An Adaptive Scheduling Algorithm Based on Dynamic Workload Adjustment for Heterogeneous Hadoop Clusters

Nitali S. Dongare , Prof. Rekha Kulkarni
Department of Computer Engg,
PICT, Pune, Maharastra, India.
dongarenitali@gmail.com, kulkarni_rekha@live.com

Abstract— Hadoop is an open source Map-Reduce platform, which has been widely used for big data processing in large scale distributed system. The original task scheduling algorithm of Hadoop cannot cater to the performance requirements of heterogeneous clusters. According to the dynamic change of load of each slave node and the difference of node performance of different tasks in the heterogeneous Hadoop cluster, a dynamic workload adjustment algorithm (DWAA) is presented. With DWAA, tasktracker can adapt to the change of load at runtime and obtain task in accordance with its computing ability. The DWAA is a highly efficient and reliable algorithm, which can make heterogeneous Hadoop cluster stable and load balancing.

Keywords— dynamic workload, clustering method, computing ability.

INTRODUCTION

Cloud computing is a new business computing model for distributed computing tasks to a large-scale computing and storage resources pool that enable application systems to access a computing ability, storage space, and software service on demand [1], [2]. Cloud computing has the advantages of high scalability and high availability [3], many of the companies have released their own cloud computing platform to provide services to the public.

Hadoop [4] is an open source distributed architecture system sponsored by Apache software Foundation, which applies to structured and unstructured data search, and data mining. Hadoop is a highly efficient and reliable cloud computing platform, which can be deployed in a cloud computing data center [3]. Most of the famous Internet service providers, including Yahoo, Twitter, Facebook, and Baidu, have already chosen Hadoop as one of the core components to build their own cloud systems to provide more efficient services.

In order to complete the request submitted by user efficiently, Hadoop needs the right job scheduling algorithm and also a proper task scheduling algorithm. Job and task are two different concepts in Hadoop [3]. When a user submits a request, Hadoop will create a job and put it in the queue of jobs that are waiting to be executed by the job scheduler. After that, job will be divided into a sequence of tasks, which can be executed in parallel [5] [6]. With the help of task scheduling algorithm, the task scheduler dispatches the tasks on various nodes in the cluster. Several job scheduling algorithms have been developed in Hadoop, including first-in-first-out scheduling, fair scheduling, and capacity scheduling. The different types of tasks have different system resources, which lead to a huge difference of system load [3]. The original task scheduling strategies for Hadoop do not consider the difference of tasks and nodes. Most strategies cannot meet the performance requirements including stability, scalability, efficiency, and load balance etc. [7] [8]. Once Hadoop starts, the maximum number of tasks deployed on task-tracker at one time cannot be changed. Task-trackers are unable to dynamically adjust according to their own real-time load. This is a disadvantage of Hadoop, making it particularly unsuitable for heterogeneous cluster.

In this paper, we studied the traditional Hadoop cluster architecture and focused on task scheduling algorithm for dealing with heterogeneous cluster. The main contribution of this paper is to present a dynamic workload adjustment algorithm (DWAA). With DWAA, task-tracker can adapt to change load at run time and accepts the load in accordance with the computing ability of each node. The whole idea of DWAA is as follows: 1) First, to calculate the current load of each task-tracker and identify the amount of load created on the whole cluster. 2) Second, Examine if the load get increases than threshold value. 3) Change the load at run time and obtain the task based on the computing ability of each node.

The rest of this paper is organized as follows: In section II, the related work, system architecture of Hadoop and its task scheduling process are described. Section III explains DWAA in detail. Experimental and performance analysis are presented in Section IV. Section V concludes this paper by summarizing the main contribution of this paper.

II. RELATED WORK

Some of task scheduling strategies that have been proposed in recent years:
In 2009, M. Yong et al. [7] introduced task-tracker resource aware scheduler (TRAS). In this algorithm, each task-tracker collects its own resource information and then reports the same to the job-tracker for the next resource scheduling.

Reference [9] proposed a speculative task execution strategy (STES) in 2005. Even If the cluster has already finished most of the tasks, few of the tasks may be left due to insufficient hardware performance and become trailing task. In order to reduce the influence of trailing tasks on the whole job execution time, job-tracker will restart copies of trailing tasks running in parallel; once any one of the task is executed, the whole job is completed.

