Positive and Negative Association Rule mining in Hadoop's MapReduce environment

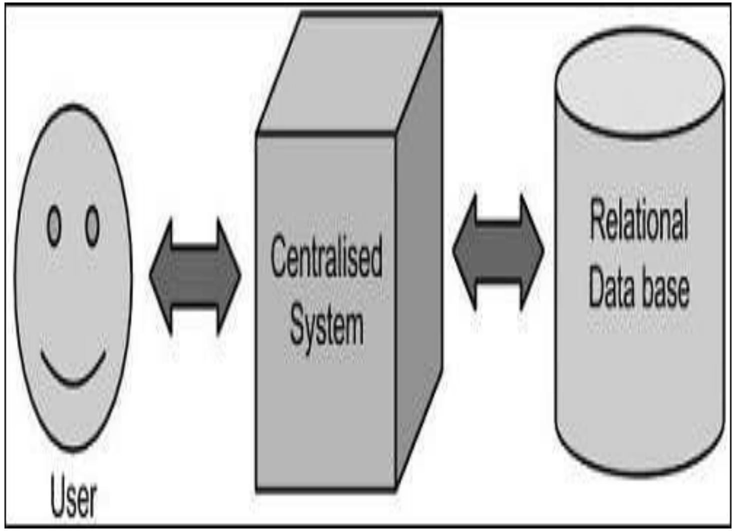
By CS Group - 05 : 1. Anubhav Kumar Giri (20188004)
2. Aritra Chatterjee (20184196)
3. Arvind Kumar Yadav (20184135)
4. Ajay Kumar Gond (20184107)
5. Anshuman Singh Chauhan (20184167)
Supervised By : Dr. Rupesh Kumar Dewang (Professor)

Problem Statement and Solution Proposed

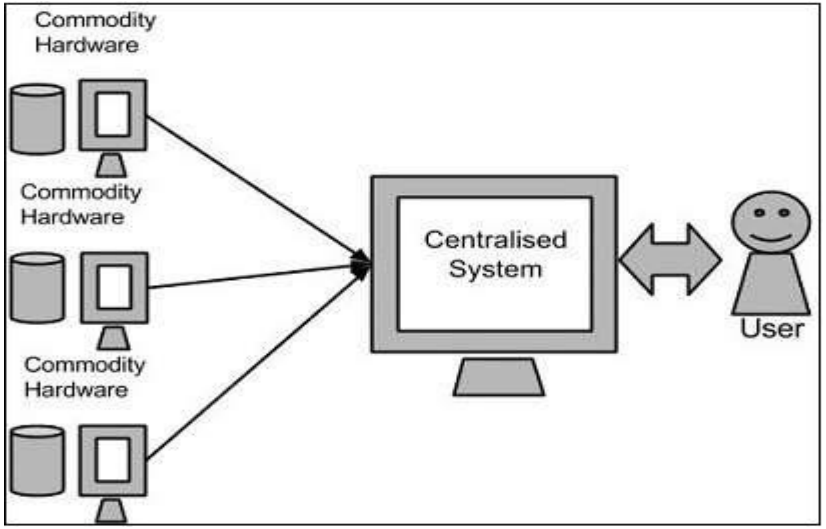
In this paper, we present a Hadoop implementation of the Apriori algorithm. Using Hadoop's distributed and parallel MapReduce environment, we present an architecture to mine positive as well as negative association rules in big data using frequent itemset mining and the Apriori algorithm.

