce-side Join.
*4 Processing that hands over generated processing results between the map and reduce functions.
*5 Only some of the rows, such as a primary key and external keys, are joined beforehand, as in the semi-join method in distributed database studies.

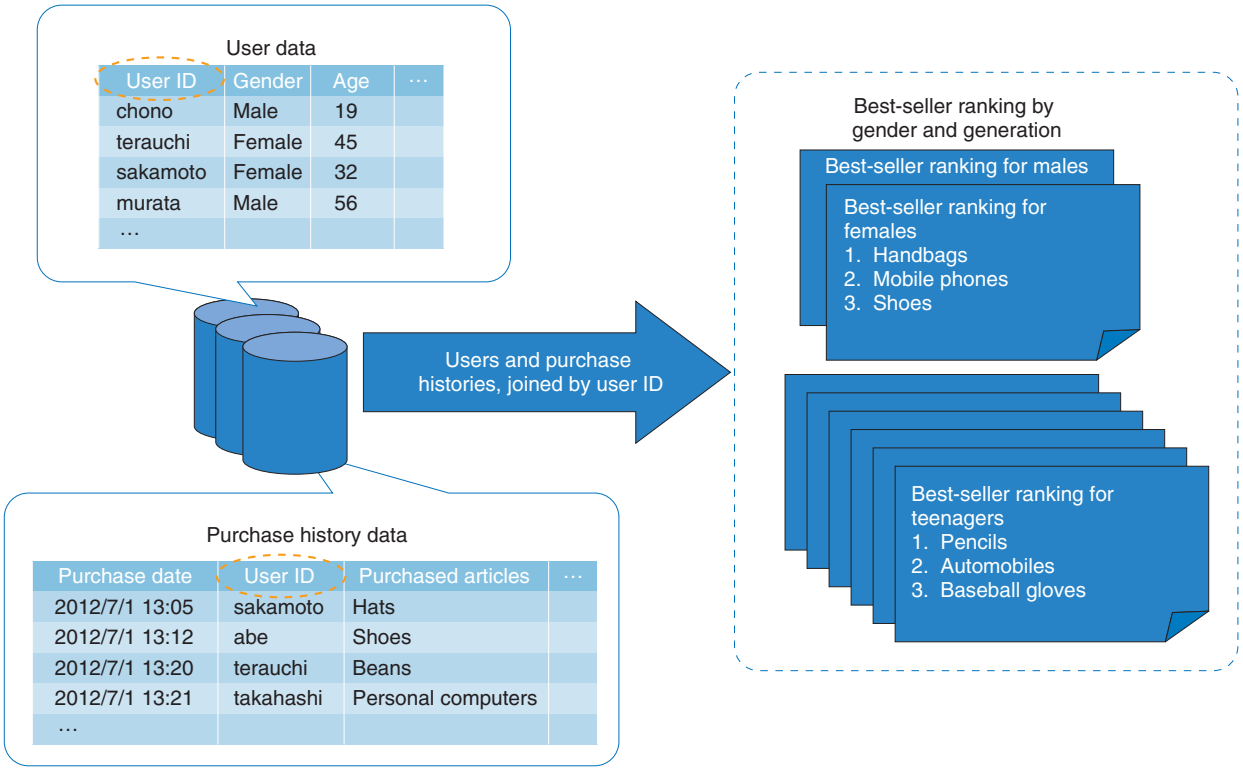
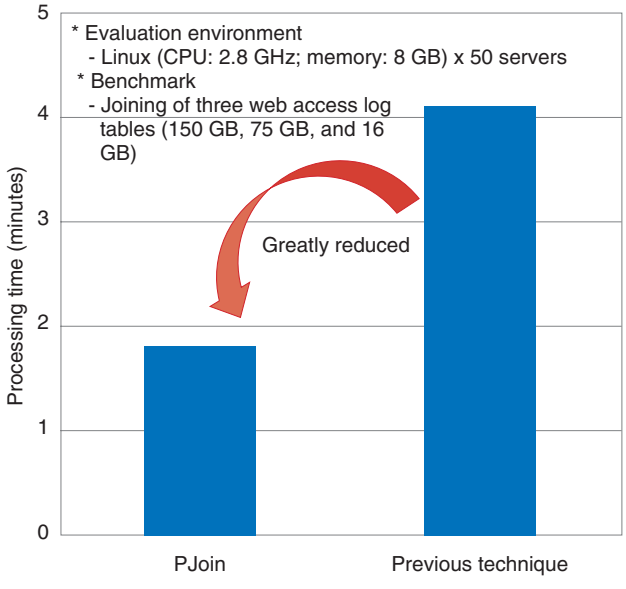


Fig. 2. Example of efficient joining by PJoin.