In 2013, Z. Tang et al. [10] showed an extensional MapReduce task scheduling algorithm for deadline constraints (MTSD), which allows user to specify a jobs deadline and makes it finished before the deadline.

In 2014, Xiao long Xu et al. [3] proposed adaptive task scheduling strategy based on dynamic workload adjustment algorithm for heterogeneous Hadoop clusters. In this algorithm, task-trackers can modify the change of load at run time and obtain tasks in accordance with the computing ability of each node.

A. Hadoop Architecture

Hadoop system contains two parts: Hadoop Distributed File System (HDFS) [11] and MapReduce [11]. An HDFS-based cluster has two types of nodes operating in the master-slave pattern i.e. name node called as master and data node called as slave node [3]. MapReduce is a programming model for data processing, which operates on both nodes i.e. job-tracker (master) and number of task-trackers (slaves). The job-tracker locates task-tracker nodes with available slots at or near the data and submits the work to the chosen task-tracker nodes. Task-trackers execute tasks and sends back progress report to the job-tracker, which keeps a record of the overall progress of each job. Hadoop divides the inputs of a MapReduce job into fixed-size pieces called input splits [3]. Hadoop creates one map task for each split. Then task-tracker runs the map tasks, whose output is processed by reducing task to produce the final output [5].

B. Task Scheduling Process of Hadoop System

The Hadoop system executes multiple tasks at the same time, so there is no surety with respect to effective task scheduling of the system within the entire cluster and execution is the premise to make full use of cluster resources.
1) Running Task: task-tracker counts current executing tasks that are assigned.
2) FixedTaskCapacity: Task-tracker determines whether the number of current executing tasks is less than the fixed number of slots for task or not. The fixed no of slots limit the number of tasks, which simultaneously run on a tasktracker.
3) AskForNewTask: It is a flag to indicate whether to obtain a new task or not. If running task is less than FixedTaskCapacity, then tasktracker could accept new tasks. The tasktracker will set flag to true, otherwise, to false.

The tasktracker periodically sends message of heartbeat via Remote Procedure Call (RPC) to the jobtracker. The default value of a period is 3 sec [12]. Tasktracker sends heartbeat menace it indicates whether it is ready to run a new task or not. The jobtracker does following work after receiving the heartbeat from tasktracker:1) The jobtracker first checks whether the last heartbeat response is completed or not; in case it is completed, it checks whether the flag AskForNewTask of heartbeat sent from the tasktracker is true or not. 2) If the flag is true, the jobtracker will use the task scheduler to assign the task to task list. If the flag is false, the jobtracker will not allocate a task to the tasktracker. 3) When jobtracker gets the task list, it will send it to tasktracker for execution.

C. Problem during the Task Execution

In heartbeat cycle, if a tasktracker wants to obtain task, then firstly it will determine whether the number of current executing task is less than the fixed number of slots or not.

The empty task slots = fixed number of slots - number of current executing tasks.

The load of each node dynamically changes in heterogeneous clusters and different nodes perform differently, which may give following problem:
1. Consider the tasktracker is a high-performance computing node and the number of current running task is equal to fixed number of slots, then the task tracker will not acquire the new tasks. At same time, if the tasktracker is still under loaded and it can run more tasks, then it is waste of ideal resources (hunger problem).
2. In second case, if the tasktracker is a low-performance computing node and number of current executing task is less than fixed number of slots, and then tasktracker can acquire new tasks. At the same time, if the tasktracker is heavily loaded, it cannot run more tasks, thus entering into saturation thereby leading to overloading.

III. DWAA

A. Main Idea

According to preceding analysis, the existed scheduling strategy of Hadoop cannot meet better performance. This paper proposes a novel load balancing algorithm, i.e. DWAA, for the heterogeneous Hadoop. The whole algorithm is working on running clusters. According to the tasktrackers own resources parameter, they can make corresponding adjustment to achieve the optimal state.

Each tasktracker periodically checks its load, which is based on collected parameters and dynamically adjusts the maximum number of slots for tasks. (MaxTaskCapacity), which replaces the approach of allocating the fixed number of slots for tasks (FixedTaskCapacity). Our strategy changes, each tasktracker measures its own workload in the next heartbeat, and then makes a decision about whether to allocate more tasks on same or not.

