IMPLEMENTATION OF BIG-DATA APPLICATIONS USING MAP REDUCE FRAMEWORK

Kratika Verma, Prof. Kapil Sahu
*[1]Research Scholar, Sanghvi Institute of Management & Sciene, Indore, M.P, India.
[2]Assistant Professor, Sanghvi Institute of Management & Science, Indore, M.P, India.
*Department of Computer Science & Engineering

ABSTRACT:

Clustering As a result of the rapid development in cloud computing, it & fundamental to investigate the performance of extraordinary Hadoop MapReduce purposes and to realize the performance bottleneck in a cloud cluster that contributes to higher or diminish performance. It is usually primary to research the underlying hardware in cloud cluster servers to permit the optimization of program and hardware to achieve the highest performance feasible. Hadoop is founded on MapReduce, which is among the most popular programming items for huge knowledge analysis in a parallel computing environment. In this paper, we reward a particular efficiency analysis, characterization, and evaluation of Hadoop MapReduce Word Count utility. The main aim of this paper is to give implements of Hadoop map-reduce programming by giving a hands-on experience in developing Hadoop based Word-Count and Apriori application. Word count problem using Hadoop Map Reduce framework. The Apriori Algorithm has been used for finding frequent item set using Map Reduce framework.

Keywords: **Performance analysis, cloud computing, Hadoop Word Count, Apriori algorithm.**

I. INTRODUCTION:

We are residing within the era of large data. In these days a tremendous amount of knowledge is generating everywhere as a result of advances within the web and verbal exchange applied sciences and the pursuits of men and women using smartphones, social media, internet of things, sensor contraptions, online offerings and lots of more. In a similar way, in improvements in knowledge applications and broad distribution of application, a couple of govt and commercial organizations such as Monetary institutions, healthcare institution, schooling and research division, power sectors, retail sectors, lifestyles sciences and environmental departments are all producing a enormous amount of information every day. For examples, international data enterprise (IDC) said that 2.8 ZB (zettabytes) knowledge of universe had been saved in the year of 2012 and this may reach up to forty ZB through 2020 [1]. In a similar fashion Facebook processes round 500 TB (terabytes) knowledge per day [2] and Twitter generates eight TB data daily [3]. The huge datasets no longer handiest comprise structured form of knowledge but greater than seventy five% of the dataset includes uncooked, semi-structured and unstructured type of data [4]. This large quantity of information with one of a kind codecs can be viewed as giant information.The derivation of big knowledge is indistinct and there are a lot of definitions on huge data. For examples, Matt Aslett outlined massive knowledge as “tremendous data is now virtually universally understood to refer to the recognition of larger business intelligence through storing, processing, and examining data that was previously ignored because of problem of normal data management applied sciences” [5]. Recently, the term of giant data has got a brilliant momentum from governments, industry and research communities. In [6], significant information is outlined as a term that encompasses using tactics to capture, approach, analyze and visualize potentially significant datasets in a cheap timeframe now not obtainable to usual IT applied sciences. Hadoop is an industrial scale batch processing distributed computing tool. It has the capability to connect computers with multiple processor cores with a scale ranging from hundreds to thousands. Vast volumes of data can be efficiently distributed across clusters of computers using Hadoop. The Hadoop scale consists of hundreds of gigabytes of data at the least. Hadoop has been built with the capability to manage vast data sets whose size can easily lie between couple of gigabytes to thousands of petabytes. Hadoop provides its solution in the form of a Distributed File System which splits the data and stores it in several different machines. This enables parallel processing of the problem and efficient computation is possible. The design of Hadoop is such that it can efficiently manage vast quantity of data sets by taking advantage of clustered computing or by connecting hundred of machines with processing power in parallel. Theoretically speaking, a single, powerful thousand CPU machine would be much more expensive than thousands of machines with individual CPUs thus making it an easier investment. Hadoop offers a cost effective solution by tying these smaller and cheaper machines together.

