Market Basket Analysis Algorithm on Map/Reduce in AWS EC2

Jongwook Woo

Computer Information Systems Department
California State University Los Angeles
jwoo5@calstatela.edu

Abstract

As the web, social networking, and smartphone application have been popular, the data has grown drastically everyday. Thus, such data is called Big Data. Google met Big Data earlier than others and recognized the importance of the storage and computation of Big Data. Thus, Google implemented its parallel computing platform with Map/Reduce approach on Google Distributed File Systems (GFS) in order to compute Big Data. Map/Reduce motivates to redesign and convert the existing sequential algorithms to Map/Reduce algorithms for Big Data so that the paper presents Market Basket Analysis algorithm with Map/Reduce, one of popular data mining algorithms. The algorithm is to sort data set and to convert it to (key, value) pair to fit with Map/Reduce. Amazon Web Service (AWS) provides Apache Hadoop platform that provide Map/Reduce computing on Hadoop Distributed File Systems (HDFS) as one of many its services. In the paper, the proposed algorithm is executed on Amazon EC2 Map/Reduce platform with Hadoop. The experimental results show that the code with Map/Reduce increases the performance as adding more nodes but at a certain point, Map/Reduce has the limitation of exploring the parallelism with a bottle-neck that does not allow the performance gain. It is believed that the operations of distributing, aggregating, and reducing data in the nodes of Map/Reduce should cause the bottle-neck.

Keywords: *Big Data, Map/Reduce, Market Basket Analysis, Association Rule, Hadoop, Cloud Computing, AWS EC2*

1. Introduction

People have talked about Cloud Computing that is nothing else but the services we have used for several years such as hosting service, web email service, document sharing service, and map API service etc. Internet and Web enable to achieve the concept of the cloud computing that is virtual computing resources and repository. Therefore, many web applications have been implemented and provided to the users with web emails, web documents, web services, mashups, web data sources, and data feeds etc. Once the user subscribes to the services, it becomes possible to use the services anytime anywhere with his/her computer.

Cloud computing is more enhanced with its scalability, reliability, and quality of services than the existing services. It is categorized into Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). SaaS is to use a service via Internet without installing or maintaining the software, for example, web email services. PaaS is to have a computing or storage service without purchasing hardware or software, for example, hosting services. IaaS is to have utility computing service that is similar to SaaS but to purchase only the amount of time to use the service like AWS [6, 7]. It is not easy to tell the difference from PaaS and IaaS. But, IaaS users can take its control from the services but PaaS users do not have the right to take the control of the platform. Thus, IaaS users can develop or install programs in more flexible environment. AWS provides S3, EC2, and Elastic MapReduce services for Map/Reduce computation as IaaS and SaaS in cloud computing

Before Internet and Web did not exist, we did not have enough data so that it was not easy to analyze people, society, and science etc with the limited volumes of data. Contradicting to the past, after Internet and web, it has been more difficult to analyze data because of its huge volumes, that is, tera- or peta-bytes of data, which is called Big Data. Google faced to the issue when collecting Big Data as the existing file systems were not sufficient to store Big Data efficiently. Besides, the legacy computing power and platforms were not enough to compute Big Data. Thus, Google implemented Google File Systems (GFS) and Map/Reduce parallel computing platform, which Apache Hadoop project is motivated from.

Hadoop is the parallel programming platform built on Hadoop Distributed File Systems (HDFS) for Map/Reduce computation that processes data as (key, value) pairs. Hadoop has been receiving highlights for the enterprise computing because business world always has the big data such as log files for web transactions. Hadoop is useful to process such big data for business intelligence so that it has been used in data mining for past few years. The era of Hadoop means that the legacy algorithms for sequential computing need to be redesigned or converted to Map/Reduce algorithms. Therefore, in this paper, a Market Basket Analysis algorithm in data mining with Map/Reduce is proposed with its experimental result in Elastic Compute Cloud (EC2) ans (Simple Storage Service) S3 of Amazon Web Service (AWS).

In this paper, Section 2 is related work. Section 3 describes Map/Reduce and Hadoop as well as other related projects. Section 4 presents the proposed Map/Reduce algorithm for Market Basket Analysis. Section 5 shows the experimental result. Finally, Section 6 is conclusion.

2. Related Work

Association Rule or Affinity Analysis is the fundamental data mining analysis to find the co-occurrence relationships like purchase behavior of customers. The analysis is legacy in sequential computation and many data mining books illustrate it.

Aster Data has SQL MapReduce framework as a product [9]. Aster provides nPath SQL to process big data stored in the DB. Market Basket Analysis is executed on the framework but it is based on its SQL API with MapReduce Database.

As far as we understand, there is not any other to present Market Basket Analysis algorithms with Map/Reduce. The approach in the paper is to propose the algorithm and to convert data to (key, value) pair and execute the code on Map/Reduce platform.

3. Map/Reduce in Hadoop

Map/Reduce is an algorithm used in Artificial Intelligence as functional programming. It has been received the highlight since re-introduced by Google to solve the problems to analyze huge volumes of data set in distributed computing environment. It is composed of two functions to specify, “Map” and “Reduce”. They are both defined to process data structured in (key, value) pairs.

