Journal of Business and Economics, ISSN 2155-7950, USA
April 2017, Volume 9, No. 4, pp. 345-349
DOI: 10.15341/jbe(2155-7950)/04.09.2018/005

http://www.academicstar.us

B-tree Construction with Huge Volume of Data on Hadoop

Huynh Cong Viet Ngu
(Department of IT, FPT University, Vietnam)

Abstract: In Socialist Republic of Vietnam, applying the big data to process any kind of data is still a challenge. For example, until now only some corporates have applied big data to develop data warehouse systems which have consistent and invaluable supports to make immediate decisions as well as planning long-term strategies. Nowadays, large amounts of traditional data are still increasing significantly, B-tree is considered as the potential data structure that can manages and organizes this kind of data. However it usually takes a lot of time to construct a B-tree for a huge volume of data, in order to solve the problems that related to collection of large datasets that cannot be processed using traditional computing techniques, big data is considered as optimal solution for scalable processing of this kind of dataset. In this paper, we propose a parallel B-Tree construction scheme based on a Hadoop framework. The proposed scheme divides the data into partitions, builds local B-trees in parallel, and merges them to construct a B-tree that covers the whole data set. While generating the partitions, it considers the data distribution so that each partition has nearly equal amount of data. Therefore the proposed scheme gives an efficient index structure while reducing the construction time.

Key words: B-tree; hadoop, map-reduce; big data; Viet Nam
JEL codes: C55, C8

1. Introduction

Nowadays, from the fact large amounts of traditional data are still increasing significantly, the B-tree that is proposed by Douglas (1979) is considered as an optimal index mechanism that will help retrieve data quickly. A B-tree is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time. The B-tree is a generalization of a binary search tree in that a node can have more than two children. Unlike self-balancing binary search trees, the B-tree is optimized for systems that read and write large blocks of data. B-trees are a good example of a data structure for external memory. It is commonly used in databases and file-systems. The B-tree example is shown in Figure 1.

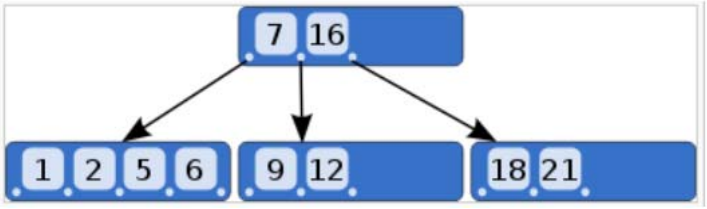

Figure 1 Example of B-tree of Order m = 5

Huynh Cong Viet Ngu, M.E., Department of IT, FPT University; research areas/interests: big data, data structure & algorithm, parallel computing, distributed computing. E-mail: nguhcv@fe.edu.vn.

The B-tree is an index structure that enables fast accesses to one dimensional data. However, it usually takes a lot of time to construct an B-tree for a huge volume of data. In this paper, we propose a parallel B-Tree construction scheme based on a Hadoop framework. The proposed scheme divides the data into partitions, builds local B-trees in parallel, and merges them to construct a B-tree that covers a whole data set. While generating the partitions, it considers the data distribution so that each partition has nearly equal amounts of data. Therefore the proposed scheme gives an efficient index structure while reducing the construction time.

2. Related Works

2.1 Hadoop-MapReduce

A few years ago, to store or process data, most enterprises had a super computer to perform this task. Here data can be stored in an RDBMS such as Oracle Database, MS SQL Server or DB2. But with this approach, when it has to handle huge amounts of data, it faces many difficulties in processing such data through a traditional database server. Facing those difficulties, in 2005, an Open Source Project called Hadoop was released. In order to handle a huge amounts of data, Hadoop runs all applications using the MapReduce algorithm, where the data is processed in the parallel way on different nodes. MapReduce is a programming model suited for parallel computation, it handles parallelism, fault tolerance and other level issues. Furthermore, MapReduce consists of both a map and reduce function which are userdefined. The input data format is specified by the user and the output is a set of <key,value> pairs. As shown in Table 1 the mapper applies user-defined logic on every input key/value pair (k1,v1) and transforms it into a list of intermediate key/value pairs (k2,v2). Then the reducer will apply user-defined logic to all intermediate values (v2) associated with the same k2 and produces a list of final output key/value pair (k3,v3). The data flow of the MapReduce framework is illustrated in Figure 2.

