JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 18, No. 8, pp. 1593-1601, Dec. 2017

# 빅데이터 플랫폼을 위한 SON알고리즘 기반의 효과적인 연관 룰 마이닝

뉘엔양쯔엉 · 뉘엔반퀴엣 · 뉘엔신응억 · 김경백*
전자컴퓨터공학부, 전남대학과

# Efficient Association Rule Mining based SON Algorithm for a Bigdata Platform

**Giang-Truong Nguyen · Van-Quyet Nguyen · Sinh-Ngoc Nguyen · Kyungbaek Kim***
Department of Electronics and Computer Engineering, Chonnam National University

**[요 약]**

빅데이터 플랫폼에서, 연관 룰 마이닝 응용프로그램은 여러 가치를 창출할 수 있다. 예를 들어, 농업 빅데이터 플랫폼에서 농가 소득을 높일 수 있는 농작물들을 농업인들에게 추천할 수 있다. 이 연관 룰 마이닝의 주요 절차는 빈발 아이템셋 마이닝으로, 이는 동시에 나타나는 아이템의 셋을 찾는 작업이다. Apriori를 비롯한 이전 연구에서는 대규모의 가능한 아이템 셋에 의한 메모리 오버로드의 이유로 만족할 만한 성능을 보일 수 없었다. 이를 개선하고자, 아이템 셋을 작은 크기로 분할하여 순차적으로 계산하도록 하는 SON 알고리즘이 제안되었다. 하지만, 단일 머신에서 SON 알고리즘을 돌릴 경우 많은 시간이 소요된다. 이 논문에서는 하둡 기반의 빅데이터 플랫폼에서 SON 알고리즘 병렬처리 방식을 이용한 연관룰 탐색 기법을 소개한다. 연관 룰 마이닝을 위한 전처리, SON 알고리즘 기반 빈발 아이템셋 마이닝, 그리고 연관룰 검출 절차를 Hadoop기반의 빅데이터 플랫폼에 구현하였다. 실제 데이터를 활용한 실험을 통해 제안된 연관 룰 마이닝 기법은 Brute Force 기법의 성능을 압도하는 것을 확인하였다.

**[Abstract]**

In a big data platform, association rule mining applications could bring some benefits. For instance, in a agricultural big data platform, the association rule mining application could recommend specific products for farmers to grow, which could increase income. The key process of the association rule mining is the frequent itemsets mining, which finds sets of products accompanying together frequently. Former researches about this issue, e.g. Apriori, are not satisfying enough because huge possible sets can cause memory to be overloaded. In order to deal with it, SON algorithm has been proposed, which divides the considered set into many smaller ones and handles them sequently. But in a single machine, SON algorithm cause heavy time consuming. In this paper, we present a method to find association rules in our Hadoop based big data platform, by parallelling SON algorithm. The entire process of association rule mining including pre-processing, SON algorithm based frequent itemset mining, and association rule finding is implemented on Hadoop based big data platform. Through the experiment with real dataset, it is conformed that the proposed method outperforms a brute force method.

**색인어 :** 빅데이터 플랫폼, 연관 룰 마이닝, 빈발 아이템셋, SON 알고리즘
**Key word :** Big data Platform, Association Rule Mining, Frequent Itemsets, SON Algorithm

**http://dx.doi.org/10.9728/dcs.2017.18.8.1593**



**Received** 21 September 2017; **Revised** 21 September 2017
**Accepted** 25 December 2017

***Corresponding Author: Kyungbaek Kim**

**Tel:** +82-62-530-3438
**E-mail:** kyungbaekkim@jnu.ac.kr

## I. Introduction

As being a farmer, choosing which products should be grown in a season is a very crucial question, because it is a factor for manipulating their income and productivity. Usually, each farmer would choose a main product which they think it would be most suitable for them. However, in order to get most benefits, they should also choose other types to grow with their chosen ones [1]. This is a big question because among many products, finding the suitable one requires a lot of conditions. Beside the best but too much time-consuming and experience-required scientific researches methods in Biology [1][2], one of the solutions is finding what other more-experience farmers often grow together. Following the footstep of those seniors, from some already chosen products, farmers could choose other suitable ones to combine for their season. This solution could be conducted by a well-known method called association rules mining.

Mining the association rules has received attentions from many researchers for a long time. The original work of this technique is from a work of many retailers like Walmart, Amazon: giving a collection of transactions and their corresponding purchased items, mining the association rules finds items (consequent) which customers could possibly grab after taking (some) one(s) (antecedent). In order to implement this mining, the prior work should be handled is finding frequent itemsets [3]: trying to get all the set of items whose rate of appearance over the total number of transaction (a.k.a. their support) is larger than a given threshold. Therefore, coming back to the agricultural solution as mentioned above: considering a given collection of farms as transactions; and their corresponding grown products as items, mining the association rules could be used to suggest the farmers to choose which products should be grown with some given ones, which partly supports those people to improve their productivity and income.