3.1. Map/Reduce in Parallel Computing

Map/Reduce programming platform is implemented in the Apache Hadoop project that develops open-source software for reliable, scalable, and distributed computing. Hadoop can compose hundreds of nodes that process and compute peta- or tera-bytes of data working together. Hadoop was inspired by Google's MapReduce and GFS as Google has had needs to process huge data set for information retrieval and analysis [1]. It is used by a global community of contributors such as Yahoo, Facebook, and Twitters. Hadoop’s subprojects include Hadoop Common, HDFS, MapReduce, Avro, Chukwa, HBase, Hive, Mahout, Pig, and ZooKeeper etc. [2].

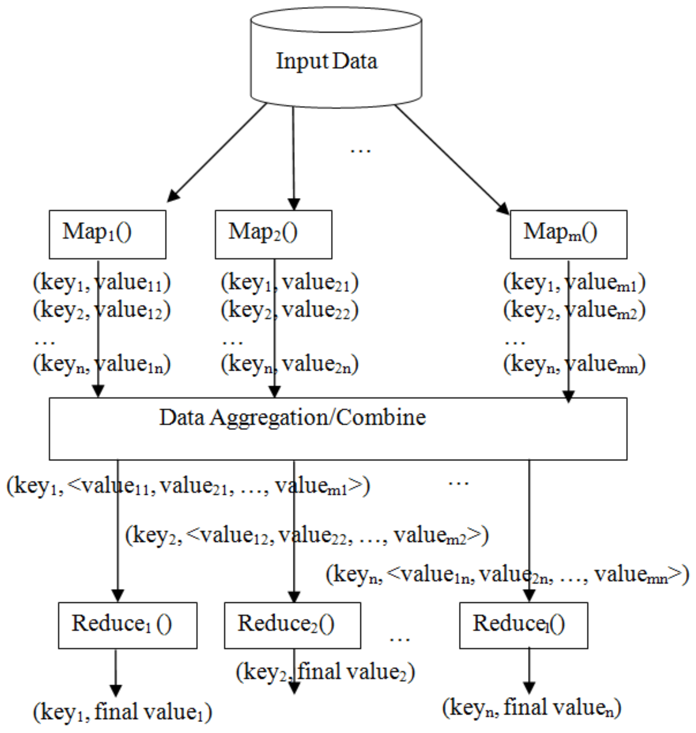

Input Data
Map1()
Map2()
Mapm()
(key1, value11) (key2, value12) … (keyn, value1n)
(key1, value21) (key2, value22) … (keyn, value2n)
(key1, valuem1) (key2, valuem2) … (keyn, valuemn)
Data Aggregation/Combine
(key1, <value11, value21, …, valuem1>)
(key2, <value12, value22, …, valuem2>)
(keyn, <value1n, value2n, …, valuemn>)
Reduce1 ()
Reduce2()
Reducel()
(key1, final value1)
(key2, final value2)
(keyn, final valuen)

Figure 3.1. Map/Reduce Flows

The map and reduce functions run on distributed nodes in parallel. Each map operation can be processed independently on each node and all the operations can be performed in parallel. But in practice, it is limited by the data source and/or the number of CPUs near that data. The reduce functions are in the similar situation because they are from all the output of the map operations. However, Map/Reduce can handle significantly huge data sets since data are distributed on HDFS and operations move close to data for better performance [5].

Hadoop is restricted or partial parallel programming platform because it needs to collect data of (key, value) pairs as input and parallely computes and generates the list of (key, value) as output on map/reduce functions. In map function, the master node parts the input into smaller sub-problems, and distributes those to worker nodes. Those worker nodes process smaller problems, and pass the answers back to their master node. That is, map function takes inputs (k1, v1) and generates <k2, v2> where < > represents list or set. Between map and reduce, there is a combiner that resides on map node, which takes inputs (k2, <v2>) and generates <k2, v2>. In Figure 3.1, a list of values is collected for a key as (keyn, <value1n, value2n, …, valuemn>) from mappers.

In reduce function, the master node takes the answers to all the sub-problems and combines them in some way to get the output, the answer to the problem [1, 2]. That is, reduce function takes inputs (k2, <v2>) and generates <k3, v3>. Figure 3.1 illustrates

Map/Reduce control flow where each valuemn is simply 1 and gets accumulated for the occurrence of items together in the proposed Market Basket Analysis Algorithm. Thus, for each key, the final value is the total number of values, that is the sum of 1s, as (keyn, final valuen)

3.2. The Issues of Map/Reduce

Although there are advantages of Map/Reduce, for some researchers and educators, it is:

1. Need tens-, hundreds-, or thousands-of-nodes to compose Hadoop Map/Reduce platform.

2. If using services of cloud computing, for example, AWS EC2, the overheads mainly come from I/O. That is, it takes long to upload big data to AWS EC2 platform or AWS S3, which is more than computing time.