CPU: central processing unit

Fig. 3. PJoin processing time.

perform aggregations in data manipulations such as SQL. When analyzing the log of an e-commerce site, this technique makes it possible to perform calculations efficiently to obtain total or average daily or seasonal sales from the purchase histories of all products, as shown in **Fig. 4**.

To increase the speed of aggregation by MapReduce, it is common to reduce the volume of data being transferred between the map and reduce functions without changing the overall processing results[*6]. Previously proposed techniques[*7] have various drawbacks, such as placing a larger load on the developer, but attempts to reduce this load also reduce efficiency.

Our Map Multi-Reduce is an extension of the MapReduce middleware. It enables the reduction of a wider range of data transferred between the map and reduce functions on the middleware side. This overcomes the drawbacks of the previously proposed techniques, enabling results to be quickly obtained

*6 Processing that is equivalent to the reduce function, executed beforehand by the server that executes the map function.
*7 Typical techniques are Combiner and In-mapper Combining.

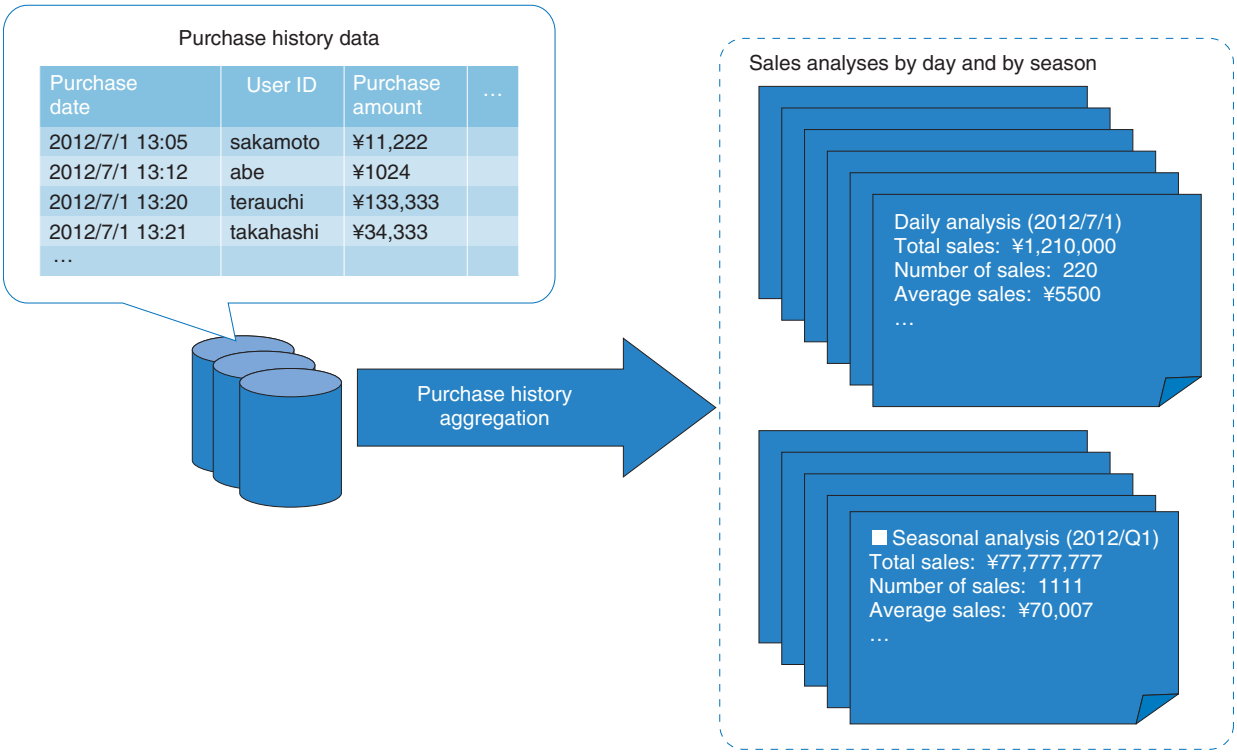


Fig. 4. Example of efficient aggregation with Map Multi-Reduce.

