ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 5, Issue 4, April 2016

Enhancement of Map Reduce Framework Based on K-Medoid

D. Arul Selve, Dr. K. Kavitha

***Abstract*— The Map-Reduce model streamlines the huge scale data handling on possessions group by abusing similar map and reduces assignments. Though numerous activities have been made to increase the execution of Map-Reduce works, they ignore the network activity produced in the shuffle stage, which assumes a fundamental part in implementation upgrade. Generally, a hash capacity is utilized to segment intermediate values among reduce assignments, which, nonetheless, is not effective of the fact that network topology and its data range connected with every key are not thought seriously about. To overcome this, Decomposition based dynamic algorithm and online algorithm is proposed to manage the huge scale optimization issue for huge information application is similarly intended to change information package and accumulation in a dynamic way. The next step to be enhanced in this method is cost and time complexity.**

In proposed system chooses K- medoid clustering algorithm for resource allocation which is implemented in decomposition based distribution algorithm. In the proposed work, the Euclidean distance is calculated from the dataset of latitude and longitude of the location to find the distance between locations which enhance network traffic during shuffling. This method is cost and time efficient compared to the previous works.

***Index Terms*—Map Reduce, Big Data, Data partition, Aggregation**

I. INTRODUCTION

As data and work rise, it takes a longer time to generate results. To produce the result in timely and efficient manner, it needs to think big. A convinced way is carried out to increase the work across many computers i.e. one needs to scale out. Map Reduce is an encoding form which is designed for dealing out large volumes of data in parallel. It is done by dividing the work into a set of independent tasks. Map Reduce programs renovate lists of input data elements into lists of output data essentials. The Map Reduce data elements are absolute, i.e. they cannot be updated. The policy is to put the file into HDFS one time and can read the file 'n' number of times

Map Reduce is a programming pattern which processes the portioned data and aggregates the intermediate results. Map

***D. Arul Selve**, Department of Computer Science Mother Teresa Women's University, Kodaikanal, India.*
***Dr. K. Kavitha**, Assistant professor, Department of Computer Science, Mother Teresa Women's University, Kodaikanal, India*

Reduce was introduced by Google. It can also be defined as a

software structure supporting scattered computing on large datasets on clusters of computer. Programs can be implemented in any language to execute the jobs which are written in Map Reduce paradigms. The motivation of Map Reduce came from two functions namely map( ) and reduce( ) functions in well-designed programming model

Map function: A function that is applied on the input individual chunks of portioned data in functional programming is called map ( ) function. This portioning is made by the Hadoop Distributed file system (HDFS). The portioning volume is a tunable constraint. Different input datasets produce different transitional values and hence it describes the characteristic that map ( ) functions run in parallel manner.

Reduce function: The intermediate values are combined to form a list. This is done after the map ( ) is over. i.e., the intermediate values are combined to get a final result for the similar output key. Reduce ( ) function also runs in parallel and each of the reduce ( ) function runs on a different production key which are generated by a map ( ) function. Reduce ( ) function only starts after the map () function.

Actual problem rise in shuffle phase which is between map and reduce function. The problem arise here is the serious network traffic which can be minimized using Decomposition based distribution algorithm along with an online algorithm to aggregate data.

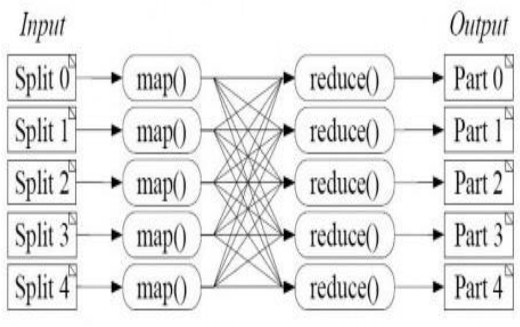

Fig 1: Architecture of Map Reduce Process

In order to further reduce the network traffic and cost K –MEDOID algorithm is used in above process.

The rest of the paper is organized as: In section II, review recent related work is done. Section III presents a system model with distribution and online algorithm, IV presents the K-Medoid Algorithm, followed by V presents the system process. Finally, Section VI concludes the paper.

II. RELATED WORK

B. Palanisamy, A. Singh, L. Liu, & B. Jain , have presented Purlieus, a Map Reduce resource allocation system, to enhance the performance of Map Reduce jobs in the cloud by locating transitional data to the local machines. This locality-awareness shrinks network traffic in the shuffle phase produce in the cloud data center [1].

L. Fan, B. Gao, X. Sun, F. Zhang, & Z. Liu, have proposed and assess two effective load balancing approaches to data skew managing for Map Reduce-based entity resolution. Unfortunately, all above work concentrate on load balance at reduce tasks, ignoring the network traffic during the shuffle phase [2].

Ibrahim et al., have developed a fairness-aware key partition approach that keeps path of the sharing of transitional keys' frequencies, and assure a fair distribution among reduce tasks [3].

A. Blanca & S. W. Shin, have researched the question of whether optimizing network usage can enhance system performance and found that high network consumption and low network blocking should be achieved simultaneously for a job with good performance [4].