3. A giant step backward in the programming paradigm for large-scale data intensive applications

4. Not new at all - it represents a specific implementation of well known techniques developed tens of years ago, especially in Artificial Intelligence

5. Data should be converted to the format of (key, value) pair for Map/Reduce, which misses most of the features that are routinely included in current DBMS

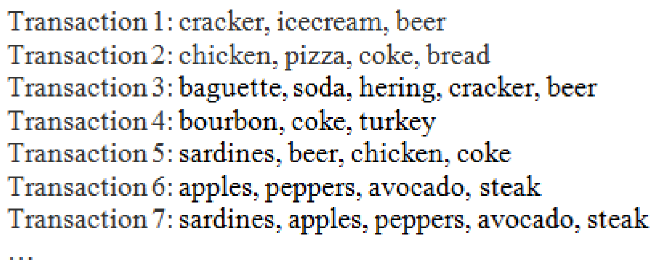
6. Incompatible with all of the tools or algorithms that have been built [4].

However, the issues clearly show us not only the problems but also the opportunity where we can implement algorithms with Map/Reduce approach, especially for big data set. It will give us the chance to develop new systems and evolve IT in parallel computing environment. It started a few years ago and many IT departments of companies have been moving to Map/Reduce approach in the states.

4. Market Basket Analysis Algorithm

Market Basket Analysis is one of the Data Mining approaches to analyze the association of data set. The basic idea is to find the associated pairs of items in a store when there are transaction data sets as in Figure 4.1.

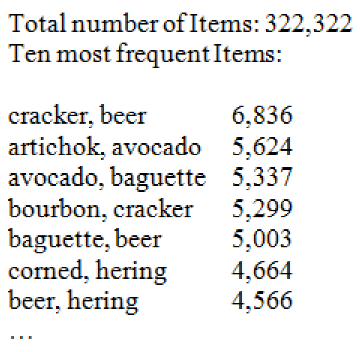
If store owners list a pair of items that are frequently occurred, s/he could control the stocks more intelligently, to arrange items on shelves and to promote items together etc. Thus, s/he should have much better opportunity to make a profit by controlling the order of products and marketing.



Transaction 1: cracker, icecream, beer
Transaction 2: chicken, pizza, coke, bread
Transaction 3: baguette, soda, hering, cracker, beer
Transaction 4: bourbon, coke, turkey
Transaction 5: sardines, beer, chicken, coke
Transaction 6: apples, peppers, avocado, steak
Transaction 7: sardines, apples, peppers, avocado, steak
…

Figure 4.1 Transaction Data at a Store

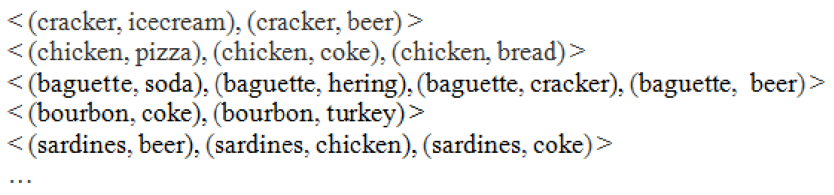
For example, people have built and run Market Basket Analysis codes – sequential codes - that compute the top 10 frequently occurred pair of transactions as in Figure 4.2. At the store, when customers buy a cracker, they purchase a beer as well, which happens 6,836 times and bourbon as well in 5,299 times. Thus, the owner can refer to the data to run the store.



Total number of Items: 322,322
Ten most frequent Items:

cracker, beer	6,836
artichok, avocado	5,624
avocado, baguette	5,337
bourbon, cracker	5,299
baguette, beer	5,003
corned, hering	4,664
beer, hering	4,566
…	

Figure 4.2 Top 10 Pair of Items Frequently Occurred at Store



< (cracker, icecream), (cracker, beer) >
< (chicken, pizza), (chicken, coke), (chicken, bread) >
< (baguette, soda), (baguette, hering), (baguette, cracker), (baguette, beer) >
< (bourbon, coke), (bourbon, turkey) >
< (sardines, beer), (sardines, chicken), (sardines, coke) >
…

Figure 4.3 Data Set Restructured for Map/Reduce

4.1. Data Structure and Conversion

The data in Figure 4.1 is composed of the list of transactions with its transaction number and the list of products. For Map/Reduce operation, the data set should be structured with (key, value) pairs. The simplest way used in the paper is to pair the items as a key and the number of key occurrences as its value in the basket, especially for all transactions, without the transaction numbers. Thus, Figure 4.1 can be restructured as Figure 4.3 assuming collecting a pairs of items in order 2 – two items as a key.

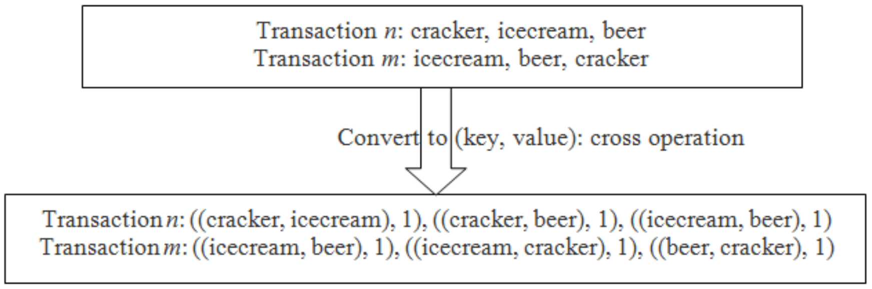

Transaction *n*: cracker, icecream, beer
Transaction *m*: icecream, beer, cracker
Convert to (key, value): cross operation
Transaction *n*: ((cracker, icecream), 1), ((cracker, beer), 1), ((icecream, beer), 1)
Transaction *m*: ((icecream, beer), 1), ((icecream, cracker), 1), ((beer, cracker), 1)