Submitting the request

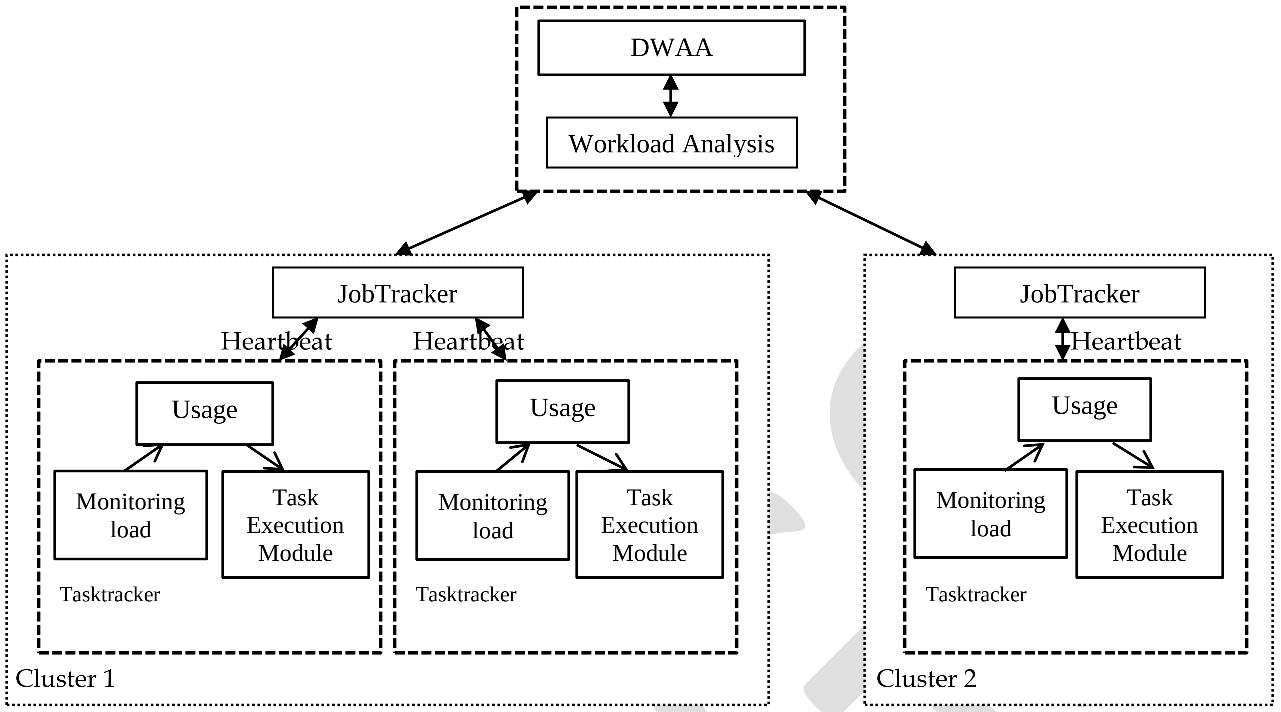

Fig. 1: Working environment of DWAA.

As Fig. 1 shows, the jobtracker deploying the tasks on tasktracker with the help of task scheduler. When jobtracker recives the heartbeat from tasktrackers, then it will assign the sutable load to the tasktracker. To efficiently access the load of tasktracker, DWAA needs to choose the appropriate load parameters for the evaluation.

The high CPU utilization rate indicates that CPU is unable to handle new task or is overloaded. Ideally, task scheduling moule will constantly get tasks executed in the task queue according to the sizes of the CPU time slice they obtain. If the task queue is too long, too many tasks will compete for resources, which makes CPU unable to process valuable tasks or response over a long period of time, resulting in the overloading state of CPU [13], [14]. Furthermore, the memory requirements are high for some applications, whereas their CPU requirements are minimum. DWAA is based on the average CPU utilization, average memory utilization, and the average number of task in queue.

B. Algorithm Description

Definition 1: In order to get the real-time information of perCpu, we can use related parameter from file /proc/stat of linux system to calculate perCpu. There are seven items that can be extracted from file /proc/stat as given: user-mode time (user), low-priority user-mode time (nice), system-mode time (sys), idle task-mode time (idle), hard disk I/O preparing time (iowait), servicing interrupts (irq), and servicing softirq (softirq).

$$perCpu = \frac{user+nice+sys}{total} \times 100 \qquad (1a)$$

$$total = user+nice+sys+idle+iowait+irq+softirq \qquad (1b)$$

The numerator is the execution time of non-system idle process and denominator is the total execution time of CPU. According to (1a)-(1b), we can get the current CPU utilization in real time.