The problem of finding frequent itemsets could be solved by many algorithms up to now, but most of them is used for only the single-machine environment only. The first and original method called Brute-Force algorithm (BFA), which lists all the possible set of items; then finds their number of appearance in each transaction, is too “naive” because it uses too much time and memory. A-Priori algorithm [4] follows the idea of the BFA, however it finds the frequent itemsets in each levels of their increasing length, then in every level; it prunes the unsatisfied sets to reduce the number of computations for the next phase. Unfortunately, this algorithm could have some problems if the number of transactions and number of items are too large, causing the single-machine to be not able to handle loading all the data from hard dish to the main memory. Avoiding this problem, an improvement made by Savarese et al. called SON algorithm [5] has been proposed, which divides the collection of transactions into non-overlapping smaller ones to avoid the problem of overloading memory which a single machine could face.

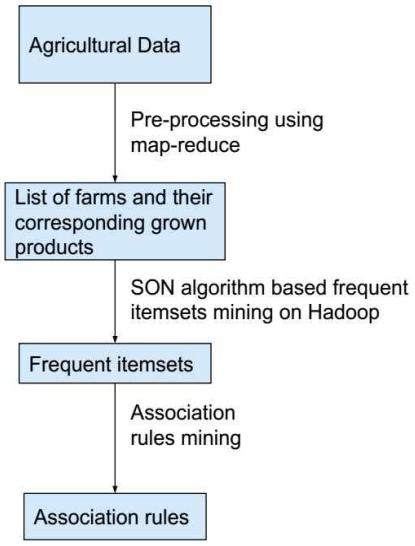


**Fig. 1.** Overall process for agriculture association rules mining

In this paper, we propose a method for mining the association rules, whose prior work is finding the frequent itemsets based on the idea of SON algorithm dividing the big collection of transactions for some machines, to process on them. However, our method is not conducted in a single machine, but in our implemented Hadoop [12] based distributed big data platform [8][9][10]. The overall process could be described in Fig. 1. First of all, from the collected agriculture data about many farms in Korea, by a pre-processing phase, the list of farms with their corresponding grown products would be gotten. Then by those so-called “transactions”, the frequent itemsets would be found. These two above phases are made with the support of MapReduce function [11]. Finally, on those results, another algorithm based on a former research is utilized to find the association rules about the grown agricultural products.

The rest of this paper is organized as follow: section 2 overviews the background supporting for mining association rules; which includes A-Priori algorithm, SON algorithm and association rules mining method, section 3 describes our big data platform for storing and processing the Korean agricultural data. Section 4 mentions about our method to mine the association rules from the agriculture big data. Our evaluation

is mentioned in section 5 and finally; section 6 is our conclusion.

## II. Background

Given a set of items $I = \{I_1, I_2 \cdots, I_n\}$ and the collection of transactions listing all the item which each user purchased at the same time, finding the frequent itemsets is trying to get all the set of items, whose rate of appearance over the total number of transaction (their support) is larger than a given threshold. As being mentioned before, the most basic and naive method BFA lists all the possible set of items, then finds their supports in each of transaction; which will be a real catastrophe because the number of transactions and items are too big. A-Priori and SON algorithm are proposed to reduce the number of computation needed to be taken.

***Algorithm 1.*** A-Priori algorithm
**Input**: *Set of items I = {I1, I2, ..., In}; collection of transaction T= { Ti , Ti c I }; threshold minimum support minsup.*
**Output**: *Set of items whose support is larger than minimum support*

```
1.  initialize returning_Set = NULL.
2.  considered_Set <items [], support> ← Getting all single
    items and their support
3.  while(considered_Set is not null)
4.     temporary_Set<items>
5.     for i ← 0 to considered_Set.length()
6.        item_Set = considered_Set[i]
7.        if(support(item_Set) > minsup)
8.           returning_Set.add(item_Set)
9.           temporary_Set.add(item_Set)
10.       considered_Set ← get new sets whose number of items
    is 1 larger than the previous one from temporary_Set.
11. return returning_Set
```

### *2-1 A-Priori algorithm*

Being quite similar to the basic algorithm, A-Priori algorithm also generates all the possible sets of items. However, it solves the problem with an enhancement: it prunes many unsatisfied sets for reducing the number of possible generated ones. A-Priori algorithm is described in Algorithm 1.

As seen in the algorithm, when the number of transactions and possible items are too large, this algorithm could make the main memory to be overloaded. Considering a case when there are m items, so there could be $2^m$ possible transactions, causing the computation complexity is $O(2^m)$. Consequently, an alternative algorithm should be proposed in order to reduce the data loaded into the main memory.