Figure 4.4 Data Set Restructured for the Same List

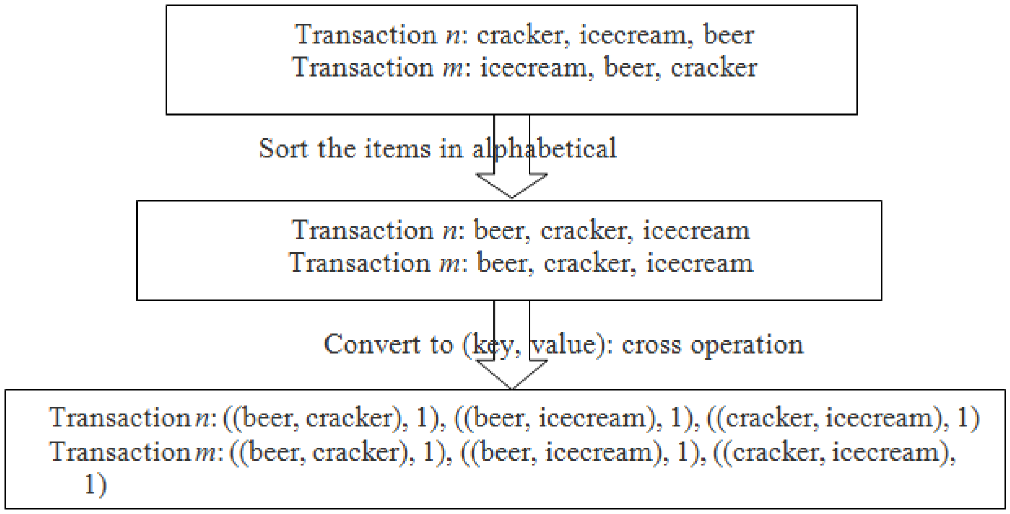

Transaction *n*: cracker, icecream, beer
Transaction *m*: icecream, beer, cracker
Sort the items in alphabetical
Transaction *n*: beer, cracker, icecream
Transaction *m*: beer, cracker, icecream
Convert to (key, value): cross operation
Transaction *n*: ((beer, cracker), 1), ((beer, icecream), 1), ((cracker, icecream), 1)
Transaction *m*: ((beer, cracker), 1), ((beer, icecream), 1), ((cracker, icecream), 1)

Figure 4.5 (a) Data Set Restructured with Sort in Order 2

However, if we select the two items in a basket as a key, there should be incorrect counting for the occurrence of the items in the pairs. As shown in Figure 4.4, transactions n and m have the items (cracker, icecream, beer) and (icecream, beer, cracker), which have the same items but in different order.

That is, for (cracker, icecream, beer), the possible pair of items in (key, value) are ((cracker, icecream), 1), ((cracker, beer), 1), ((icecream, beer), 1). And, for (icecream, beer, cracker), the possible pair of items are ((icecream, beer), 1), ((icecream, cracker), 1), ((beer, cracker), 1).

Therefore, we have total SIX different pair of items that occurs only once respectively, which should be THREE different pairs. That is, keys (cracker, icecream) and (icecream, cracker) are not same even though they are, which is not correct.

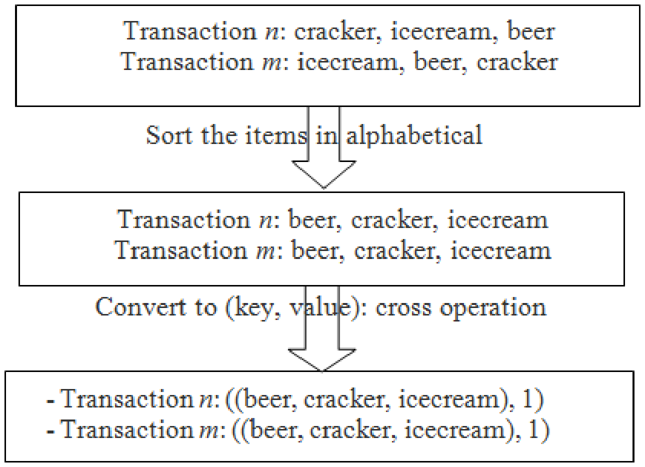

Transaction *n*: cracker, icecream, beer
Transaction *m*: icecream, beer, cracker
Sort the items in alphabetical
Transaction *n*: beer, cracker, icecream
Transaction *m*: beer, cracker, icecream
Convert to (key, value): cross operation
- Transaction *n*: ((beer, cracker, icecream), 1)
- Transaction *m*: ((beer, cracker, icecream), 1)