Definition 2: The memory utilization is defined as perMem. In order to get the real-time information of perMem, we can use related parameter from file /proc/meminfo of linux system. There are four items that can be extracted from file /proc/meminfo as given: total memory size (MemTotal), free memory (MemFree), block-device buffers (Buffers), and file cache (Cached).

$$perMem = \frac{MemTotal-(Memfree+Buffer+Cache)}{MemTotal} \times 100 \qquad (2)$$

The following is the pseudocode of the algorithm DWWA.

```
DWWA
Input: Current tasktracker load information
        AvgClusterLoad: Calculate the average clusters CPU and Memory   load information.
        K: be the total number of node in cluster.
        $C_{th}$, $L_{th}$: threshold parameter.
        MaxTaskCapacity: total capacity of cluster to run task in parallel.
Output: AskForNewLoad, MaxTaskA, MaxTaskB
```

```
1. for (i=0; i<n; i++){
  /* Every cluster contains the n number of nodes, and   collect the set of information, deposit them in a corresponding load array. */
2. perCpu[i] = getCpuPercentageUsage
   /* Get CPU percentage utilization of each node */
3. perMem[i] = getMemPercentageUsage
  /*Get Memory percentage utilization of each node*/
4. }
5. for (j=0; j<n j++){
6. AvgperCpu = perCpu[j]
7. AvgperMem = perMem[j]
8. }
9. CpuInfo  = AvgperCpu/K
   /*K is the number of node in cluster*/
10. MemInfo = AvgperMem/K
11. if (CpuInfo < $C_{th}$ && MemInfo < $M_{th}$){ then
12. if (CpuInfoA > CpuInfoB ){
    /*Compare the CPU load between the clusters */
13. MaxTaskB =  MaxTaskB + no_of_task
     /*Assign the next tasks (no_of_task) on cluster B*/ }
14.  elsif (CpuInfoA < CpuInfoB ){ then
15.  MaxTaskA =  MaxTaskA + no_of_task }
16. else{
17. if (MemInfoA > MemInfoB){
    /*Compare the Memory load between the clusters */
18. MaxTaskB =  MaxTaskB + no_of_task }
19. elsif (MemInfoA < MemInfoB){
20. MaxTaskA =  MaxTaskA + no_of_task }
21. else{
22. if (MaxTaskCapacityA > MaxTaskCapacityB){
     /*Compare the Maximum task running capacity of   clusters */
23. MaxTaskA =  MaxTaskA + no_of_task }
24. else { MaxTaskB =  MaxTaskB + no_of_task }
25. else {
26. MaxTaskA =  MaxTaskA - 1
27. MaxTaskA =  MaxTaskA - 1 }
28. fi
29. fi
30. fi
31. fi
```

IV. EXPERIMENTAL AND PERFORMANCE ANALYSIS

A. Performance Indicator

1. The appropriate *Resource utilization rate* indicates the scheduling algorithm has well adaptable. It can increase the utilization of resource without losing efficiency and quality of service offered by the Hadoop system. However, increase the resource utilization rate does not improve the efficiency of system every time. Hence, the resource utilization rate is given to evaluate the systems adaptability.
2. The *Speedup ratio* indicates the performance improvement rate of the parallel computing ability with the scale up of system when processing the task. Consider the system's scale increases from S_1 to S_2 and time completing the same task decrease from T_1 to T_2. Then speedup ratio as given below:

$$SR = | T_1 - T_2|/| S_1 - S_2| \qquad (6)$$

B. Experimental Environment

TABLE I
Configuration Parameter of Nodes

Node No.	CPU	Memory
1.	Intel®Core^{TM}i5-2310 CPU@2.90GH$_Z$×4	7.8GiB
2.	Intel®Core^{TM}i3-2100 CPU@3.10GH$_Z$×2	3.8GiB
3.	Intel®Core^{TM}i3-2100 CPU@3.10GH$_Z$×4	3.8GiB

TABLE II
Configuration Parameter of Hadoop

Parameter	Default value
dfs.replication	2
dfs.block.size	64MB
dfs. Heartbeat.interval	3s

C. Experimental Results

We compile the source code based on Hadoop-1.2.1 with Eclipse and create the jar (job) using maven. The all jar deployed on all nodes in both cluster.