***Algorithm 2.*** SON algorithm
**Input**: *Set of items I = {I1, I2, ..., In}; collection of transaction T= { ti , ti c I }; threshold minimum support minsup.*
**Output**: *Set of items whose support is larger than minimum support*

```
1.  Dividing T into m parts
2.  Initialize local_sets ← null; global_sets ← null

//first passing
3.  for i ← 0 to m
4.     local_set←get_frequent_itemset(i)
5.     local_sets.add(local_set)
6.  for i ← 0 to local_sets.length()
7.     current_set ← local_sets[i]
8.     current_set.quantity ← get_all_quantity(current_set)

//second passing
9.  for i ← 0 to local_sets.length()
10.    current_set ← local_sets[i]
11.    support ← current_set.quantity() / T.length()
12.    if (support > minsup )
13.       global_sets.add(current_set)
14. return global_sets
```

### *2-2 SON algorithm*

SON is a kind of "divide to conquer" algorithm [6]: it first splits the overall collection of transactions into smaller non-overlapping parts. Afterwards, 2 passing of transactions collection are employed to solve the problem. In the first passing, a "local" frequent itemsets finding algorithm will be executed on each small part with a minimum supply threshold being much smaller than the overall one. The output of this phase is all of each smaller part's frequent itemset, which then will be aggravated to become the global frequent itemset candidates. After that, in order to get their number of appearances, a second pass will be executed on each of smaller part again. Finally, the total number of appearances of each global frequent candidate will be gotten and compared with the overall minimum threshold to get the final results. SON algorithm is described in Algorithm 2.

### *2-3 Mining the association rules*

After getting the frequent itemsets, association rules could be gotten. Following the article [4], this finding could be conducted by making a loop for every found frequent itemset, then generating every possible subset of them and calculating a so-called confidence.

Consider a frequent itemset *l* whose an instance of subset is (a), then the confidence will be calculated by:

$$confidence(a \rightarrow (l-a)) = \frac{support(l)}{support(a)}$$

If the confidence of any given rules is larger than a given threshold called minimum confidence, this rule will be regarded

as an association rule.

The problem of this method is it considers all the possible subsets of the gotten frequent itemset, which is not appropriate. In order to improve this method, the authors from [4] also suggested 2 methods for finding those rules. The first one is generating the subset of each frequent itemset by their length increasing in each level: if the minimum confidence condition is satisfied by a given subset, it would be added to the output list and its own generated subset with length is 1-lower than its will be added to the next level for checking. This work is done when the generated subset list is blank. The second method is quite different: initially, each frequent itemset will generate their subsets with length of 1; then if any of those subsets satisfies the condition, all of the possible subsets of the considered frequent itemset which contains this size-1 one will be added to the output list without considering. Afterwards, the next level would come with the length increases and without the subsets satisfies in the previous level. This implementation will be terminated until the length of generated subset reach the length of the considered itemset. The latter method seems to be more effective than the former one; however, it depends on the condition of the data: if the association rules with short length is more than the ones with long length, the former will be more suitable; and vice versa, the latter will be better to be applied if the rules with long length are more than the short ones.

Employing the first method and considering a case: giving an itemset I = {ABCDE}, and in the third level, the subset {AB} will be considered at least twice, while it is needed to be considered only once. This problem should be solved more carefully.

## III. Overview of Agriculture big data platform

Previously, to store the agriculture data, we have implemented a big data platform [8][9][10]. It ís described in Fig. 2.

Among all the kinds of input data (Sensor, External services and Internal data), agricultural farms data is taken from Internal ones as CSV files. After being processed for removing the redundancy, those data is handled by Sqoop and Flume, to be stored in Hadoop distributed file system (HDFS) under the form of CSV file. These data is also stored in Hive and Hbase Database as well.

Afterwards, from those data, some analysis like clustering, classifications are made to gain the deeper knowledge inside. And in this paper, we implement another analysis: Association rules mining, which is with the support of MapReduce module

