A Load Balance Algorithm Based on Nodes Performance in Hadoop Cluster

Zhipeng Gao[1], Dangpeng Liu[1], Yang Yang[1]，Jingchen Zheng[2], Yuwen Hao[2]
1. Beijing University of Posts and Telecommunications, Beijing, China
2. General Hospital of Chinese People's Armed Police Forces, Beijing, China
E-mail: liudangpeng@126.com

***Abstract*—MapReduce is an important distributed programming model for large-scale data-parallel applications like web indexing, data mining, and scientific simulation. Hadoop is an open-source implementation of MapReduce and it is often applied to short jobs for which low response time is critical. When the cluster nodes are homogeneous, Hadoop has a good performance. In practice, the homogeneity assumptions do not always hold. In heterogeneous environment, there are various devices which vary greatly in the capacities of computation, communication, architectures, memories and power. When different nodes process the same amount of data, load balancing problem occurs. In this paper we address the problem of how to assign data after Map phase to balance the execution time of each Reduce task by proposing a novel load balancing algorithm based on nodes performance (LBNP), in which the input data of poor performance nodes are decreased. Simulation results indicate that all the Reduce tasks can be completed in the same time which shortens the whole Reduce phase. Thus the efficiency of MapReduce is improved**

***Keywords—MapReduce; Hadoop; Load balance; Heterogeneous environment; Nodes performance*;**

I. INTRODUCTION

Hadoop [1], [11] is an open-source software project that is licensed under Apache, which provides a distributed system framework. It makes the programmers who have no development experience in parallel and distribution can easily develop distributed applications. MapReduce [2] is the core architecture of Hadoop, which comes from Google's MapReduce model.

A MapReduce job execution can be divided into three phases: Map, Shuffle, and Reduce. In Map phase, Mapper (a java progress executing Map function) processes the input data and generates the input data into key/value pairs which will be transmitted into different Reducers (a java progress executing Reduce function) in Shuffle phase. In Reduce phase, each Reducer chooses a node to process key/value pairs and the results are finally written into HDFS [3]. In Map phase, the partition based on a hash partition function:

$$PartitionNum = Hash(key)\%\, ReduceNum \quad (1)$$

Where PartitionNum is the number of partition, and *ReduceNum* is the number of Reducer.

In this case, key/value pairs will be distributed to each Reducer equally. In homogeneous environment, the computing capacity of each node is alike. It is reasonable to distribute the output of Map tasks in Hash partition to Reduce tasks. But in heterogeneous environment, the nodes performance is quite different because the nodes vary in the capacities of computation, communication, architectures, memories and power. In this case, the poor performance nodes will spend more time in Reduce phase, and the execution time of the whole MapReduce job is prolonged, if we allocate equal amount of data to nodes by default hash partition.

In this paper, through a deep study of operational mechanism of MapReduce, focused on the skewed running time of Reduce phase caused by uneven nodes performance, we present a load balance algorithm based on nodes performance(LBNP) to balance the execute time of Reduce phase.

The rest of the paper is organized as follows. Section 2 describes the related work of MapReduce. Section 3 shows the overall model of LBNP. After that we present the details of LBNP in section 4. Simulation results are given in section 5, and the whole paper is concluded in section 6.

II. RELATED WORK

MapReduce Framework in Heterogeneous Environment: Increasing evidence shows that heterogeneity problems must be tackled in MapReduce frameworks [4], [5], [12]. Zaharia[6] implemented the LATE scheduler in Hadoop, which can improve MapReduce performance by speculatively executing tasks that hurt response time the most. Guo [7] proposed Benefit aware speculative execution which predicts the benefit of running new speculative tasks and greatly eliminates unnecessary runs.

Load Balancing of MapReduce: Load balancing is well-known data management problems and MapReduce has been criticized for having overlooked the skew issue [8], [13]. Smriti [9] proposed a load balancing model. It balances the workload using Key Chopping, to split keys with large loads into sub-keys that can be assigned to different distributive Reducers, and Key Packing, to assign keys with medium loads to Reducers to minimize the maximum Reducer load. Venkata [10] tacked the work load balancing issue by introducing a hierarchical MapReduce, which identifies a heavy task by a properly defined cost function. The heavy task is divided into child tasks that are distributed among available workers as a new job in MapReduce framework.