Table 1 Input and Output in MapReduce

	Input	Output
Map	<k1,v1>	List(k2,v2)
Reduce	<k2,list(v2)>	List(k3,v3)

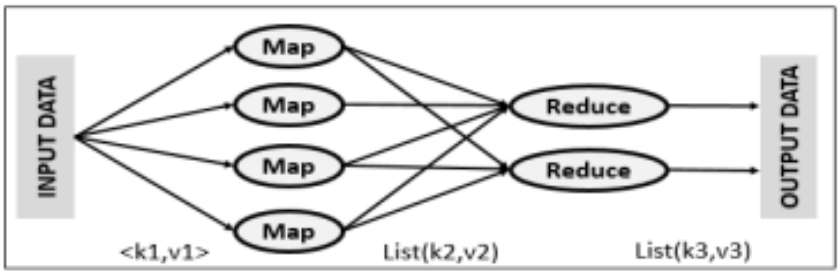

Figure 2 MapReduce Framework

3. Our parallel B-tree Construction Scheme

Our parallel B-tree construction has three phases, and two of them are implemented in the Hadoop environment using the MapReduce model:

- Partitioning phase — Partition boundary of big data set.
- B-tree construction — small B-trees are built concurrently.
- B-tree consolidation — merge small B-trees into the final B-tree.

Firstly, let us start our description by defining the problem. The data set that we are using in this paper is a large CVS file where each line represents one transaction, it contains <o.d, o.t> where o.d is the date of the transaction, it is used as a unique identifier and o.t is the transaction information such as sender ID, transaction type, money, etc.., as show in Table 2.

Table 2 Transaction Log Data

Date	Sender ID	Transaction type	Money	Receiver ID	Sender after transaction amount	Receiver after transaction amount
01/03/2015	15	Transfer	500	16	1000	3255
01/14/2015	52	Deposit	1200	52	1500	15000
02/05/2015	125	Transfer	250	168	2503	4456
04/22/2015	12	Transfer	200	88	1112	2284

Our scheme consists of three phases executed in sequence, as shown in Figure 3. First, we determine the partition boundary based on the date of the transaction. Next, data is partitioned into the corresponding partition which creates small B-Trees. Finally, the small B-Trees are merged into the final B-Tree. The first two phases are executed in MapReduce, while the last phase does not require high computation, so it is executed outside of the cluster.