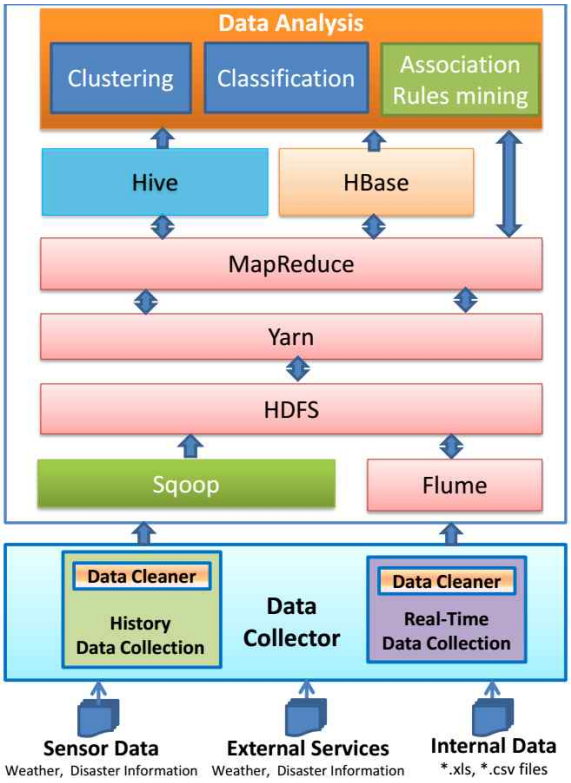


**Fig. 2.** Agricultural Big Data Platform

in our platform. The "transactions" as lists of grown products ("items") will be gotten by dealing with agricultural data stored in HDFS. This pre-processing phase as the preparation will store the result in HDFS again to prepare for the main mining purpose. This method will be described in the next section.

## IV. Association rules mining with agriculture big data

### *4-1 Data pre-processing using MapReduce*

On HDFS, the saved CSV file, which contains information about farms and their corresponding grown products, comprises many lines. Each of them has many fields sequently, like: farms' registration number v1; grown product's name f25; cultivation area f23; studied area f14. However, in the scope of this paper, to get only the data needed for the main mining, 2 fields are considered only: v1 and f25. Each farms could have many grown products, consequently in this file, there are many rows with the same v1's value, but the pair v1 and f25 should be unique.

To get the transactions with their corresponding items like what has been mentioned, a phase of MapReduce is employed. This phase could be described in Fig. 3.