III. SYSTEM MODEL

Typical Map Reduce jobs on a large cluster consist of a set of N machines. Let d_{xy} denote the space between two machines x and y, which represents the rate of delivering a unit data. As the job is executed, two types of tasks are created namely Map and Reduce tasks. The sets of map and reduce errands are denoted by M and R, respectively, which are placed on machines already. The input data are alienated into independent chunks and processed in parallel by map tasks. The generated transitional results in forms of key/value pairs are shuffled and sorted by the framework, and then are fetched by reduce tasks to make final results. The objective in this paper is to further minimize the total network traffic cost of a Map Reduce job by jointly considering aggregator placement and intermediate data partition.

A.DISTRIBUTED ALGORITHM

The system model also includes the distributed algorithm to solve the problem on multiple machines in a parallel manner. The basic idea of this algorithm is to decompose the original large-scale problem into several distributive solvable sub problems that are coordinated by a high-level master problem. Below is the distributed algorithm.

```
Step 1: Set t=1 and $v_j^p$ $(j \in A, p \in P)$ to arbitrary non
        negative values;
Step 2: For t<T do
Step 3: distributively solve the sub problem SUB_DP and
        SUB_AP on multiple machines in a parallel manner;
Step 4: update the values of $v_j^p$ with the gradient method, and
        sent the result to all sub problems;
Step 5: set t=t+1;
Step 6: end for
```

In the above algorithm t is the time slot, v_j^p denotes Lagrangian multiplier where j is the node in A, set of nodes in aggregation layer. Here T is the total time taken for the process and p is an intermediate key in P, the set of all intermediate keys.

In summary, above algorithm is used to solve the distribution problem efficiently.

The network topology is based on three tier architectures: an access tier, an aggregation tier and a core tier. The access tier is built of cost effective Ethernet switches connecting rack VMs. The access switches are linked via Ethernet to a set of aggregation switches which in turn are connected to a layer of core switches. An inter-rack link is the most contentious resource as all the VMs hosted on a rack transfer data across the link to the VMs on other racks. The VMs are distributed in three different racks, and the map-reduce tasks are scheduled as per the system. The intermediate data forwarding between mappers and reducers should be transferred across the network. The VMs are distributed in three different racks, and the map reduce tasks are scheduled. For example, rack 1 consists of node 1 and 2; mapper 1 and 2 are scheduled on node 1 and reducer 1 is scheduled on node 2. The intermediate data forwarding between mappers and reducers should be transferred across the network.

B. ONLINE ALGORITHM

Online algorithm dynamically adjusts data partition and aggregation during the implementation of map and reduces tasks**.** The execution of a Map Reduce job is divided into several time slots with a length of several minutes or an hour.

In this section, an online algorithm is used whose basic idea is to postpone the migration operation until the cumulative traffic cost exceeds a threshold.

The objective in this paper is to further minimize traffic cost of a Map Reduce job after the implementation of decomposition based distribution algorithm and online algorithm.

```
Step 1: t=1 and $\hat{t}$ = 1;
Step 2: solve the OPT_ONE_SHOT problem for t=1;
Step 3: while t≤T do
Step 4: if $\sum_{T=\hat{t}}^{t} \sum_{p\in P} \sum_{t}^{p}(T) > \gamma C_M(\hat{t})$ then
Step 5: solve the following optimization problem:
            min$\sum_{p\in P} C^p(t)$      for time slot t.
```

ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 5, Issue 4, April 2016

```
Step 6:  if the solution indicates a migration event then
Step 7:  conduct migration according to the new solution;
Step 8:  $\hat{t}$ = 1;
Step 9:  update $C_M(\hat{t})$
Step 10: end if
Step 11: end if
Step 12: t=t+1;
Step 13: end while
```

In the above algorithm t is the time slot. $C_M(\hat{t})$ denotes the total migration cost at $\hat{t}$ the time of last migration operation. The accumulative traffic cost $\sum_{T=\hat{t}}^{t}\sum_{p\in P}\sum_{t}^{p}$(T) is checked with number of times of $C_M(\hat{t})$. If it is lesser than the number of times of $C_M(\hat{t})$, the optimization problem with the objective of minimizing traffic cost is solved. The migration operation according to the results is conducted and $C_M(\hat{t})$ is updated.

IV. K-MEDOID CLUSTERING ALGORITHM

K- MEDOID algorithm is commonly known as PAM algorithm. PAM stands for "Partition Around Medoids". This is mainly used to find a series of objects called *Medoids* that are centrally located in clusters. Objects that are hesitantly defined as Medoids are located into a set S of *selected objects*.

If O is the set of objects then, U = O − S is the group of *unselected objects*.

The algorithm is made with a goal to minimize the average variation of objects to their contiguous selected object.

Equivalently, it can reduce the sum of the dissimilarities between object and their contiguous selected object.

The two phases of the algorithm:

(i) In the first phase, BUILD, a set of k objects are selected for an initial set S.

(ii) In the SWAP phase, one tries to develop the feature of the clustering by swap selected objects with unselected objects.