Figure 4.5 (b) Data Set Restructured with Sort in Order 3

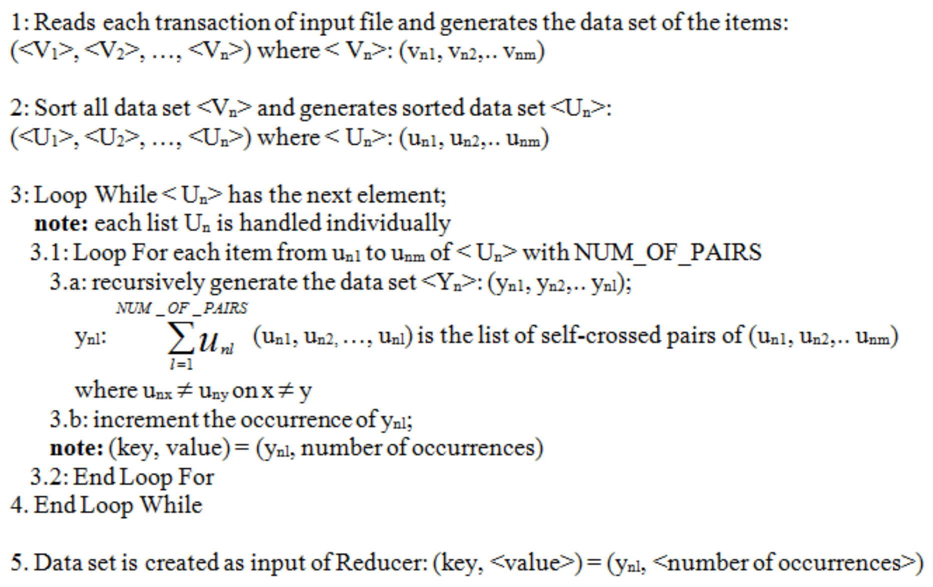
We can avoid this issue if we sort the transaction in alphabetical order before generating (key, value) as shown in Figure 4.5 (a). Now each transaction have the following THREE pair of items ((beer, cracker), 1), ((beer, icecream), 1), ((cracker, icecream), 1). That is TWO different pair of items, key in order 2, that occurs twice respectively so that we accumulate the value of the occurrence for these two transactions as follows: ((beer, cracker), 2), ((beer, icecream), 2), ((cracker, icecream), 2), which is correct to count the total number of occurrences. We call THREE different pair of items as key in order 3 which has the following (key, value) pair: ((beer, cracker, icecream), 2) as shown in Figure 4.5 (b).

4.2. The algorithm

The Market Basket Analysis (MBA) Algorithms for Mapper and Reducer are illustrated in Figures 4.6 and 4.7 respectively. Mapper reads the input data and creates a list of items for each transaction. As a mapper of a node reads each transaction on Hadoop, it assigns mappers to number of nodes, where the assigning operation in Hadoop is hidden to us. For each transaction, its time complexity is O(n) where n is the number of items for a transaction.

Then, the items in the list are sorted to avoid the duplicated keys as shown in Figures 4.4 and 4.5 (a). Its time complexity is O(n log n) on merge sort. Then, the sorted items should be converted to pairs of items as keys, which is a cross operation in order to generate cross pairs of the items in order 2 list as shown in Figures 4.4 and 4.5 (a). Its time complexity is O(n x m) where m is the number of items that compose a pair together in the transaction. Thus, the time complexity of each mapper is O(n + n log n + n x m). For the items with key in order 3 as shown in Figures 4.4 and 4.5 (b), its time complexity is O(n x m x l) where m is the number of items remained to compose a key pair and l is the number of items remained to compose a triple key pair in the transaction. Thus, the time complexity of each mapper is O(n + n log n + n x m x l) for the items with the key in order 3.

The reducer is to accumulate the number of values per key. Thus, its time complexity is O(v) where v is the number of values per key.



```
1: Reads each transaction of input file and generates the data set of the items:
(<$V_1$>, <$V_2$>, …, <$V_n$>) where < $V_n$>: ($v_{n1}$, $v_{n2}$,.. $v_{nm}$)

2: Sort all data set <$V_n$> and generates sorted data set <$U_n$>:
(<$U_1$>, <$U_2$>, …, <$U_n$>) where < $U_n$>: ($u_{n1}$, $u_{n2}$,.. $u_{nm}$)

3: Loop While < $U_n$> has the next element;
   note: each list $U_n$ is handled individually
   3.1: Loop For each item from $u_{n1}$ to $u_{nm}$ of < $U_n$> with NUM_OF_PAIRS
      3.a: recursively generate the data set <$Y_n$>: ($y_{n1}$, $y_{n2}$,.. $y_{nl}$);
         $y_{nl}$:   $\sum_{l=1}^{NUM\_OF\_PAIRS} u_{nl}$  ($u_{n1}$, $u_{n2}$, …, $u_{nl}$) is the list of self-crossed pairs of ($u_{n1}$, $u_{n2}$,.. $u_{nm}$)
         where $u_{nx} \neq u_{ny}$ on $x \neq y$
      3.b: increment the occurrence of $y_{nl}$;
      note: (key, value) = ($y_{nl}$, number of occurrences)
   3.2: End Loop For
4. End Loop While

5. Data set is created as input of Reducer: (key, <value>) = ($y_{nl}$, <number of occurrences>)
```

Figure 4.6. MBA Algorithm for Mapper

```
1: Read ($y_{nl}$, <number of occurrences>) data from multiple nodes
2. Add the values for $y_{nl}$ to have ($y_{nl}$, total number of occurrences)
```

Figure 4.7. MBA Algorithm for Reducer

4.3. The Code

The *ItemCount.java* code is implemented on Hadoop 0.20.2 and 0.21.0 and executable on stand-alone and clustered modes. The code generates the top 10 associated items that customers purchased together as shown in Figure 4.2. Anyone can download the files to test it, which takes the sample input "*AssociationsSP.txt*" as introduced in the blog [8]. The sample input has 1,000 transactions with data as shown in Figure 4.1. But, the experimental result in section 5 is generated with much bigger data on multiple nodes of AWS EC2 not on a single node.