As seen from the Fig. 3, first of all, the big CSV file

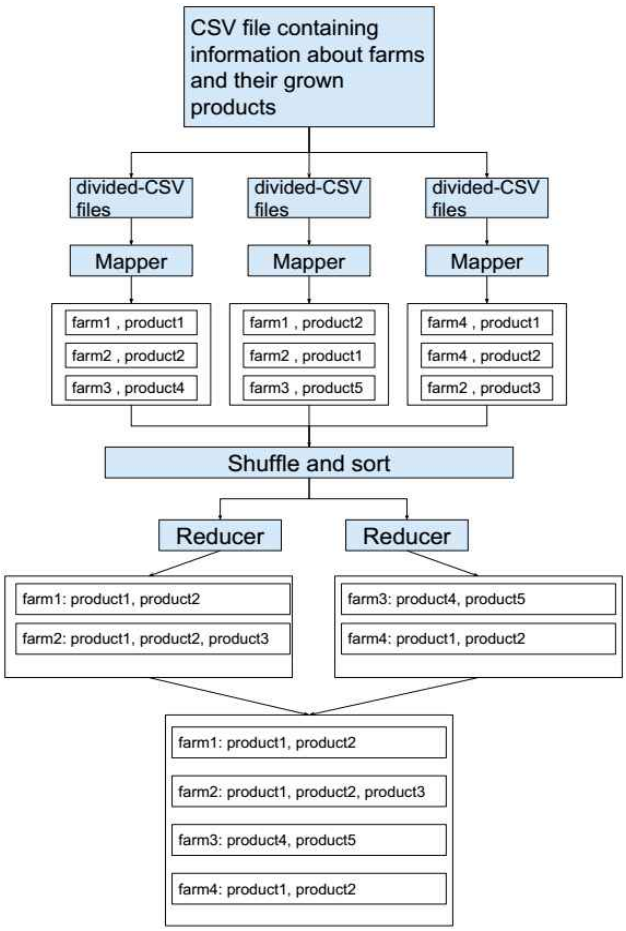


**Fig. 3.** Pre-processing data using MapReduce

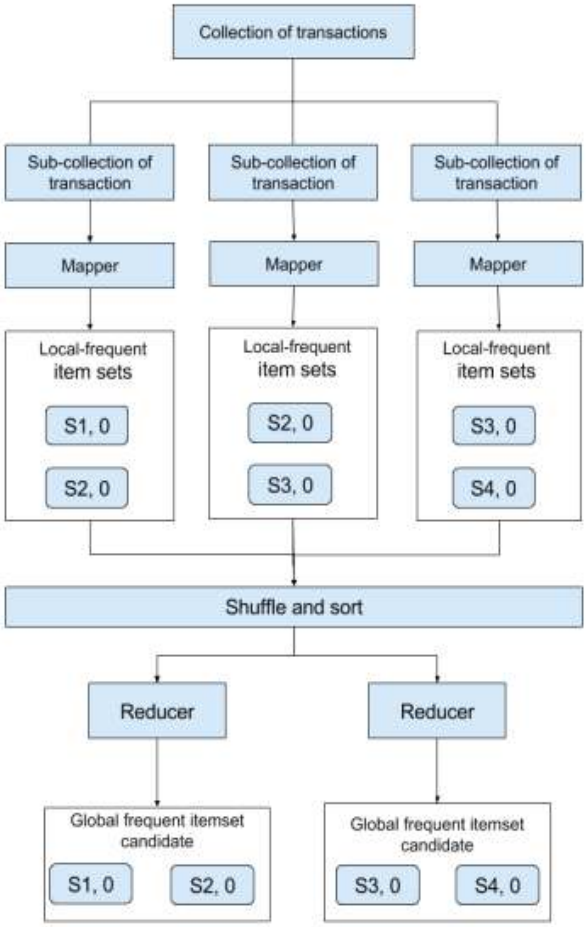


**Fig. 4.** First phase with MapReduce to get global frequent itemset candidates

containing information about farms and their grown products is divided into many smaller parts. Afterwards, they are handled by some mappers. The output of this phase is pairs of key and value, which are farms' registration number and grown products' name respectively. Then in the reduce phase, all the grown products of all the farms will be aggravated, which will return the output containing the transactions and their corresponding items.

After the pre-processing phase, the data is gotten by being storied in a file whose each line contains the grown products of a specific farm in Korea. In order to make the work easier to handle, we implement a hash function to encode each of the product with a specific identification number (id). Afterwards, our algorithm would work with this id instead of product name.

### *4-2 SON algorithm based frequent itemset finding on Hadoop*

As seen from the SON algorithm above, the collection of transaction is divided to be processed and their results are aggregated before being processed again to get the final frequent itemsets. Consequently, it's possible that we could apply this algorithm for Hadoop with 2 phases of MapReduce.

The first phase is described in Fig. 4. In the first map procedure, each non-overlapping sub-collection of transaction will be assigned for each mapper; in order to get their local frequent itemsets $S = \{\{S_i\},\ S_i \in$ frequent itemsets$\}$. The key in this phase is frequent itemset and value is 0 because the quantity of frequent set in this phase is not needed. Then in the reducer procedure, the map procedure's results will be aggregated together to get the global frequent itemset candidates.

Afterward, in the second map procedure phase, the global local frequent itemset candidate will be assigned for each node, which has been allocated with sub-collection of transaction before, to calculate each of their appearance quantity. Then, the second phase' reduce procedure will aggregate the result of all nodes, summarize the number of appearances for each frequent itemset candidate. Finally, their support will be calculated and compared to decide if they are truly frequent itemsets of the main collection or not. This procedure is described in Fig. 5.

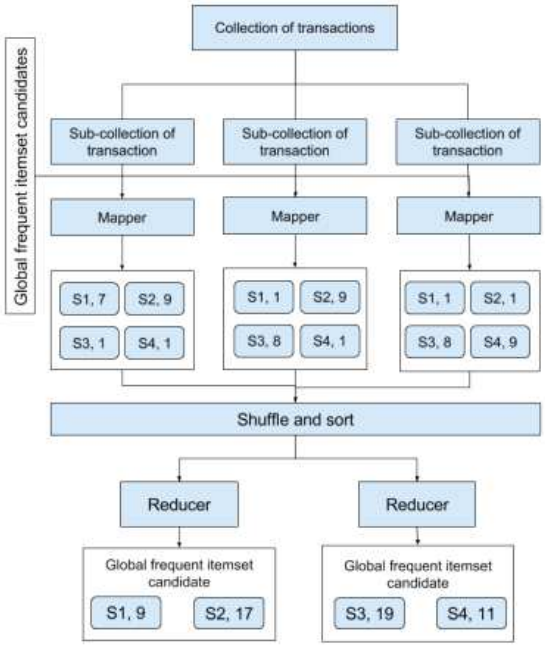


**Fig. 5.** Second phase with MapReduce to get global frequent itemsets

### *4-3 Association rules mining*

In this implementation, the first method mentioned in 2.3 is used and improved to reduce the number of subset to be considered and the number of needed computations. In this phase, the master machine is exploited instead of many machines like before. Our proposed method is described in Algorithm 3.