Architecture Overview (Why use Hadoop)

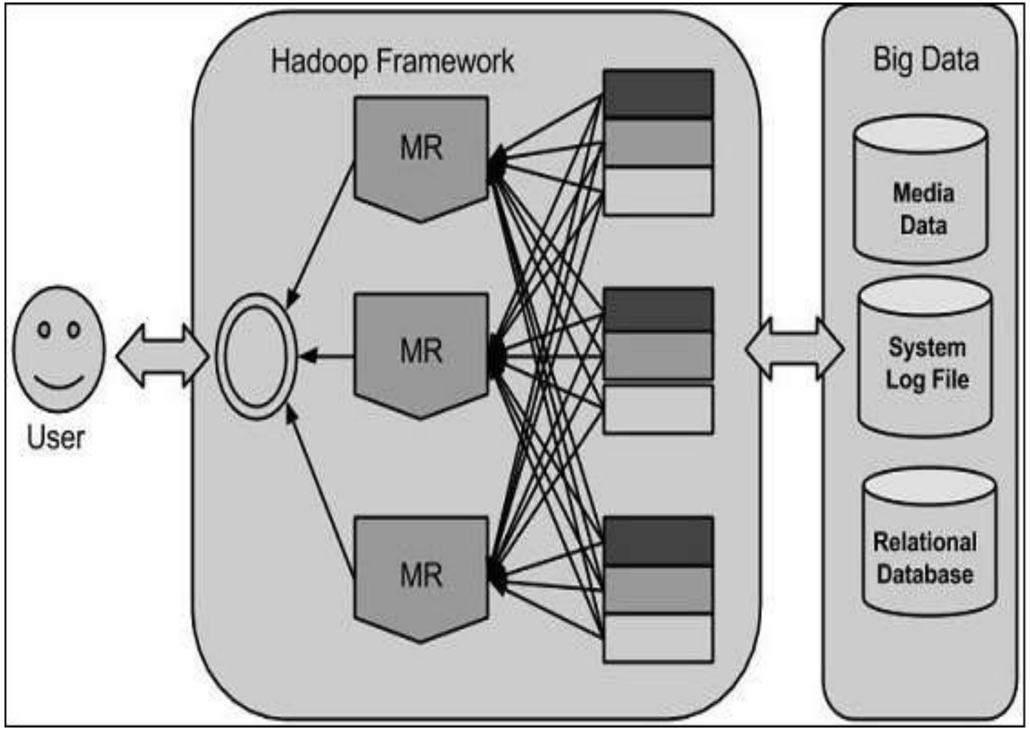
Traditional Approach to Big Data



Google Solution to Big Data



Hadoop Solution to Big Data



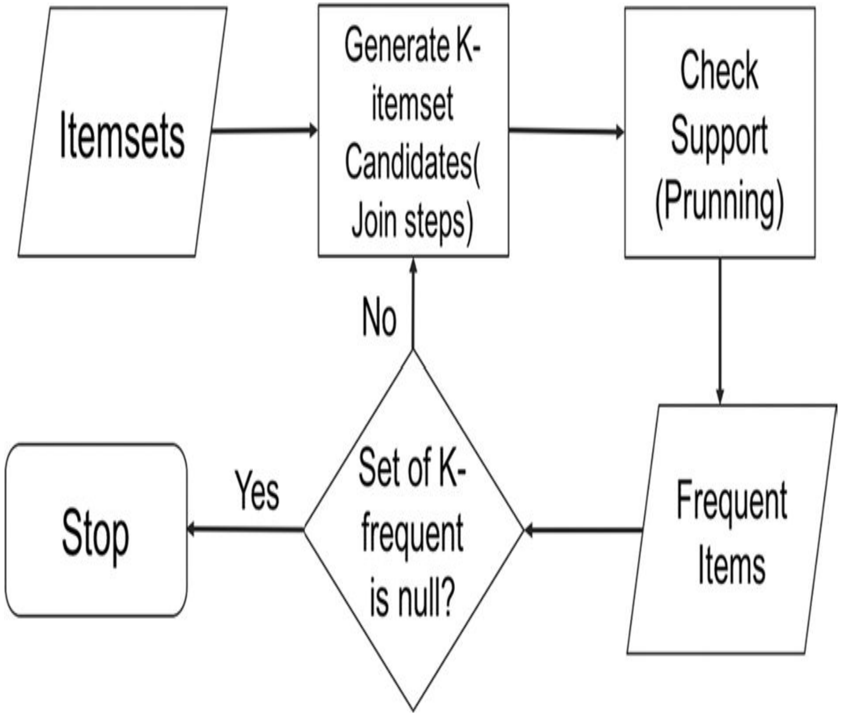
Hadoop is an open source project of Apache, based on Google's Solution to big data and MapReduce environment