Figure and table captions should be 11-point Helvetica boldface (or a similar sans-serif font). Callouts should be 10-point Helvetica, non-boldface. Initially capitalize only the first word of each figure caption and table title. Figures and tables must be numbered separately. For example: "*Figure 1. Database contexts*", "*Table 1. Input data*". Figure captions are to be below the figures. Table titles are to be centered above the tables.

5. Experimental Result

We have 4 transaction files for the experiment: 200 MB (3.4M transactions), 400 MB (6.7M transactions), 800MB (13M transactions), 1.6 GB (26M transactions). Those are run on extra large instances of AWS EC2 which allows to instantiate number of nodes requested, where each node is of 1.0-1.2 GHz 2007 Opteron or Xeon Processor, 15GB memory, 1,690GB instance storage on 64 bits platform, and high I/O performance. The data are executed on 1, 2, 5, 10, 15, and 20 nodes respectively and its execution times are shown in Table 5.1 (a, b). For 13 and 26 Mega transactions of node 1, it took too long to measure the execution times so that we do not execute them and its times are Not Applicable (NA) in the Table 5.1 (a, b).

Table 5.1 (a). Execution Time (msec) in Order 2

	3.4M (200M)	**6.7M (400MB)**	**13M (800MB)**	**26M (1600M)**
1	1,600,339.00	3,124,972.00		
2	1,099,115.00	1,885,996.00	3,471,285.00	5,632,922.00
5	986,255.00	1,091,723.00	1,325,612.00	2,529,631.00
10	978,587.00	986,329.00	1,102,558.00	1,290,564.00
15	1,003,506.00	1,060,552.00	1,028,739.00	1,172,775.00
20	990,033.00	1,045,166.00	1,028,399.00	1,131,514.00

Table 5.1 (b). Execution Time (msec) in Oder 3

	3.4M (200MB)	6.7M (400MB)	13M (800MB)	26M (1600MB)
1	2,672,560.00	5,304,069.00		
2	1,862,665.00	3,194,292.00	5,981,474.00	9,640,743.00
5	1,661,617.00	1,883,061.00	2,263,369.00	4,273,803.00
10	1,678,787.00	1,653,237.00	1,903,042.00	2,273,986.00
15	1,644,821.00	1,668,803.00	1,673,344.00	1,952,917.00
20	1,727,036.00	1,655,310.00	1,736,326.00	1,899,179.00

The output of the computation in Table 5.2 (a) presents the number of items (total: 1.255G) and keys (total: 212) that are associated pair of items in orders 2, especially for data of 26M transactions in file size 1.6GB. And, the 10 most frequently occurred items and its frequency, which is (key, value), are shown. Similarly, Table 5.2 (b) presents the same number of items but different keys (total: 1,318) that are associated pair of items in order 3.

Figure 5.1 (a) based on Table 5.1 (a) is the chart of the experimental result with 200 MB (1.4 transactions), 400 MB (6.7M transactions), 800MB (13M transactions), 1.6 GB (1,600 MB 26M transactions). The more the nodes are, the faster the computation times are. Since the algorithm is simply to sort the data set and then convert it to (key, value) pairs, the linear result is expected. The performance is linearly increased by certain number of nodes for some transaction data sets but from a certain point, it has the limitation so that it does not have any performance improvement.