***Algorithm 3.*** Mining Association rules
**Input**: *Set of frequent itemset FQ = {fq1, fq2, …, fqn} with corresponding support; minimum confidence min_conf*
**Output**: *Association rules*

```
1.  initialize returning_Set = NULL.
2.  for i ← 0 to n do
3.     cur_fq =  FQ.get( i )
4.     subset_list = gen_subset(cur_fq , length(cur_fq) - 1)
5.     while(subset_list != null)
6.        satisfied_subset_list = NULL
7.        for j ←0 to (subset_list →length - 1)
8.           cur_subset = subset_list[ j ]
9.           cur_conf = support(cur_fq) / support(cur_ subset)
10.          if (cur_conf > min_conf)
11.             satisfied_subset_list.add(cur_subset)
12.             returning_Set.add(cur_subset , cur_fq)
13.       subset_list←get_next_subsets(satisfied_subset_list)
14. return returning_Set

 Function get_next_subsets(subset)
1.  return_hash_list = NULL
2.  for i ← 0 to (subset→length - 1)
3.     cur_subset ← subset[i]
4.     for j ← 0 to (length(cur_subset) – 1)
5.        new_set ← ( cur_subset.remove(i) )
6.        return_hash_list.add_hash(new_set)
7.  return return_hash_list
```

The original loop for each satisfied itemsets is used again. However, in this improvement, the number of subset for each level and the total number of computation are reduced by the following cultivation. Firstly, the subset of the considered itemset is generated with length is 1-lower than the length of the original one's. Then the array satisfied_subset_list is used, which contains the considered itemset's subset satisfying the minimum confidence condition. The key idea for the improvement here is using a function called get_next_subsets; which is used to avoid making duplicated itemsets, to generate the new subset for the next length level. This function generates the subset for next level by making loops for each satisfied subset in the previous level, then like the original method, it lists all of subsets of those sets whose length is 1-lower than theirs. What makes the difference here is those generated subsets are added to return_hash_list by using a hash table, so the duplication case is avoided and each subset in the next level will be checked only once.

## V. Implementation and evaluation

To evaluate our implementation, we deployed a Hadoop cluster on five machines: one machine for master node, and fours for computing nodes. Each machine has 4 CPU and 16GB of RAM. There are totally 1,561,632 farms (transactions) and 739 unique grown products (items); which are taken from 2.4

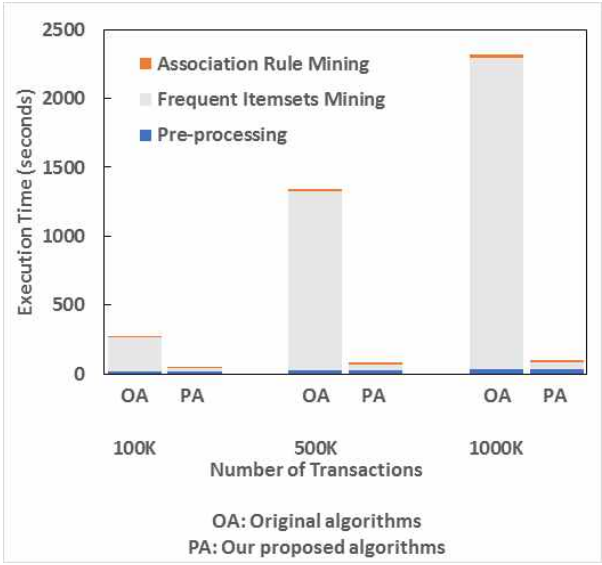


**Fig. 6.** Comparisons of execution time between our proposed algorithms and the original algorithms

gigabytes of data about agriculture farm in Korea.

In Fig. 6, a comparison is made between the original methods and our proposed improved ones conducted on some sets of different transactions quantity. Specifically, the former employs the parallel Brute-Force algorithm on Hadoop to get the frequent itemsets and the method proposed by [4] for mining association rules; while the latter exploits our proposed parallel SON algorithm on Hadoop and our improvement for mining the final results. Both of them use the pre-processing phase of MapReduce for getting the transactions of grown products. Beside; an 0.1 minimum support and an 0.6 minimum confidence are given for both of the cases. Being seen from this Fig., our proposed algorithm uses much less time than the original ones, especially in the frequent itemsets mining phase.

**Table 1.** Evaluation of the pre-processing phase.

| Volume of Data(Mb) | Number of records | Executed time(s) | Number of gotten transactions |
|---|---|---|---|
| 800 | 4,053,802 | 24 | 419,569 |
| 1200 | 6,082,807 | 28 | 642,614 |
| 1600 | 8,110,107 | 32 | 909,954 |
| 2000 | 10,100,124 | 36 | 1,261,391 |
| 2400 (original) | 11,275,355 | 39 | 1,561,632 |

We also make some more evaluations in each phase. Our first evaluation is conducted with the pre-processing phase; which is described in Table 1, for getting the list of farms and their corresponding grown products. This evaluation is made by dividing the original data into some smaller parts with different volumes; then the pre-processing program is executed on them. As seen from Table 1, with the increasing of the data volume from 800 Mb to 2400 Mb; the executed time increases too; but in a marginal way: from 24 seconds to only 39 seconds.