After the data is loaded into clusters in Hadoop it is distributed to all the nodes. The HDFS then splits the data into sets which allow management by individual nodes within the cluster. To handle unavailability of data due to failure, each part is also replicated across the cluster. The data is also re-replicated in response to failure of the system. All these parts of data are easily accessible through a universal namespace, despite the parts being distributed and replicated on multiple machines.

II. LITERATURE SURVEY:

1] **Samneet Singh and Yan Liu**,“A Cloud Service Architecture for Analyzing Big Monitoring Data”,ISSNll1007-0214ll05/10llpp55-70 Volume 21, Number 1, February 2016 .In this paper, author proposed a structure that integrates search-headquartered clusters and semantic media wiki by using relaxation APIs to help the exploration of cloud monitoring data. This structure advantages from an internet-based Media-Wiki interface and enables a person to outline the entry to monitoring knowledge and prepare the processing results. The quest-based cluster developed on SolrCloud permits indexing of significant size of knowledge, and thus makes the entire architecture compatible to explore and display the ever-gathering data such as the traces constructed from knowledge centers. The structure additionally involves an extension, which runs spark on Yarn cluster for deploying effective evaluation ways for gigantic knowledge set. It utilizes the spark’s MapReduce paradigm to establish the cluster in the dataset utilizing k-way clustering approach.

[2] JOSEPH A. ISSA, “Performance Evaluation and Estimation Model Using Regression Method for

Hadoop WordCount", Received November 19, 2015, accepted December 12, 2015, date of publication December 18, 2015, date of current version December 29, 2015. In this paper, the writer offered a distinct performance analysis and analysis for Hadoop WordCount workload utilizing different processors similar to Intel's ATOM D525, Xeon X5690, and AMD's Bobcat E350. Our analysis suggests that Hadoop WordCount is compute-sure workload in both map segment and scale down segment. The outcome exhibit that enabling HT and growing the number of sockets have a high impact on the Hadoop WordCount performance even as reminiscence velocity and capacity does now not have an impact on efficiency vastly.

[3] Yaxiong Zhao, Jie Wu, and Cong Liu, "Dache: A Data Aware Caching for Big-Data Applications Using the MapReduce Framework",ISSNll10070214ll05/10llpp39-50 Volume 19, Number 1, February 2014 .In this paper, author recommends Dache, a knowledge-conscious cache framework for big-data functions. In Dache, tasks publish their intermediate outcome to the cache manager. A project queries the cache supervisor before executing the specific computing work. A novel cache description scheme and a cache request and reply protocol are designed. We enforce Dache by means of extending Hadoop. Test bed experiment results show that Dache tremendously improves the completion time of MapReduce jobs.

[4] Zhuoyao Zhang Ludmila Cherkasova, "Benchmarking Approach for Designing a MapReduce Performance Model"**,** *ICPE'13,* April 21-24, 2013.In this work, author presents a novel efficiency analysis framework for answering this question. We observe that the execution of every map (lessen) duties consists of distinctive, good-defined knowledge processing phases. Handiest map and scale back services are customized and their executions are consumer-outlined for exclusive MapReduce jobs. The executions of the remaining phases are time-honored and rely on the amount of information processed by means of the phase and the performance of underlying Hadoop cluster. First, we design a suite of parameterizable micro benchmarks to measure normal phases and to derive a platform performance model of a given Hadoop cluster.