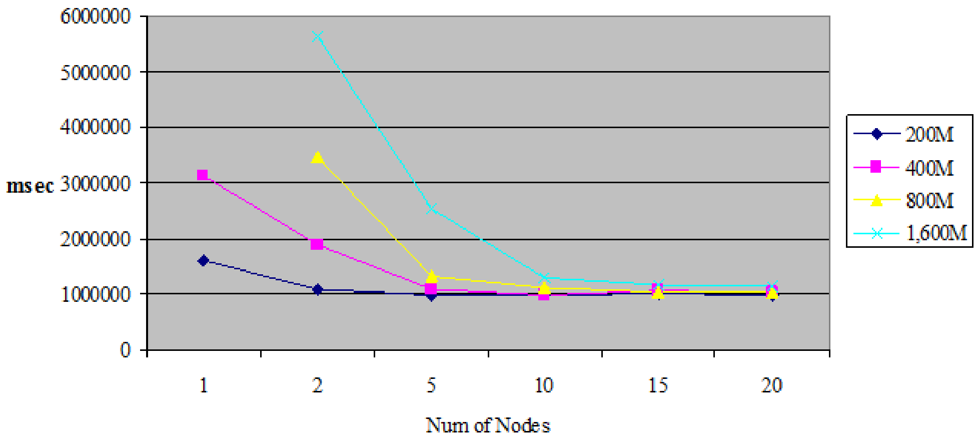


Figure 5.1 (a). Chart for Execution Time in Oder 2

To find the associated items in order 2, for 200MB file, the performance is not improved after executing it on more than 2 nodes. For 400MB file, there is not much difference among nodes 5, 10, 15 and 20. Similarly, for 800MB and 1.6GB files, there are not many differences on nodes 10, 15 and 20. To find the associated items in order 3, for 200MB file, the performance is not improved after executing it on 5 nodes. For 400MB file, there is not much difference among nodes 10, 15 and 20. Similarly, for 800MB and 1.6GB files, there are not many differences between nodes 15 and 20. There is bottleneck in EC2 x-large instance, which shows that there is a trade-off between the number of nodes and the operations of distributing transactions data to nodes, aggregating the data, and reducing the output data for each key so that it should not have much performance gain even though adding mode nodes for faster parallel computation.

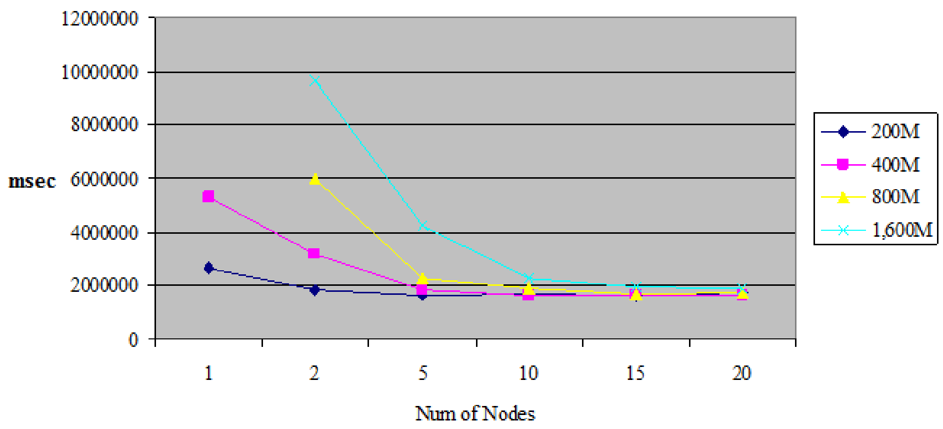
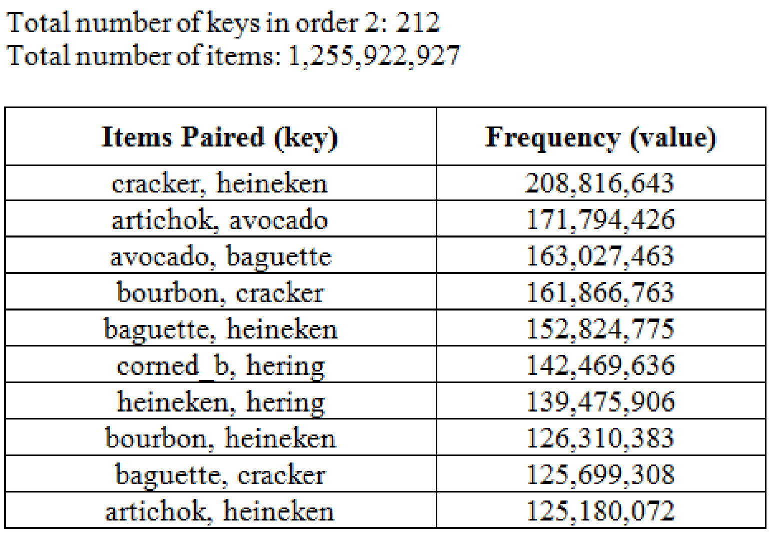


Figure 5.1 (b). Chart for Execution Time in Oder 3

Table 5.2 (a). 10 Most Frequently Associated Items on 1.6GB 26M Transactions in Order 2



Total number of keys in order 2: 212
Total number of items: 1,255,922,927

Items Paired (key)	Frequency (value)
cracker, heineken	208,816,643
artichok, avocado	171,794,426
avocado, baguette	163,027,463
bourbon, cracker	161,866,763
baguette, heineken	152,824,775
corned_b, hering	142,469,636
heineken, hering	139,475,906
bourbon, heineken	126,310,383
baguette, cracker	125,699,308
artichok, heineken	125,180,072

Table 5.2 (b). 10 Most Frequently Associated Items on 1.6GB 26M Transactions in Order 3

Total number of keys in order 3: 1,318
Total number of items: 1,255,922,927

Items Paired (key)	Frequency (value)
baguette, heineken, hering	9,146,936
corned_b, hering, olives	8,953,427
artichok, avocado, heineken	8,791,531
bourbon, cracker, heineken	8,531,341
baguette, cracker, heineken	8,499,540
cracker, heineken, hering	7,903,994
apples, avocado, baguette	7,895,321
artichok, avocado, cracker	7,843,283
corned_b, ham, hering	7,123,424
chicken, coke, heineken	7,036,694

In summary, the experimental data illustrates that even though the number of nodes are added, at a certain point, there is a bottleneck that cannot increase the performance because the more the nodes are added, the more the time is spent to distribute, aggregate, and reduce the data to Map/Reduce nodes.