For the second evaluation, we make a comparison of the executing time between BFA conducted in parallel mode and our proposed system with the same environment. The minimum support is set to 0.1 for all the cases. As be shown in Fig. 7, by the increasing of the transactions quantities, executing time used by both of the above methods increases too. Nevertheless, while the executing time by the former increases significantly, it increases marginally following the latter; proving that our algorithm is much better than the BFA. We also made some experiments with higher quantity of transaction, but the Parallel BFA could not handle the enormous volume of data and fails easily, while our proposed algorithm could still handle it well.

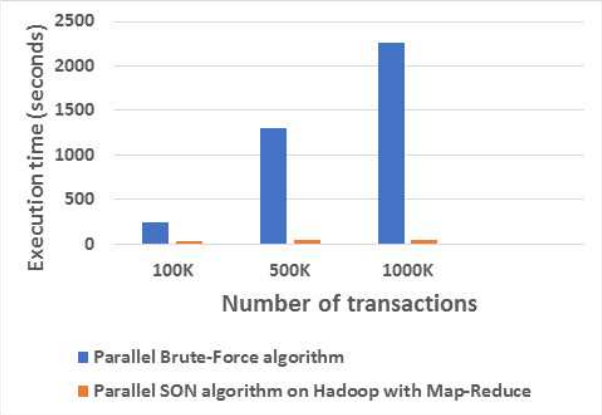


**Fig. 7.** Comparison of execution time between Parallel SON algorithm and Parallel Brute-Force algorithm on Hadoop

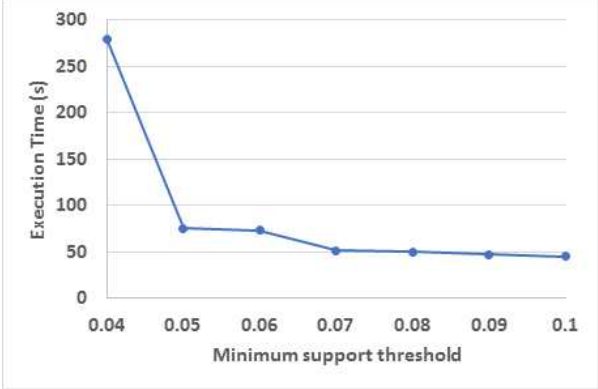


**Fig. 8.** Evaluation by the minimum support threshold

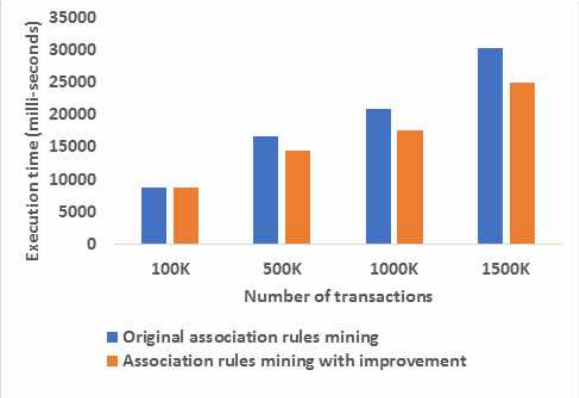


**Fig. 9.** Comparisons of execution time between the association rules mining method proposed by [4] and our improved one.

Afterwards, our evaluation is conducted with the minimum threshold of support to be decreased. Being observable from Fig. 8, the execution time decreases much when the threshold increases from 0.04 to 0.05, then it decreases mildly when the

latter increases to 0.1. It could be explained that when the threshold is low, there are many frequent itemsets generated, so the execution time is quite high at first.

Subsequently, a comparison between the method for mining association rules proposed by [4] and our improved one is described in Fig. 9. This evaluation is made with the change of number of transactions from 419,569 to 1,561,682. Because the duplication of considered subsets is handled; our proposed method spends a little less time comparing to the compared one. With the increasing of the transactions' number; the difference becomes more significant.

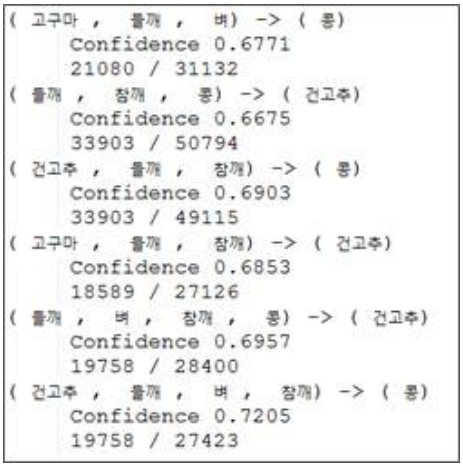


**Fig. 10.** The output of association rules mining

In the final evaluation, based on the found association rules made by Algorithm 3, a mapping from the id of the product to its name is made. Fig. 10 describes the output of this phase. It contains the antecedent; consequent; rule's confidence; number of transaction containing both of consequent and antecedent and number of transaction containing the antecedent sequently. Taking a peek at any rule, it could be understood that when some products on the left of the arrow (→) are chosen as antecedent, there is a rate whose value is *Confidence* percentage for the products on the right side of the arrow to be picked as consequent. Also, the information about the number of transactions containing consequent and antecedent over (/) the number of transactions containing antecedent could be used to describe specifically the mentioned rate above. In this experiment, the minimum support is set as 0.01 and the minimum confidence is set as 0.6.