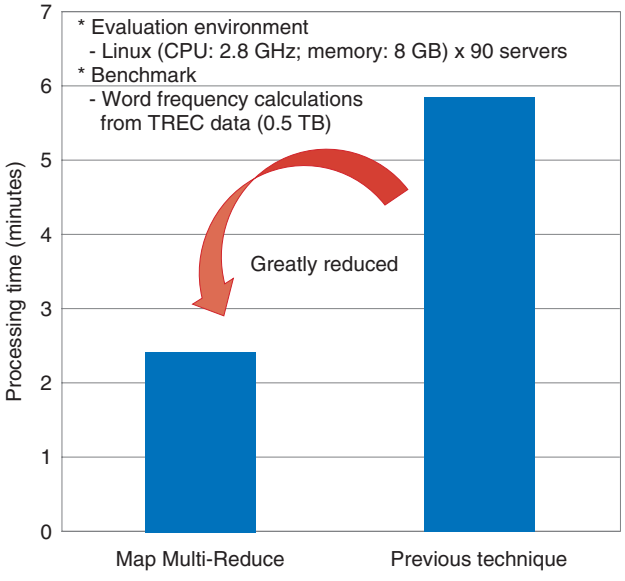


Fig. 5. Map Multi-Reduce processing time.

efficiently while keeping the load on the developer small (**Fig. 5**). When Map Multi-Reduce is used, there is an overhead[*8] due to the reduction of the transferred data, so it is most suitable for use with data and processing where the reduction effect is expected to be large[*9].

2.3 MRFusion

MRFusion is a technique that can be used to repeat analytical processing of the same data. For example, to optimize product recommendations on an e-commerce site, such as that shown in **Fig. 6**, it is necessary to apply various kinds of analytical processing to histories of past purchases by trial and error. If the trial and error is not done sufficiently well, accurate recommendation models cannot be constructed and the e-commerce site will recommend products that the customer does not want. MRFusion ensures that the trial and error (iterative processing) for con-

*8 Overhead: Processing time that initially did not exist.
*9 In the example of the e-commerce site discussed in this article, the reduction effect is larger for the processing to obtain total sales for each season than that for daily total sales.

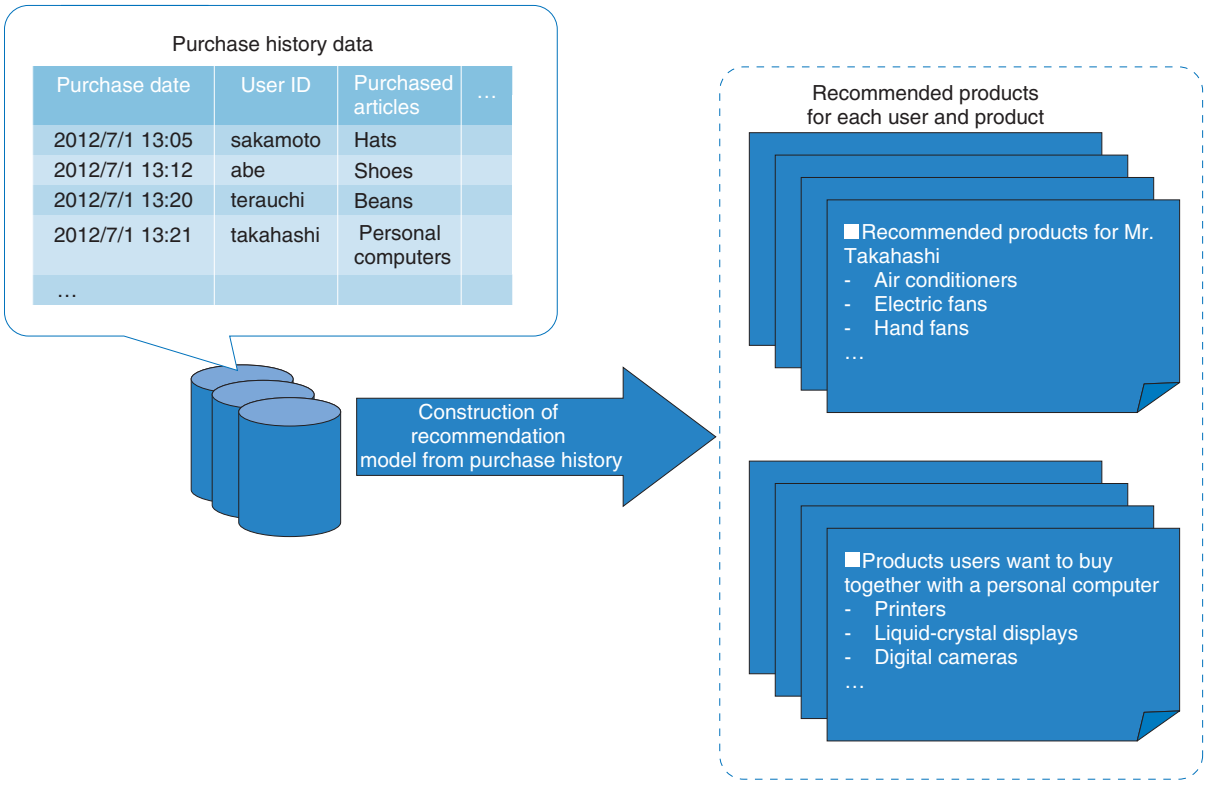


Fig. 6. Example of efficient model construction with MRFusion.