Architecture Overview (More about Hadoop)

Hadoop is a large scale distributed framework developed for parallel processing of big data. It is built on top of the Google File System and Google's MapReduce. Given that Hadoop is deployable on a large cluster of nodes, the size of Hadoop's Distributed File System (HDFS) depends on size of cluster and hardware specifications. HDFS makes fast and scalable access to data. Files in HDFS are replicated, hence data stores in HDFS are fault tolerant. MapReduce programming paradigm ensures parallel processing of data.

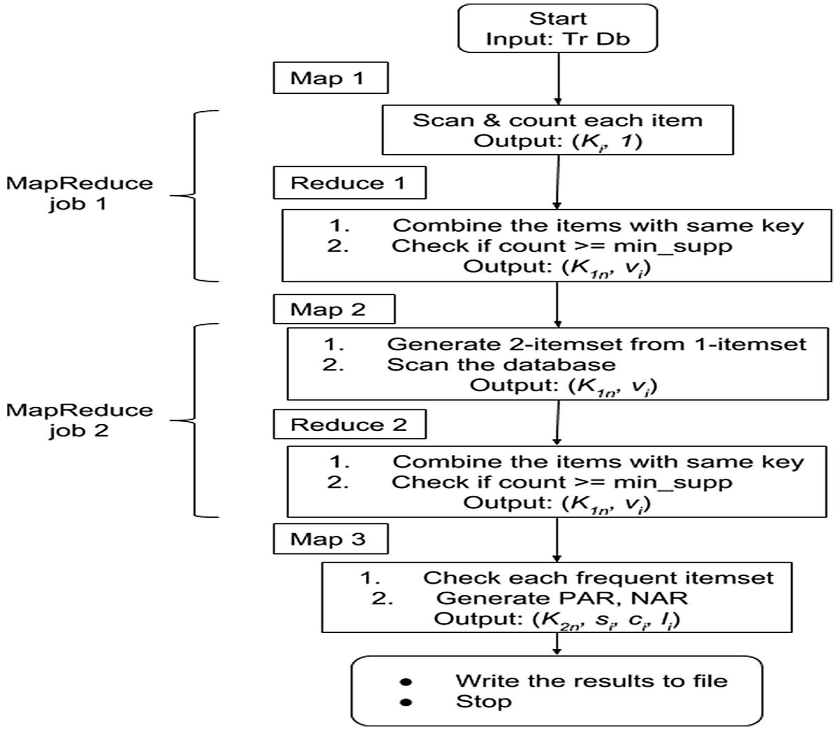
Hadoop engages master(NameNode)-slave(DataNode) architecture. Each cluster contains one NameNode to allocate block IDs and many DataNodes to store actual files.

Architecture Overview (Apriori Algorithm)


Itemsets
Generate K-itemset Candidates(Join steps)
Check Support (Prunning)
Frequent Items
Set of K-frequent is null?
No
Yes
Stop

Apriori algorithm is used to find frequent patterns in data. It uses iterative approach where K-itemsets are used to explore (K+1)-itemsets. First set of frequent 1-itemsets are found and out of those which satisfy minSupp are kept and are then used to find frequent 2-itemsets. This goes on till there are no more itemsets possible. Then itemsets are checked against minConf to determine association rules.