III. EXISTING SYSTEM:

Based on Amdahl's law definition we discussed before, the performance of a given processor can be divided into two parts, the part which increases with the performance improvement and is said to scale is defined as variable *a*, and the other part which does not improve due to the performance enhancement and is said to not scale is defined as variable *b*. The *a* and *b* variables can be derived using the basic definition of Amdahl's law which can be written in the form of:

$$T = T_o + (T_1 - T_o) \times \frac{I_1}{I} \qquad (1)$$

where T1 is the measured execution time at a given input size I1, and To be the non-scale execution time. We can write To in terms of a second measurement T2 at I2:

$$T_o = \frac{T \times I_2 - T_1 \times I_1}{(I_2 - I_1)} \qquad (2)$$

When we substitute Eq(2) for To in Eq(1) we obtain Amdahl's law in terms of two specific measurements without reference to:

$$T = a + b \times \frac{1}{I} \qquad (3)$$

$$a = \frac{I_2 \times T_2 - I_1 \times T_1}{(I_2 - I_1)} \qquad (4)$$

$$b = \frac{(I_1 \times I_2)(T_2 - T_1)}{(I_2 - I_1)} \qquad (5)$$

The variables *a* and *b* can be transformed to the performance instead of the time domain by using P D 1/T. This will give us *a* and *b* variables in terms of performance and input size as shown in Eq(6) and Eq(7).

$$a = \frac{P_1 \times I_2 - P_2 \times I_1}{(P_1 \times P_2)(I_2 - I_1)} \qquad (6)$$

$$b = \frac{(I_1 \times I_2)(P_2 - P_1)}{(P_1 \times P_2)(I_2 - I_1)} \qquad (7)$$

For two data points, we will have (*I*1; *P*1) and (*I*2; *P*2), and for *n* data points, we will have (*I*1; *P*1); : : : ; (*In*; *Pn*). We expect these points to satisfy an equation of the form (except for noise):

$$P_i = \frac{I_i}{a \times I_i + b} \qquad (8)$$

Because of noise, we cannot expect to end values for *a* and *b* that produce equality for each point *i*. In this case, we resort to the theory of linear least-squares estimation to obtain best estimates for *a* and *b*. In particular, given *a* and *b*, we take the error in our estimate for Pi in terms of Ii to be the difference between the measured and estimated value for Pi:

$$e_i = \left(\frac{1}{P_i} - \left(a + b \cdot \frac{1}{P_i}\right)\right) \qquad (9)$$

The best estimates for*a* and *b* are those that minimize the sum of the squares of these errors:

$$E = \sum_{i=1}^{n} 2e^2 \qquad (10)$$

The estimates for *a* and *b* are those at which the values of the partial derivatives *@E=@a* and *@E=@b* are simultaneously zero. By computing these derivatives explicitly, we obtain equations satis_ed by the best choices for *a* and *b*, which is the best functional _t to the measured data.

$$\frac{\partial E}{\partial a} = \sum_{i=1}^{n} 2e_i \frac{\partial}{\partial a}\left(\frac{1}{P_i} - \left(a + b \cdot \frac{1}{I_i}\right)\right) \qquad (11)$$

Thus, ∂E/∂a = 0 implies

$$\left(\sum_{i=1}^{n} \frac{1}{P_i}\right) = n \cdot a + \left(\sum_{i=1}^{n} \frac{1}{I_i}\right) \cdot b, \qquad (12)$$

$$c_{11} \equiv n, \qquad (18)$$

$$c_{12} \equiv c_{21} \equiv \sum_{i=1}^{n} \frac{1}{I_i}, \qquad (19)$$

$$c_{22} \equiv \sum_{i=1}^{n} \frac{1}{I_i^2}, \qquad (20)$$

$$d_1 \equiv \sum_{i=1}^{n} \frac{1}{P_i}, \qquad (21)$$

$$d_2 \equiv \sum_{i=1}^{n} \frac{1}{P_i}\frac{1}{I_i}. \qquad (22)$$

Given these best estimates for a and b in terms of (I1, P1); : : : ; (In, Pn), we have the following best estimate for P in terms of I.