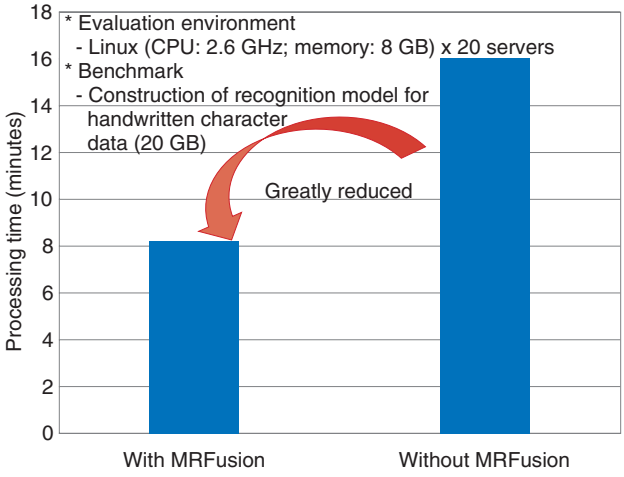


Fig. 7. MRFusion processing time.

structing valuable models can be done quickly and efficiently.

Usually with MapReduce, when a developer wants to repeat analytical processing efficiently as described above, he or she must personally create the processing logic to enable acceleration for each variation of the analytical and iterative processing every time. This imposes a large load on the developer and also hinders rapid introduction of the latest analysis algorithms.

As an extended version of the MapReduce middleware, MRFusion provides automatic detection of the same processing parts among multiple processing with the same data, combines them, and performs batch execution. This keeps the load on the developer small and enables the quick and efficient execution of multiple processing such as that described above (**Fig. 7**). Forcibly expanding the automatic detection range in this way can sometimes lead to side effects. Therefore, when MRFusion is being used, the most suitable usage method is to first apply it to small-scale data, check for an acceleration effect and any side effects, and then, if there do not seem to be any side effects, apply it to big data.

3. Concluding remarks

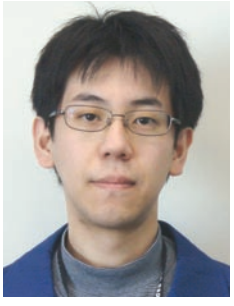
Attention is focusing on cloud computing, and the importance of using big data is now being recognized. It is also becoming more and more necessary to have information processing systems that are capable of processing big data.

We introduced three techniques that we have implemented to extend MapReduce and increase its speed. In the future, we plan to research and develop techniques for efficient processing that can be used over a wider range, as well as various easy-to-use techniques, to help create value-added services that use big data.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," OSDI '04. http://static.usenix.org/events/osdi04/tech/full_papers/dean/dean_html/index.html
[2] Apache Hadoop. http://hadoop.apache.org/
[3] Hadoop Wiki. http://wiki.apache.org/hadoop/PoweredBy
[4] J. Lin and C. Dyer, "Data-Intensive Text Processing with MapReduce," Morgan and Claypool Publishers, 2010.

Rui Kubo
Research Engineer, Distributed Computing Technology Project, NTT Software Innovation Center.
He received the M.S. degree in information systems from the University of Electro-Communications, Tokyo, in 2003. Since joining NTT in 2003, he has been engaged in R&D of sensor networks and distributed computing. As a result of organizational changes in July 2012, he is now in NTT Software Innovation Center. He is a member of the Japanese Society for Artificial Intelligence.



Yoshifumi Fukumoto
Research Engineer, Distributed Computing Technology Project, NTT Software Innovation Center.
He received the B.E. degree from the Faculty of Environment and Information Studies from Keio University, Kanagawa, in 2009. Since joining NTT in 2009, he has been engaged in R&D of distributed computing and distributed storage. As a result of organizational changes in July 2012, he is now in NTT Software Innovation Center. He is a member of the Database Society of Japan.

Makoto Onizuka
Distinguished Technical Member, Distributed Computing Technology Project, NTT Software Innovation Center and Visiting Associate Professor at the Graduate School of Information Systems, University of Electro-Communications.
He received the Ph.D. degree in computer science from Tokyo Institute of Technology in 2007. His primary research interests include systems and algorithms for data-intensive cloud computing.