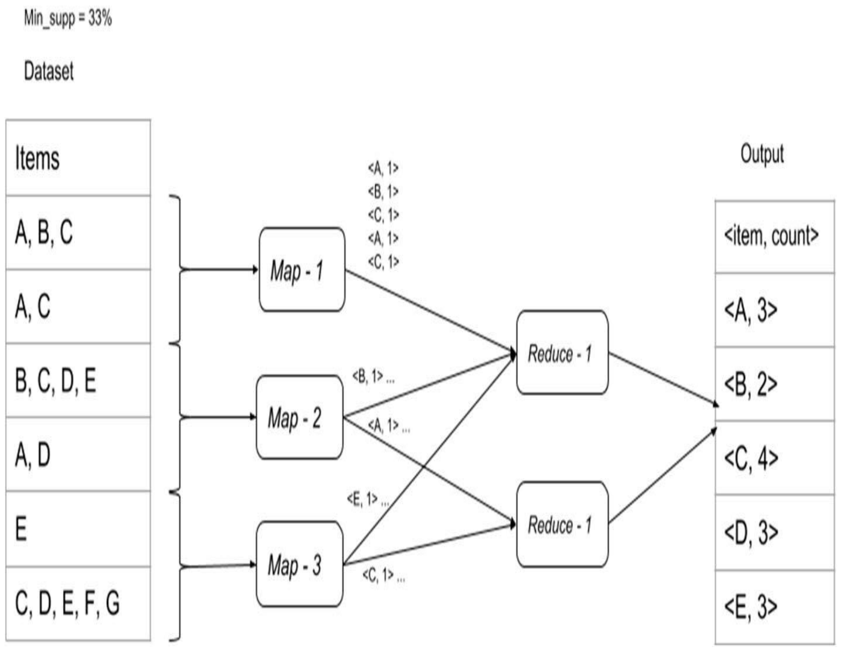
Architecture Overview (Hadoop Implementation)


Start
Input: Tr Db
Map 1
Scan & count each item
Output: (K_i, 1)
MapReduce job 1
Reduce 1
1. Combine the items with same key
2. Check if count >= min_supp
Output: (K_{1n}, v_i)
Map 2
1. Generate 2-itemset from 1-itemset
2. Scan the database
Output: (K_{1n}, v_i)
MapReduce job 2
Reduce 2
1. Combine the items with same key
2. Check if count >= min_supp
Output: (K_{1n}, v_i)
Map 3
1. Check each frequent itemset
2. Generate PAR, NAR
Output: (K_{2n}, s_i, c_i, l_i)
• Write the results to file
• Stop

In this implementation of Apriori Algorithm,

1. Discover frequent itemsets.
2. Find positive and negative association rules.
3. MapReduce jobs are used to find frequent itemsets and MapOnly job is used to find positive and negative association rules.
4. Number of MapReduce iterations depend on number of itemsets being generated.

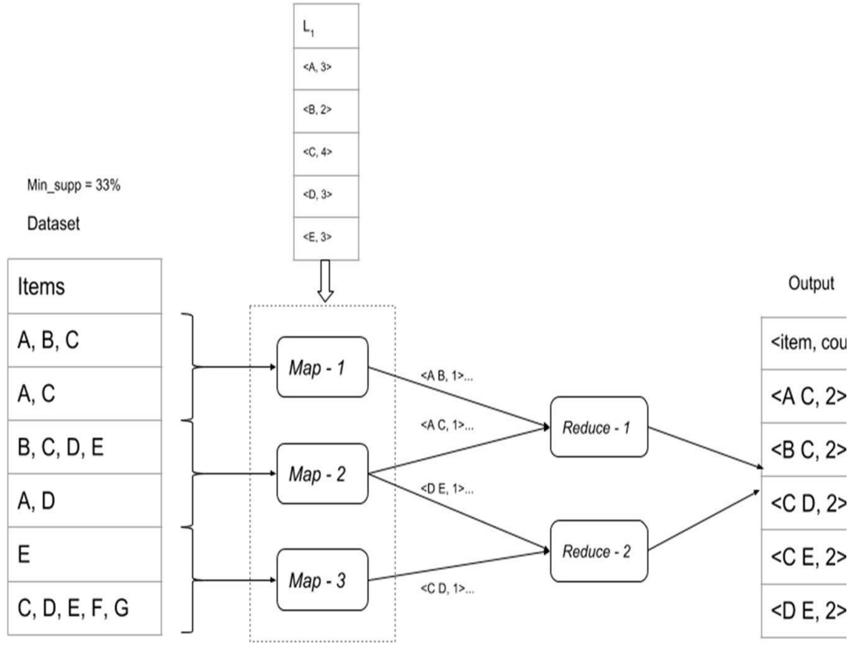
Architecture Overview (MapReduce Job)



MapReduce Job 1

MapReduce Job 1 determines frequent 1-itemset. It takes input from transactional dataset and then HDFS divides data into blocks and distributes over multiple mappers. Mapper outputs (key, value) pair of form (item, count), and passes it to reduce phase. Reduce phase will sum it up and output (item, total_count) and those with total_count >= minSupp are kept as frequent 1-itemset and written to distributed cache.