IV. PROBLEM FORMULATION

Hadoop is specially designed for two core concepts: HDFS and MapReduce. Both are related to distributed computation. Hadoop architecture is primarily a distributed master slave architecture that consists of a single master and many slaves. The Hadoop Distributed File System (HDFS) is used for storage and MapReduce for computational capabilities. The functions of Hadoop in the architecture are data partitioning and parallel computation of large datasets. Its storage and computational capabilities scale with the addition of hosts to a Hadoop cluster, and can reach volume sizes in the petabytes on clusters with thousands of hosts [2].

The MapReduce master schedules the computational work on the slave nodes and organizes where the computational work will be scheduled. The HDFS master is responsible for storing the files and partitioning the storage across the slave nodes and keeping track of where data is located. There is a need to extract useful information from the data and to interpret the data. The traditional Apriori algorithm is generally used top to bottom approach.Scans the very big database repeatedly to produce LK increase I/O load and reduce efficiency. Each item in the candidate item sets must scan database one time to decide whether it can be joined to the Lk. So it needs to scan the transaction database as the same number as the elements of the frequent item set. When it carries on the k-th scanning, the algorithm does not use the former result. At the same time, the algorithms mentioned above are improvement or extension 846 based on architecture of the existing algorithm, the efficiency has not been much improved. The key of the improved algorithm that we propose in this work that how to reduce the scan through the results of previous scan.

The main aim of this paper is to give implements of Hadoop map-reduce programming by giving a hands-on experience in developing Hadoop based Word-Count and Apriori application. Word count problem using Hadoop Map Reduce framework. The Apriori Algorithm has been used for finding frequent item set using Map Reduce framework.

V. PROPOSED WORK:

Word count is a typical example where Hadoop map reduce developers start their hands on with. This sample map reduce is intended to count the no of occurrences of each word in the provided input files.The word count operation takes place in two stages a mapper phase and a reducer phase. In mapper phase first the test is tokenized into words then we form a key value pair with these words where the key being the word itself and value ‘1’. For example consider the sentence “tring tring the phone rings”In map phase the sentence would be split as words and form the initial key value pair as

<tring,1>

<tring,1>

<the,1>

<phone,1>

<rings,1>

In the reduce phase the keys are grouped together and the values for similar keys are added. So here there are only one pair of similar keys ‘tring’ the values for these keys would be added so the out put key value pairs would be

<tring,2>

<the,1>

<phone,1>

<rings,1>

This would give the number of occurrence of each word in the input. Thus reduce forms an aggregation phase for keys. Here is tha apriori algorithm where map reduce framework has been applied.

```
Input: database (D), minimum support (min_sup).
Output: frequent item sets in D.
L1= frequent item set (D)
j=k; /* k is the maximum number of element in a transaction from the database*/
for k= maxlength to 1
{
    fori=k to 2
    {
        for each transaction Ti of order i
        {
            if (Ti has repeated)
            {
                Ti.count++;
            }
            m=0;
            while (i<j-m)
            {
                if (Ti is a subset of each transaction Tj-m of order j-m)
                Ti.count++; m++; }
            If (Ti.count>=min_sup)
            Rule Ti generated

        }
```

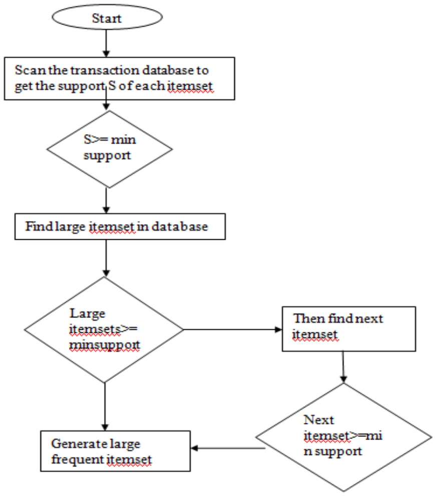
Start
Scan the transaction database to get the support S of each itemset
S>= min support
Find large itemset in database
Large itemsets>= minsupport
Then find next itemset
Next itemset>=mi n support
Generate large frequent itemset