5.1. Future Work with Database for Big Data

Input/Output files are processed on HDFS instead of using HBase DB in the paper, which does not have any benefit we can get when using DB. However, as HBase that is (key, value) DB executed on HDFS so that it is better integrated with the algorithm in the future, the section briefly introduces HBase. There are some drawbacks when we use RDBMS to handle huge volumes of data, like impossible deleting, slow inserting, and random failing. HBase on HDFS is distributed database that supports structured data storage for horizontally scalable tables. It is column oriented semi-structured data store.

It is relatively easy to integrate with Hadoop Map/Reduce because HBase consists of a core map that is composed of keys and values - each key is associated with a value. Users store data rows in labeled tables. A data row has a sortable key and an arbitrary number of columns. The table is stored sparsely, so that rows in the same table can have different columns.

Using the legacy programming languages such as Java, PHP, and Ruby, we can put data in the map as Java JDBC does for RDBMS. The file storage of HBase can be distributed over an array of independent nodes because it is built on HDFS. Data is replicated across some participating nodes. When the table is created, the table's column families are generated at the same time. We can retrieve data from HBase with the full column name in a certain form. And then HBase returns the result according to the given queries as SQL does in RDBMS [10].

6. Conclusion

Hadoop with Map/Reduce motivates the needs to propose new algorithms for the existing applications that have had algorithms for sequential computation. Besides, it is (key, value) based restricted parallel computing so that the legacy parallel algorithms need to be redesigned with Map/Reduce.

In the paper, the Market Basket Analysis Algorithm on Map/Reduce is presented, which is association based data mining analysis to find the most frequently occurred pair of products in baskets at a store. The data set of the experimental result shows that associated items can be paired in orders 2 and 3 with Map/Reduce approach. Once we have the associated items, it can be used for more studies by statically analyzing them even sequentially, which is beyond this paper.

The algorithm has been executed on EC2 x-large instances of AWS with nodes 1, 2, 5, 10, 15, and 20. The execution times of the experiments show that the proposed algorithm gets better performance while running on larger number of nodes to a certain point. However, from a certain number of nodes, Map/Reduce in Hadoop does not guarantee to increase the performance even though we add more nodes because there is a bottle-neck to enhance the parallelism, which is negatively affected by the time spent for distributing, aggregating, and reducing the data set among nodes against computing powers of additional nodes.

References

[1] J. Dean and S. Ghemawa, “MapReduce: Simplified Data Processing on Large Clusters”, Google Labs, OSDI 2004, **(2004)**, pp. 137–150.
[2] Apache Hadoop Project, http://hadoop.apache.org/.
[3] B. Stephens, “Building a business on an open source distributed computing”, Oreilly Open Source Convention (OSCON) 2009, **(2009)** July 20-24, San Jose, CA
[4] W. Kim, “MapReduce Debates and Schema-Free”, Coord, **(2010)** March 3.
[5] J. Lin and C. Dyer, “Data-Intensive Text Processing with MapReduce”, Tutorial at the 11th Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT 2010), **(2010)** June, Los Angeles, California
[6] J. Woo, “Introduction to Cloud Computing”, the 10th KOCSEA 2009 Symposium, UNLV, **(2009)** December 18-19.
[7] J. Woo, “The Technical Demand of Cloud Computing”, Korean Technical Report of KISTI (Korea Institute of Science and Technical Information), **(2011)** February.
[8] J. Woo, “Market Basket Analysis Example in Hadoop”, http://dal-cloudcomputing.blogspot.com/2011/03/market-basket-analysis-example-in.html, **(2011)** March.
[9] Aster Data, “SQL MapReduce framework”, http://www.asterdata.com/product/advanced-analytics.php.
[10] Apache HBase, http://hbase.apache.org/.
[11] J. Lin and C. Dyer, “Data-Intensive Text Processing with MapReduce”, Morgan & Claypool Publishers, **(2010)**.
[12] GNU Coord, http://www.coordguru.com/.
[13] J. Woo, D. -Y. Kim, W. Cho and M. Jang, “Integrated Information Systems Architecture in e-Business”, The 2007 international Conference on e-Learning, e-Business, Enterprise Information Systems, e-Government, and Outsourcing, Las Vegas, **(2007)** June 26-29.

Author

Jongwook Woo

Jongwook Woo is currently an Associate Professor at Computer Information Systems Department of California State University, Los Angeles. He received the BS and the MS degree, both in Electronic Engineering from Yonsei University in 1989 and 1991, respectively. He obtained his second MS degree in Computer Science and received the PhD degree in Computer Engineering, both from University of Southern California in 1998 and 2001, respectively. His research interests are Information Retrieval/Integration/Sharing on Big Data, Map/Reduce algorithm on Hadoop Parallel/Distributed/Cloud Computing, and n-Tier Architecture application in e-Business, smartphone, social networking and bioinformatics applications. He received teaching/research grants from Amazon AWS and owns Academic Partnership with Cloudera in Big Data. He has published more than 40 peer reviewed conference and journal papers. He also has consulted many entertainment companies in Hollywood.