## VI. Conclusion

In this paper, we paralleled SON algorithm for finding frequent itemset on distributed environment, then the association rules mining is conducted. We implemented 2 phases of MapReduce to deal with the problem of large dataset, which a single machine could not handle well. In the future, we will focus on optimizing the algorithm by using Spark [7]. Moreover, we will also focus on other conditions like the income, growing area to make the recommendation better.

## Acknowledgment

This work was supported by the National Research Foundation of Korea Grant funded by the Korean Government (NRF-2014R1A1A1007734). This work was carried out with the support of "Cooperative Research Program for Agriculture Science and Technology Development (Project No. PJ011823022017)" Rural Development Administration, Republic of Korea.

## References

[1] https://www.growingmagazine.com/fruits/crop-selection/
[2] http://www.cropsreview.com/crop-selection.html
[3] WIREs Data Mining Knowl Discov 2012, 2: 437–456 doi: 10.1002/widm.1074
[4] Agrawal, Rakesh, and Ramakrishnan Srikant. "Fast algorithms for mining association rules." Proc. 20th int. conf. very large data bases, VLDB. Vol. 1215. 1994.
[5] Savasere, Ashok, Edward Robert Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. Georgia Institute of Technology, 1995.APA
[6] Divide and conquer algorithm https://en.wikipedia.org/wiki/Divide_and_conquer_algorithm
[7] https://spark.apache.org/
[8] Van-Quyet Nguyen, Sinh Ngoc Nguyen, Kyungbaek Kim, "Design of a Platform for Collecting and Analyzing Agricultural Big Data", Journal of Digital Contents Society Vol.18 No.1 pp. 149-158, Feburary 28, 2017.
[9] Van-Quyet Nguyen, Sinh Ngoc Nguyen, Duc Tiep Vu, Kyungbaek Kim, "Design and Implementation of Big Data Platform for Image Processing in Agriculture", In Proceedings of KIPS Fall Conference November 04-05, 2016, Pusan National University, Busan, South Korea.
[10] Ngoc Nguyen-Sinh, Quyet Nguyen-Van, Kyungbaek Kim, "Design of Spark based Agricultural Big Data Analysis Platform", In Proceedings of KISM Spring Conference April 29-30, 2016, Silla University, Busan, South Korea.
[11] MapReduce function https://en.wikipedia.org/wiki/MapReduce
[12] Apache Hadoop http://hadoop.apache.org/

**Giang-Truong Nguyen**

2010: Hanoi University of Science and Technology (B.S. Degree)
2017: Chonnam National University, South Korea (M.S. Degree).

2015~2017 : Software Engineer at VNIST
2017~now : School of Electronics and Computer Engineering
※ Research Interest : Big Data Platform, Recommendation Systems

**Van-Quyet Nguyen**

2005: Hung Yen University of Technology and Education. (B.S. Degree).
2011: Hanoi University of Science and Technology (M.S. Degree).
2015: Chonnam National University. South Korea (Ph.D Degree).

2009~2015: Lecturer in Hung Yen University of Technology and Education.
2015~now : School of Electronics and Computer Engineering.
※ Research Interest : Big Data Platform, Content Delivery Network, Recommendation Systems

**Sinh-Ngoc Nguyen**

2009: VietNam National University Ho Chi Minh City - University of Information Technology (B.S. Degree)
2015: Chonnam National University, South Korea (M.S. Degree).

2013~2015 : Software Engineer at Integrated Circuit Design Research and Education Center
2015~now : School of Electronics and Computer Engineering
※ Research Interest : Big Data Platform, Software Defined Network, IoT Security

**Kyungbaek Kim**

1999: Korea Advanced Institute of Science and Technology (KAIST) (B.S. Degree)
2001: Korea Advanced Institute of Science and Technology (KAIST) (M.S Degree)
2007: Korea Advanced Institute of Science and Technology (KAIST) (Ph.D Degree)

2007~2011: Postdoctoral Researcher in University of California Irvine
2012~ now : Professor in Chonnam National University, Gwangju, Korea
※ Research Interest : Distributed System, Middleware, P2P/Overlay Network, Social Network, SDN