These researches have studied the MapReduce framework in heterogeneous environment or load balancing independently. Few studies can combine the two aspects, focused on the skewed workload caused by unbalance nodes performance in heterogeneous environment.

III. OVERALL MODEL

The main idea of LBNP is assigning data to Reduce tasks based on the performance of Tasktracker nodes which is running the Reduce task. Hadoop is based on the mechanism of heart beating. Only if there are free Reduce slots, Hadoop will assign tasks when Tasktracker reports heartbeat. So we need to make data pre-allocation before the allocation of Reduce tasks.

This paper is supported by NSFC (61272515, 61121061), Beijing Higher Education Young Elite Teacher Project (YETP0474), and NDRC High Tech Dept ([2013]-2140).

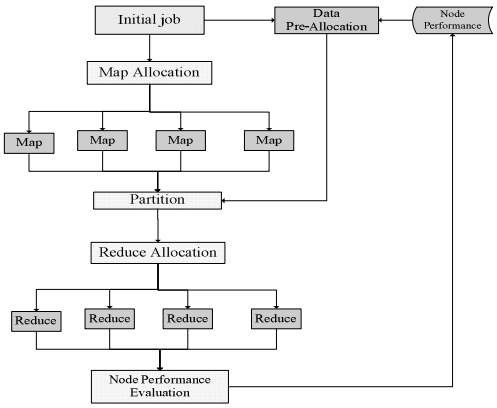

Figure 1. The overall model of LBNP

The overall model of LBNP is shown in Figure1.The first step is Initialization, In this phase, data pre allocation should be done. And a pre-allocation proportion list $RP_{ReduceNum}$ should be created based on the performance data of all nodes in the cluster. *ReduceNum* is the number of Reduce tasks configured by job. Every element in the $RP_{ReduceNum}$ is a proportion of data which the corresponding Reduce tasks should be allocated. In the phase of partition, data is distributed to the Reduce tasks based on the element in the $RP_{ReduceNum}$. Then Reduce tasks are assigned to different nodes to execute. At this time, different nodes need to carry out different number of data. If one node has a better performance, it will get more data. The performance of the Tasktracker nodes will be evaluated after the completion of all the Reduce tasks.

IV. LBNP ALGORITHM

In the part 3, we have introduced the overall model of LBNP. LBNP has four steps: data pre-allocation, adaptation of partition function, Reduce tasks allocation, and evaluation of nodes performance. In this part, we will explain the LBNP in detail.

A. Data Pre-allocation

TABLE 1 Notation and terminology

N	all the computing nodes in the cluster
N_i (i=1,2,3,...*NodeNum*)	the ith computing node
NodeNum	the number of available computing nodes in the cluster
N_i.*host*	host name of the *ith* computing node
N_i.*performance*	the performance of the *ith* computing node
ReduceNum	the number of Reduce tasks
PartitionNum	the partition number
SN	all the nodes are sorted according to performance
TL	a temp nodes list
$S = \{S_1, S_2, S_3...S_n\}$	*SN* is divided into n parts on average and S_i is *ith* part
r_i	the *ith* Reduce task
$P = \{p_1, p_2, p_3...p_n\}$	the proportion of data after pre-allocation
$< r_i, p_i >$	the relationship between the *ith* Reduce task and the *ith* pre-allocating data proportion which means the *ith* Reduce task should handle p_i data.

a) Nodes sort: All the element in the *N* should be sorted incrementally according to N_i.*performance*. Then we can get a new list *SN*.

b) Nodes division: *SN* should be divided according to the *NodeNum* and *ReduceNum*.

If *NodeNum* > *ReduceNum*, *SN* will be divided into *ReduceNum* parts equally. Those parts make a collection $S = \{S_1, S_2, S_3...S_n\}$ (n is *ReduceNum*), and S_i, S_j is subject to:

$$\begin{cases} \bigcup_k^1 S_i = SN \\ S_i \bigcap S_j = \varnothing \end{cases} 1 \le i \le n, 1 \le j \le n, i \ne j \tag{2}$$

$$Num(S_i) \in \left[\frac{Num(SN)}{ReduceNum}, \frac{Num(SN)}{ReduceNum} + 1 \right] \tag{3}$$

Where $Num(S_i)$ is the number of nodes in S_i. $Num(SN)$ is the number of nodes in *SN*.