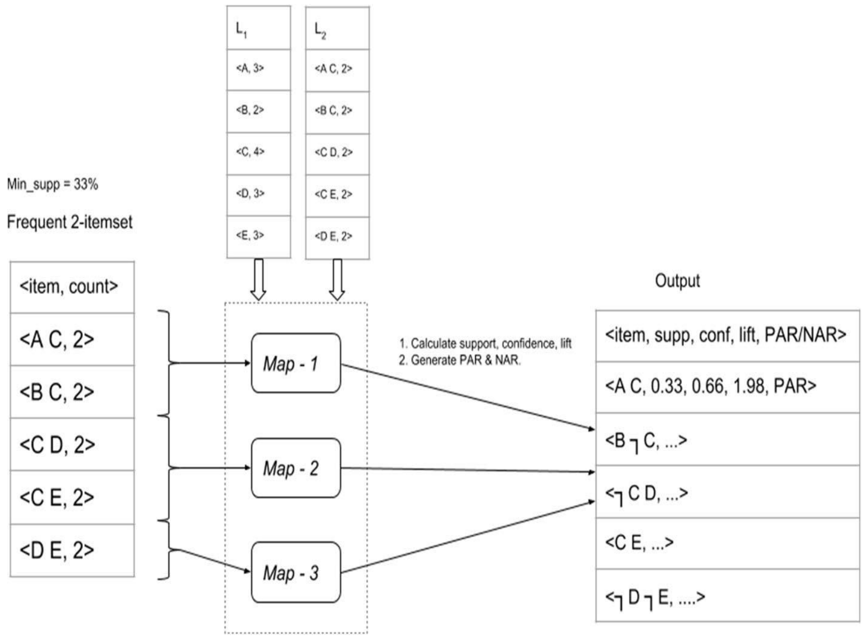
Architecture Overview (MapReduce Job)



MapReduce Job 2

MapReduce Job 2 determines frequent 2-itemset. It takes input from transactional dataset and frequent 1-itemset (read from distributed cache). It then parallels the mapper and reduce phase to get frequent 2-itemset. Frequent 2-itemset and respective support values are written to distributed cache.

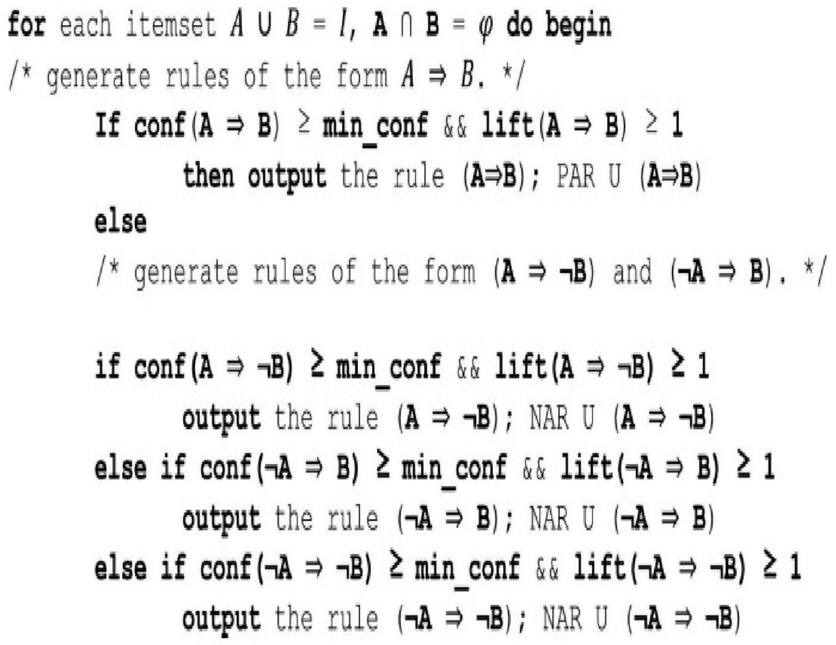
Architecture Overview (MapReduce Job)