For each object *p* we maintain two numbers:

- D_p, the dissimilarity between *p* and the contiguous object in S, and
- E_p, the dissimilarity between *p* and the second closest object in S.

These numbers *must be reorganized every time when the sets* S *and* U *change*. Note that $Dj \le Ej$ and that have $p \in S$ if and only if $D_p = 0$.

The BUILD phase includes the following steps:

1. Initialize S by adding up to it an object for which the sum of the space to all other objects is nominal.

2. Consider an object $i \in U$ as a applicant for inclusion into the set of chosen objects.

3. For an object $j \in U - \{i\}$ compute D_j, the dissimilarity between j and the closest object in S.

4. If $D_j > d(i, j)$ object *j* will add to the decision to select object *i* (because the quality of the clustering may benefit);

let $C_{ji} = max\ \{D_j - d(j, i), 0\}$.

5. Compute the total obtained by adding i to S as $g_i = \sum_{j\in U} C_{ji}$

6. Choose the object *i* that maximizes g_i ;

let $S := S \cup \{i\}$ and $U = U - \{i\}$.

These steps are performed until k objects have been selected.

The second phase, SWAP, attempts to improve the set of selected objects and, therefore, to improve the feature of the clustering.

This is done by taking into account of all pairs $(i, h) \in S \times U$ and consists of computing the effect T_{ih} on the sum of dissimilarities between objects and the closest selected object caused by swapping *i* and *h*, i.e., by moving *i* from *S* to *U* and transferring h to from *U* to *S*.

The computation of T_{ih} involves the computation of the contribution K_{jih} of each object $j \in U - \{h\}$ to the swap of *i* and *h*. Note that it have either $d(i, j) > D_j$ or $(i, j) = D_j$.

1. K_{jih} is computed taking into account the following cases

(a) if $d(i, j) > D_j$, then two sub cases occur:

- if $d(j, h) \ge Dj$, then $K_{jih} = 0$;
- if $d(j, h) < Dj$, then $K_{jih} = d(j, h) - Dj$.

In both sub cases, $K_{jih} = min\{d(j, h) - Dj , 0\}$.

(b) if $d(i, j) = D_j$, it have two sub cases:

- if $d(j, h) < E_j$, where E_j is the difference between j and the second closest selected object, then $K_{jih} = d(j, h) - D_j$; note that K_{jih} can be either positive or negative.
- if $d(j, h) \ge E_j$, then $K_{jih} = E_j - D_j$; in this case $K_{jih} > 0$.

In each of the above sub cases we have

$K_{jih} = min\{d(j, h),E_j\} - D_j$.

2. Compute the total result of the swap as

$T_{ih} = \sum\{K_{jih} \mid j \in U\}$.

V. SYSTEM PROCESS

System Process starts with data uploading in which the input dataset is selected and loaded into Hadoop Distributed File Server. HDFS server performs parallel storage and processing. In the next stage dataset is partitioned using K-Medoid clustering. It is followed by task assignment in which the Data Center should be selected according to calculation and space capacity of servers resides in the data center. Then data loading takes place followed by Processing of task in which the population of data is processed depend upon the computational capability of servers reside in the data centers.

The following diagram shows the system architecture

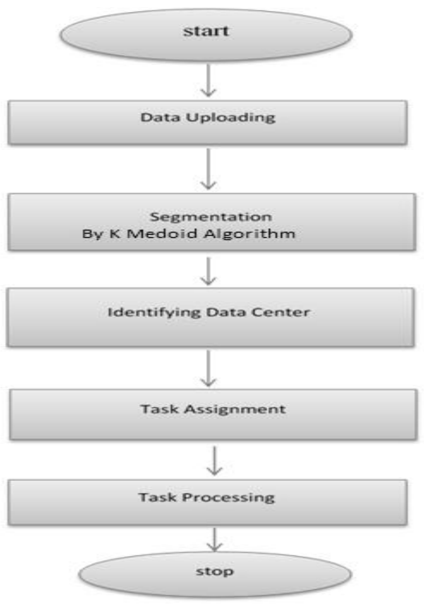

Fig 1: System Flow

VI. CONCLUSION

This paper deals with the implementation of K-MEDOID clustering algorithm in Decomposition based distribution algorithm in Map Reduce in order to optimize the time constraint and also an online algorithm is used to jointly aggregate the data in more efficient way. Using this method minimize network traffic cost can further be minimized for big data applications.

REFERENCES

[1] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: localityaware resource allocation for mapreduce in a cloud," in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis.

[2] S.-C. Hsueh, M.-Y. Lin, and Y.-C. Chiu, "A load-balanced mapreduce algorithm for blocking-based entity-resolution with multiple keys," Parallel and Distributed Computing 2014, p. 3, 2014.

[3] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi, "Leen: Locality/fairness-aware key partitioning for mapreduce in the cloud," in Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference.

[4] A. Blanca and S. W. Shin, "Optimizing network usage in mapreduce scheduling."

[5] L. Fan, B. Gao, X. Sun, F. Zhang, and Z. Liu, "Improving the load balance of mapreduce operations based on the key distribution of pairs," arXiv preprint arXiv:1401.0355, 2014.