If *NodeNum* <= *ReduceNum*, all the elements in the *SN* make a collection $S = \{S_1, S_2, S_3...S_n\}$ (n is *NodeNum*). S_i is corresponding to every node in the *SN*.

c) Data pre-allocation:

If *NodeNum* > *ReduceNum*, we should compute the average performance of every collection in the *S*. Then we can create the collection *P* based on the average performance.

$$r_i = \frac{Avg(S_i.performance)}{\sum_1^n Avg(S_k.performance)} \tag{4}$$

Where $Avg(S_i.performance)$ is the average performance of all nodes in S_i

If *NodeNum* <= *ReduceNum*, *TL* should be created according to *S*. And the number of elements in *TL* is *ReduceNum*.

$$TL_i = S_k.performance \quad (k = i\%NodeNum) \tag{5}$$

$$r_i = \frac{TL_i}{\sum_1^n TL_j} \tag{6}$$

B. Adaptation of partition function

Key/Value pairs will be distributed to Reduce tasks through partition function after the Map tasks finishes. The partition function can assign the equal key value to the same Reduce task based on the collection *P*. The pseudo-code of the adapted partition function is as follows.

Algorithm 1：partition function

Input： key: key value of input data. *ReduceNum*: the number of Reduce. P: P = {p_1,p_2,p_3…p_n}, the proportion of data after pre-allocation
Output： PartitionNum: PartitionNum corresponding to the Reduce task

```
1    TempNum ← Hash(key)%100
2    SumNum ← 0
3    foreach i in [0,ReduceNum-1]
4      if TempNum ∈ (SumNum,SumNum+p_i] then
5        PartitionNum = i
6        return PartitionNum
7      end if
8    end for
```

C. Allocation of Reduce tasks

In the data pre-allocation step, cluster and data are divided by performance. We get the nodes collection $S=\{S_1, S_2, S_3 \dots S_n\}$ and data proportion collection $P = \{p_1, p_2, p_3 \dots p_m\}$.The essence of the data pre-allocation is to contact nodes collection S_i with the corresponding proportion data p_i. For example, $<S_i, p_i>$ means nodes in S_i executing the proportion of data p_i. Because of $p_i=p_{i+k*NodeNum}$ $(k=0,1,2\dots and\ (i+k*NodeNum)<=ReduceNum)$, $<S_i, p_{i+k*NodeNum}>$ can be created.

In partition phase, all the data are divided into *ReduceNum* parts. Every part represents the data which different Reduce need to execute. The essence of partition is combining p_i and $PartitionNum_i$ up. For example, $<p_i, PartitionNum_i>$ means p_i proportion data should be assigned to r_i whose number is $PartitionNum_i$.

Algorithm2：allocation of Reduce tasks

Input： TaskTrackerStatus：Tasktracker's status.
S：S = {S_1，S_2，S_3 … S_K}, collection divided by performance. NonRunningReduce: Reduce tasks waiting to execute. IsRunning：a Boolean list corresponding to the collection in S，it means whether the proportion corresponding collection in S has been executed. ReduceNum: the number of Reduce configured by job. NodeNum: the number of nodes in the cluster.
Output： Tip：the index of elements in the NonRunningReduce，it means selecting tip satisfied conditions in the NonRunningReduce.

```
1   Index ← 0 //the index of Tasktracker in S
2   k ← 0 //the index of Reduce tasks
3   Flag ← 0 // look ahead or look behind
4   foreach i in [0,ReduceNum] do
5     if (TaskTrackerStatus.host ∈ Si) then
6       Index ← i
7       k ← Index
8       break
9     end if
10  end for
11  while(true) do
12    n ← 0
13    t ← k+n*NodeNum
14    while(t < ReduceNum)
15      if(!IsRunning_t) then
16        tip ← NonRunningReduce_t
17        return Tip
18      end if
19      n ← n+1
20      t ← k+n*NodeNum
21    end while
22    if(!flag) then
23      k ← k+1
24      if (k > ReduceNum-1) then
25        k ← index
26        flag = 1
27      end if
28    else
29      k ← k-1
30    end if
31  end while
```

After these two steps, nodes collection S_i is in contact with r_i. For example, $<S_i, r_{i+k*NodeNum}>$ means nodes in the S_i should run the Reduce task whose number is $i+k*NodeNum$. At the time of allocating Reduce tasks, the $r_{i+k*NodeNum}$ should be assigned to the nodes in S_i.