MapReduce Job 3

MapReduce Job 3 is a map only operation. The output from MapReduce Job 2 is read into MapReduce Job 3 from the distributed cache. In MapReduce job 3, confidence and lift are calculated for the last generated itemsets to determine positive and negative association rules.

Architecture Overview (Algorithm for Rule Mining)



```
for each itemset $A \cup B = I$, $A \cap B = \varphi$ do begin
/* generate rules of the form $A \Rightarrow B$. */
    If conf(A ⇒ B) ≥ min_conf && lift(A ⇒ B) ≥ 1
        then output the rule (A⇒B); PAR U (A⇒B)
    else
    /* generate rules of the form (A ⇒ ¬B) and (¬A ⇒ B). */

    if conf(A ⇒ ¬B) ≥ min_conf && lift(A ⇒ ¬B) ≥ 1
        output the rule (A ⇒ ¬B); NAR U (A ⇒ ¬B)
    else if conf(¬A ⇒ B) ≥ min_conf && lift(¬A ⇒ B) ≥ 1
        output the rule (¬A ⇒ B); NAR U (¬A ⇒ B)
    else if conf(¬A ⇒ ¬B) ≥ min_conf && lift(¬A ⇒ ¬B) ≥ 1
        output the rule (¬A ⇒ ¬B); NAR U (¬A ⇒ ¬B)
```

Positive Association Rule (PAR) :

1. conf(A=>B) >= minConf
2. lift(A=>B) >= 1

Negative Association Rule (NAR) :

1. conf(A=>~B) >= minConf or
   conf(~A=>B) >= minConf or
   conf(~A=>~B) >= minConf
2. lift(A=>~B) >= 1 (same rule) or
   lift(~A=>B) >= 1 (same rule) or
   lift(~A=>~B) >= 1 (same rule)

Experimental Setup

Software specification of setup used :

1. Java Development Kit (JDK) = 1.8
2. Hadoop = 2.7.3

Hardware specification of setup used :

1. Processor : Intel (R) Core (TM) i7-8750H CPU @ 2.20 Ghz
2. GPU : NVIDIA GeForce 1060 (6 GB VRAM)
3. RAM : 16GB DDR4 Memory
4. Storage Type : SSD