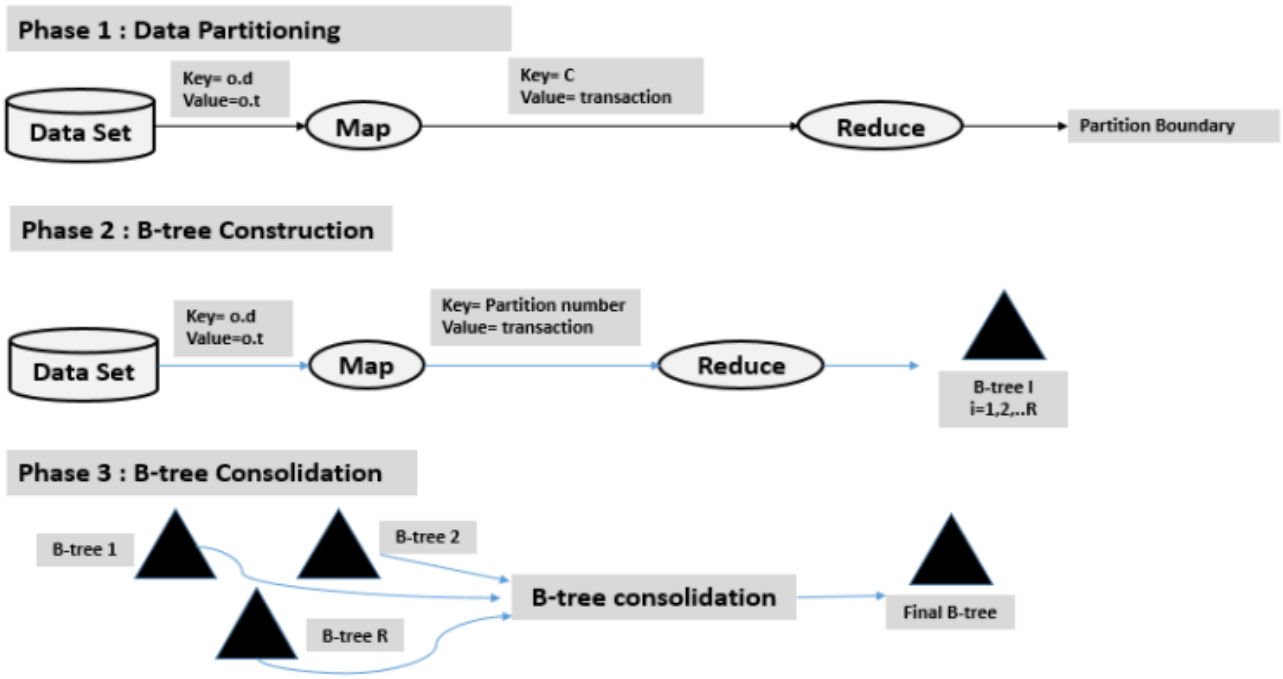

Figure 3 B-tree Construction with MapReduce Framework

3.1 Data Partitioning

In this phase, to determine the partition boundary for data, our idea is to read random objects from the input file via data sampling with a default ratio of input data. The MapReduce algorithm runs M Mappers that take sample objects from the input file, then in each Mapper, it reads transaction, then takes the date of each item. Then a single Reducer, firstly, it sorts the list from Mapper' output, then it determines a new list K of R-1 partition boundary that split the list of sample into R nearly equal-size partition.

The specific MapReduce key/value input pairs are presented in Table 3. Mappers read the default ratio of

data from input file and take the date of each item. The intermediate key is a constant that helps to send all the Mappers' outputs to a single Reducer. Then Reducer determines the list splitting point K base on the date of transaction.

Table 3 Map Reduce Input/Outputs for Data Partitioning

Function	Input	Output (key, value)
Map	(o,d,o,t)	(C, list(transaction date)
Reduce	(C, list(transaction date)	K

3.2 B-tree Construction

In this phase, an individual B-tree is built concurrently. Mappers partition the input file into R groups, then every partition is executed by a different Reducer, each Reducer, B-tree is built independently. The output of every Reducer is a root node of their constructed B-Trees, as shown in Table 4.

Table 4 Map Reduce Input/Outputs for B-tree Construction

Function	Input	Output (key, value)
Map	(o,d,o,t)	(partition number, transactions)
Reduce	(partition number, list(transactions))	B-tree, root

3.3 B-tree Consolidation

In this phase, we are going to combine the R individual B-trees, built in the second phase, under a single root. Because it's not computationally intensive and the logic to run this phase is fairly simple, it is executed outside the cluster as shown in Figure 4.

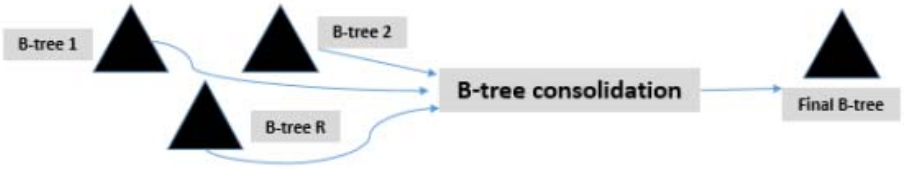

Figure 4 B-tree Consolidation

4. Conclusion and Future Work

In this paper, we proposed a scheme for parallel B-tree construction for Transaction log data on Hadoop environment using the MapReduce model. With our scheme, we hope to contribute to the improvement of the data processing in general, and in particularly, process the transaction log data in Bank system.

Our scheme has three phases, in which, the first two phases are executed in parallel with MapReduce model, while the last phase is executed outside the cluster because it does not require the high computational. Nowadays, with the amount of data is increasing significantly, the availability of Big Data and commodity hardware, has opened many opportunities for analyzing astonishing data sets quickly and cost effectively for the first time in history.

References

Huynh Cong Viet Ngu and Jun-ho Huh (2017). "B+-tree on massive data with Hadoop", *Journal of Networks, Software Tools and Application (Cluster Computing)*, doi: 10.1007/s10586-017-1183-y.

Viet-Ngu Huynh Cong et al. (2017). "Improving the quality of an R-tree using the Map-Reduce framework", *Advanced Multimedia and Ubiquitous Engineering, (CUTE 2016)*, Vol. 448, Springer, Singapore, pp. 164-170.

Viet-Ngu Huynh Cong et al. (2017). “Improving the B+tree construction for transaction log data in bank system using Hadoop”, in: *International Conference on Information Science and Application (ICISA 2017)*, LNEE, Vol. 424, Springer, Singapore, pp. 519-525.

Douglas C. (1979). “The ubiquitous B-tree”, *Comput. Surv.*, Vol. 11, No. 2, pp. 121-137.

Konstantin Shvachko, Hairong Kuang and Sanjay Radia (2010). “The hadoop distributed file system”, *Mass Storage Systems and Technologies (MSST), IEEE 26th Symposium*, 3-7 May 2010.