TABLE III
Comparison Results

Algorithm	Files (MB)	1st run (s)	2nd run (s)	3rd run (s)	Average Time (s)
Original	1160.2	372	350	302	341.33
DWAA	1160.2	222	312	211	248.33
Original	2165.71	484	444.5	515	481.16
DWAA	2165.71	318	350	478	382
Original	3016.53	513	792	645	650
DWAA	3016.53	331	570	463	454.66

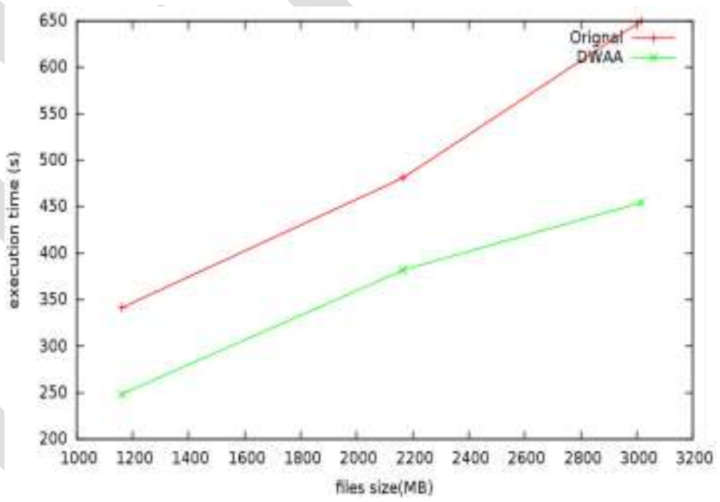

Fig 2: Experimental result

V. CONCLUSION

Scheduling is a major factor for task execution in cloud environment for better performance. In this paper, we identify the problem of load balancing of Hadoop system, and proposed new DWAA algorithm. DWAA is highly efficient algorithm and reliable for heterogeneous cluster. With the help of this algorithm, the performance of node can become more stable and failure rate of task execution can be decreases on tasktracker side. The hunger and saturation problem of task execution can be avoided at the same time. On the jobtracker side, the failure and bottleneck due to overloading can be avoided.

REFERENCES:

1. M. Armbrust et al., “A view of cloud computing”, Commun. ACM, vol. 53, no. 4, pp. 50-58, Apr. 2010.

2. P. Mell and T. Grance, The NIST Definition of Cloud Computing (Draft):

3. Xaiolong Xu et al., “Adaptive task scheduling strategy based on dynamic workload adjustment for heterogeneous Hadoop clusters”, IEEE system journal, 2014.

4. Apache (2012, Aug). Hadoop, The Apache Software Foundation, Forrest Hill, MD, USA. Available: http://hadoop.apache.org/

5. J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large cluster”, Commun. ACM, vol. 51, no. 1, pp. 107-113, jan. 2008.

6. T. White, “Hadoop: The Definitive Guide”, Sebastopol, CA, USA:O’Reilly Media, 2012, pp. 167-188.

7. M. Yong, N. Garegrat, and S. Mohan, “Towards a resource aware scheduler in Hadoop”, in Proc. ICWS, 2009, pp. 167-188.

8. Y. H. Wu, “Research of scheduling policies in cloud computing”, M. S. thesis, College Comput, Shanghai Jiao Tong Univ., Shanghai, China, 2011.

9. E. B. Nightingale, P. M. Chen, J. Flinn, “Speculative execution in a distributed file system,” ACM SIGOPS Oper, Syst. Rev., vol. 39, no. 5, pp. 191-205, Dec. 2005.

10. Z. Tang, J. Q. Zhou, K. L. Li, “A MapReduce task scheduling algorithm for deadline constraints,” Cluster Compute., vol. 16, no. 4, pp. 651-662, Dec. 2013.

11. Apache. (2013, Apr). MapReduce tutorial, The Apache Software Foundation, Forrest Hill, MD, USA [online]. Available: http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

12. K. Sims, IBM introduces ready-to-use cloud computing collaboration services get clients started with cloud computing, IBM corp., Armonk, NY, USA.[online]. Available:http://www.3.ibm.com/press/us/en/pressrelease/22613.was

13. A. Yuan, LoadAverage.21cn.com, Feb. 2012. [Online]. Available:http://edu.21cn.com/java/g_189_786430-.htm

14. R.Walker, Examining Load Average. Linux Journal, Dec. 2006. [Online]. Available: http://www.linuxjournal.com/article/9001