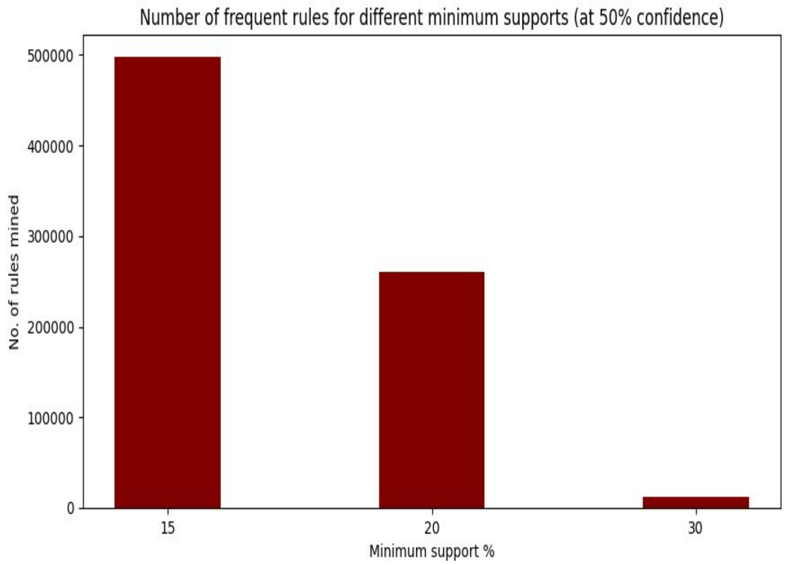
Limitations of Setup Used

Due to some limitations we could not set up the experiment in fully distributed Hadoop environment at the moment. So for present we experimented our code on pseudo distributed mode or single cluster mode in which both namenodes and datanodes are present in same machine and uses different JVM processes to accomplish the task. This reduced the performance and limited us to use small dataset with fewer transactions. However we believe that algorithm works same in both environment and could perform better on bigger dataset in fully distributed mode.

Due to these limitations we used smaller dataset (mushroom.dat), with 8124 transactions for experimental purpose.

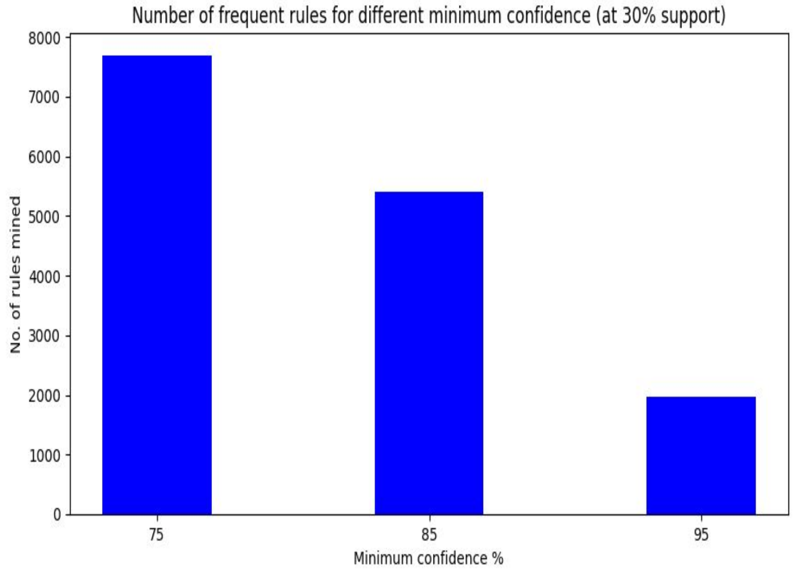
Graphical Results 1

Increasing the minimum support %, decreases the number of frequent item sets generated, and as a result, decreases the number of association rules generated.



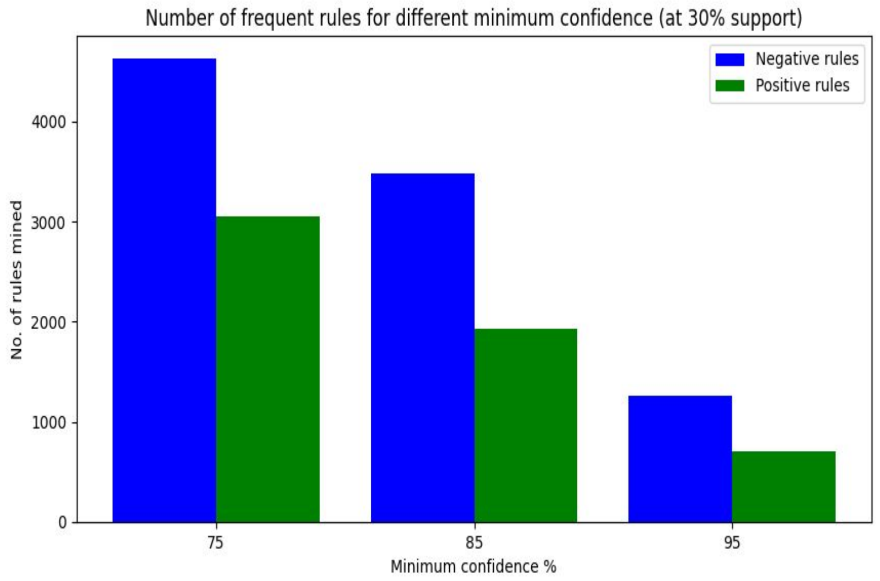
Graphical Results 2

Increasing the minimum confidence, decreases the number of association rules generated.