Specifically, when Tasktracker requests Reduce tasks, there are two steps to be executed. Firstly, we need to select S_i if it contains the Tasktracker. Secondly, we should select $r_{i+k*NodeNum}$ which corresponds to S_i and assign the task to that tasktracker. If all the tasks that S_i corresponds to have been executed, unexecuted tasks closest to the S_i will be selected. If all the tasks behind the S_i have been executed, unexecuted tasks recently in front of the S_i will be selected.

D. *Evaluation of nodes performance*

When all the Reduce tasks complete, we should compute the executing velocity of nodes that run Reduce tasks. The executing velocity v_i of *ith* Reduce task is computed as follows.

$$v_i = \frac{r_i}{t_i} \tag{7}$$

Where r_i is the data proportion of *ith* Reduce task, and t_i is the executing time of *ith* Reduce task.

All the nodes score is 100, when cluster is initialized. If node participates in executing Reduce task, the score will be renewed according to its executing velocity. We use the relative weight to compute score *P*. The score of the node that executes *ith* Reduce task is written as p_i and computed as follows.

$$p_i = \frac{v_i}{\sum_{1}^{n} v_k} \times 100 \times ReduceNum \tag{8}$$

Finally, the performance of all the nodes will be renewed effectively.

V. RESULT OF EXPERIMENT

We set up the experiment environment through virtual machines. All the virtual machines install REDHAT system using VMware. The version of JDK is 1.6.0.27 and Hadoop is 1.0.3 where the LBNP is carried out. The configuration of the environment is shown in the table2. Nodes in the Slave.Hadoop3-5 run a shell program of a read-write file to increase the load, so the performance variances between nodes can be bigger.

TABLE 2. The setting of simulation environment

name of host	Memory	cpu occupy proportion of other procedure
Master.Hadoop	2G	0
Slave.Hadoop1	2G	0
Slave.Hadoop2	2G	0%
Slave.Hadoop3	1G	60-80%
Slave.Hadoop4	500M	60-80%
Slave.Hadoop5	500M	60-80%

The performance testing process is the default WordCount. Testing data is created by RandomTextWriter. Assume that every Tasktracker has one Reduce slot. In the experiment, the number of Reduce tasks is set 5. To different data size, we will execute the WordCount repeatedly so as to make the state of work steady. We analyze the result from execution time of Reduce task and job, weight of data and performance.

A. *Execution time of Reduce tasks*

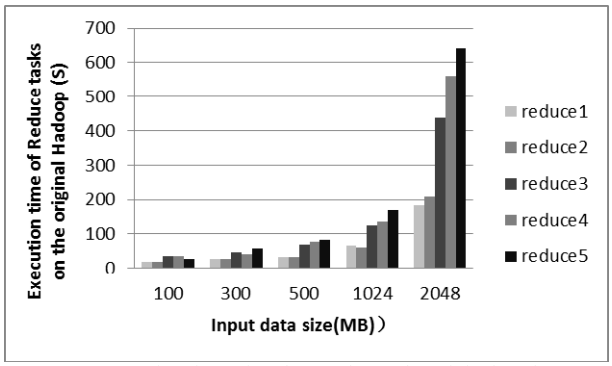

Figure 2. Execution time of Reduce tasks on the original Hadoop

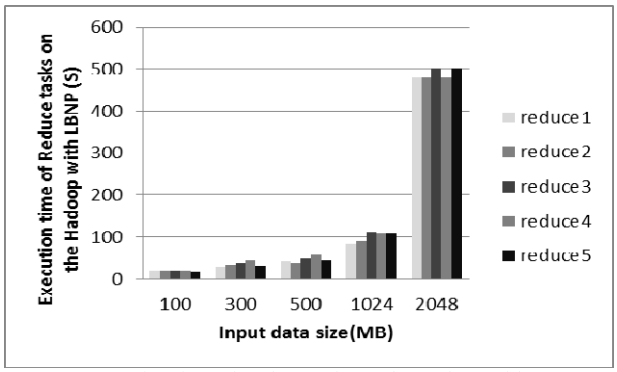

Figure 3. Execution time of Reduce tasks on the Hadoop with LBNP

Figure2 shows execution time of Reduce tasks on the original Hadoop. From the Figure, the performance of nodes that execute Reduce1 and Reduce2 is better, so the execution time is shorter while Reduce3-5's execution time is longer because of the poor nodes performance. Figure3 shows execution time of Reduce on the improved Hadoop through LBNP. From the Figure, the execution time of all the Reduce tasks is very close when tasks are distributed according to the performance. All the Reduce tasks can be finished in the same time which cuts down the whole executing time of Reduce phase. So the execution time of MapReduce job can be shortened.