Figure 5|Flow chart of improved Apriori algorithm

```
   Algorithm Mapper()
   {
   String line = value.toString();
   String[] words=line.split(",");
   for(String word: words )
   {
   Text outputKey = new Text(word.toUpperCase().trim());
IntWritable outputValue = new IntWritable(1);
     con.write(outputKey, outputValue);
   }
   }
   Algorithm Reducer()
   {
```

```
int sum = 0;
  for(IntWritable value : values)
  {
  sum += value.get();
  }
  con.write(word, new IntWritable(sum));
}
}
```

VI. CONCLUSION

Map-Reduce have become an important platform for a variety of data processing applications. Word Count Mechanisms in Map-Reduce frameworks such as Hadoop, suffer from performance degradations in the presence of faults. Word Count Map-Reduce, proposed in this paper provides an online, on-demand and closed-loop solution to managing these faults. The control loop in word count mitigates performance penalties through early detection of anomalous conditions on slave nodes. Anomaly detection is performed through a novel sparse-coding based method that achieves high true positive and true negative rates and can be trained using only normal class (or anomaly-free) data. The local, decentralized nature of the sparse-coding models ensures minimal computational overhead and enables usage in both homogeneous and heterogeneous Map-Reduce environments.

Map-Reduce have become an important platform for a variety of data processing applications. Word Count Mechanisms in Map-Reduce frameworks such as Hadoop, suffer from performance degradations in the presence of faults. Our algorithm provides an online, on-demand and closed-loop solution to managing these faults. The local, decentralized nature of the sparse-coding models ensures minimal computational overhead and enables usage in both homogeneous and heterogeneous Map-Reduce environments.

REFERENCES:

[1] Samneet Singh and Yan Liu,“A Cloud Service Architecture for Analyzing Big Monitoring Data”,ISSNll1007-0214ll05/10llpp55-70 Volume 21, Number 1, February 2016

[2] JOSEPH A. ISSA, “Performance Evaluation and Estimation Model Using Regression Method for Hadoop WordCount”, Received November 19, 2015, accepted December 12, 2015, date of publication December 18, 2015, date of current version December 29, 2015.

[3] Yaxiong Zhao, Jie Wu, and Cong Liu, “Dache: A Data Aware Caching for Big-Data Applications Using the MapReduce Framework”,ISSNll10070214ll05/10llpp39-50 Volume 19, Number 1, February 2014

[4] Zhuoyao Zhang Ludmila Cherkasova, “Benchmarking Approach for Designing a MapReduce Performance Model”**,** *ICPE’13,* April 21-24, 2013

[5] Nikzad Babaii Rizvandi, Albert Y. Zomaya , Ali Javadzadeh Boloori, Javid Taheri1, “On Modeling Dependency between MapReduce Configuration Parameters and Total Execution Time”, 2012

[6] Nikzad Babaii Rizvandi, Javid Taheri1, Reza Moraveji, Albert Y. Zomaya, “On Modelling and Prediction of Total CPU Usage for Applications in MapReduce Enviornments”, 2011

[7] A. Baratloo, M. Karaul, Z. Kedem, and P.Wyckoff, ``Charlotte: Meta computing on theWeb,'' in *Proc. 9th Int. Conf. Parallel Distrib. Comput. Syst.*, 1996, pp. 1_13.

[8] J. Bent, D. Thain, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. Livny, ``Explicit control in the batch-aware distributed _le system,'' in *Proc. 1st USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Mar. 2004, pp. 365_378.

[9] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier,``Cluster-based scalable network services,'' in *Proc. 16th ACMSymp. Oper. Syst. Principles*, Saint-Malo, France, 1997, pp. 78_91.

[10] S. Ghemawat, H. Gobioff, and S.-T. Leung, ``The Google _le system,'' in *Proc. 19th Symp. Oper. Syst. Principles*, New York, NY, USA, 2003, pp. 29_43.