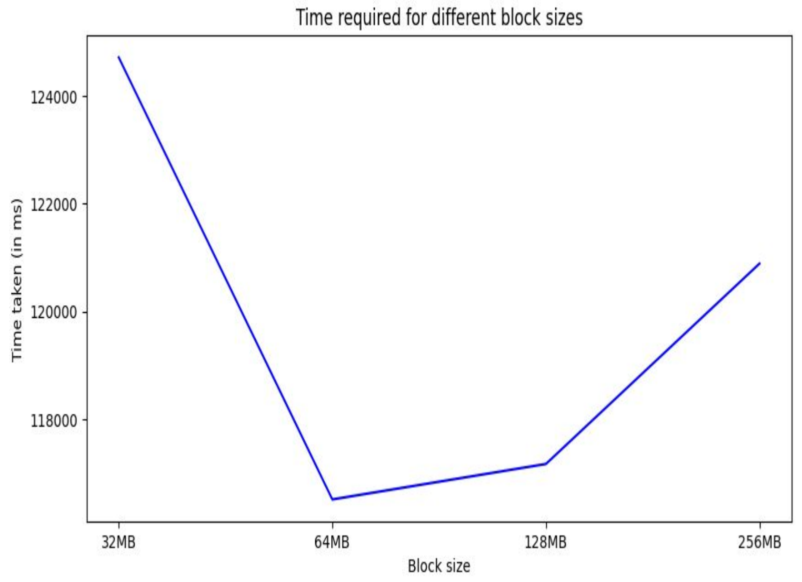
Graphical Results 3

Dividing our results from the previous slide into positive and negative association rules, we can observe that Negative rules are usually more than positive rules for same min confidence. This shows the importance of negative association rules. If we only mined positive rules, we would be losing a lot of information.



Graphical Results 4

Effect of HDFS block size on the run time of our algorithm - we observe a trade-off between parallelism and block overhead. Decreasing block size, increases parallelism and vice-versa, but also results in higher overhead as managing a large number of DataNodes becomes quite a chore.



Scope for Future Work

In our experimental setup we have used pseudo distributed mode of Hadoop which runs NameNodes and DataNodes on same machine providing an emulation of a multi node cluster on single system. Due to this we were bounded to use smaller test sets and could not use it to full extent.

As a future work we propose to use it on cluster of nodes either by creating it manually or using some cloud services like Cloudera or AWS. This would allow us to run the code on bigger datasets and also analyse the effect of increasing number of slave nodes on run time of algorithm.

Scope for Future Work

Providing optimization through the use of combiner functions. In MapReduce, the combiner is an optimization function that reduces the amount of data shuffled between the mappers and reducers. Hadoop allows users to specify a combiner function to be run on the map output. The combiner function's output becomes the input of the reducer function. Combiners are usually more helpful if the MapReduce jobs have limited bandwidth available on the cluster.The combiner output will be in the form (item, list(count, count,..., count)), where items will be from the 1-itemset, 2-itemset, 3-itemset and so on.

References Used

Following references were used for the research paper whose implementation is provided in our project :

1. Aggarwal CC, Yu PS. Mining associations with the collective strength approach. IEEE Trans Knowl Data Eng. 2001;13(6):863–73.
2. Aggarwal CC, Yu PS. A new framework for item-set generation. In: Proceedings of the seventeenth ACM SIGACT- SIGMOD-SIGART symposium on principles of database systems, PODS’98. 1998. p. 18–24.
3. Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large databases. In: ACM SIGMOD conference. New York City: ACM Press; 1993. p. 207–16.
4. Agrawal R, Srikant R. Fast algorithms for mining association rules. In: VLDB 1994 proceedings of the 20th international conference on very large data bases. 1994. p. 487–99.
5. Bala PK. A technique for mining negative association rules. In: Computer2009, Bangalore, India, 2009. p. 1–4.