B. Execution time of job：

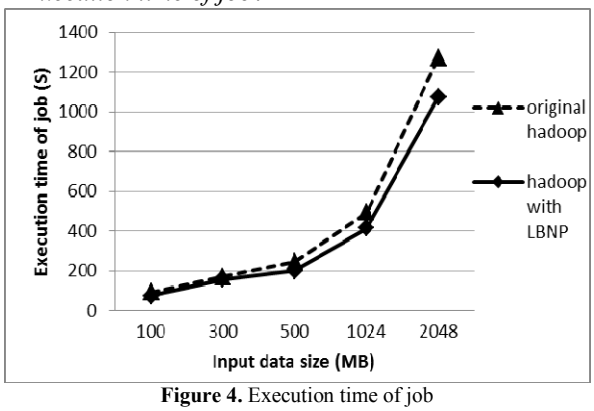

Figure 4. Execution time of job

Figure 4 shows the comparison of job's execution time between the original Hadoop and the improved Hadoop as the amount of data grows. From the figure, we can see job's execution time decreases clearly using the LBNP. And the improvement is obvious with the increase of data amount. The reason is LBNP can balance the load in the Reduce phase.

VI. CONCLUSION

In this paper, we propose LBNP: a load balance algorithm based on nodes performance, which uses historical information to evaluate the performance of nodes and assigns the tasks according to the nodes performance. Experimental results prove effectiveness of LBNP in decreasing the execution time of jobs and improving the efficiency of MapReduce in heterogeneous environments.

However, LBNP could be further improved from two aspects. On one hand, the evaluation algorithm of nodes' performance in LBNP is simple. A new reasonable evaluation algorithm should be designed to enhance the stability of new algorithm. On the other hand, LBNP will be evaluated on various platforms after first evaluated on virtual Hadoop cluster.

REFERENCES

[1] The Apache Hadoop project, http://Hadoop.apache.org/
[2] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," In Communications of the ACM, Volume 51, Issue 1, pp. 107-113, 2008.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
[3] WHITE, T. Hadoop: the Definitive Guide. O'Relly Mdeia, Inc.,2009.
[4] Rafique M M, Rose B, Butt A R, et al. Supporting MapReduce on large-scale asymmetric multi-core clusters[J]. ACM SIGOPS Operating Systems Review, 2009, 43(2): 25-34.
[5] Xie J, Yin S, Ruan X, et al. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters[C]//Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on. IEEE, 2010: 1-9.
[6] Zaharia M, Konwinski A, Joseph A D, et al. Improving MapReduce Performance in Heterogeneous Environments[C]//OSDI. 2008, 8(4): 7.
[7] Guo Z, Fox G. Improving MapReduce performance in heterogeneous network environments and resource utilization[C]//Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). IEEE Computer Society, 2012: 714-716.
[8] DeWitt D J, Naughton J F, Schneider D A, et al. Practical skew handling in parallel joins[M]. University of Wisconsin-Madison, Computer Sciences Department, 1992.
[9] Ramakrishnan S R, Swart G, Urmanov A. Balancing Reducer skew in MapReduce workloads using progressive sampling[C]//Proceedings of the Third ACM Symposium on Cloud Computing. ACM, 2012: 16.
[10] [Martha V S, Zhao W, Xu X. h-MapReduce: A Framework for Workload Balancing in MapReduce[C]//Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on. IEEE, 2013: 637-644.
[11] Venner J, Cyrus S. Pro Hadoop[M]. New York, NY: Apress, 2009.
[12] Rao B T, Sridevi N V, Reddy V K, et al. Performance issues of heterogeneous Hadoop clusters in cloud computing[J]. arXiv preprint arXiv:1207.0894, 2012.
[13] Gufler B, Augsten N, Reiser A, et al. Handing Data Skew in MapReduce[C]//Proceedings of the 1st International Conference on Cloud Computing and Services Science. 2011, 146: 574-583.