On Demand Resource Scheduler Based on Estimating Progress of Jobs in Hadoop

Liangzhang Chen, Jie Xu, Kai Li, Zhonghao Lu, Qi Qi(✉), and Jingyu Wang

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, People's Republic of China
{luzhonghao, xujie, qiqi, wangjingyu}@ebupt.com

Abstract. In order to meet the need of setting deadline for Hadoop MapReduce job and improve resource utilization of Hadoop cluster, a resource scheduler based on collecting the running information of tasks is proposed. According to the information of resource usage, the progress of job, the deadline of job, and the handling time of job, we estimate the resource demand of jobs, and then schedule these jobs according to their resource demand. Meanwhile, a method to judge whether the resource of cluster can meet the deadline of all the jobs in cluster is proposed. When the jobs will miss the deadline under the allocated resources, scheduler applies to cloud platform for extra resources. Experimental results show the on demand resource scheduler can increase the utilization of resource in Hadoop cluster and approximately ensure the deadline of jobs.

Keywords: Hadoop · On demand · Scheduler · Job with deadline

1 Introduction

The Cloud Computing provides a virtualization of hardware and software resource pool to a variety of service for users in the network platform which makes it possible to store and compute the huge amounts of data.

At present, many cloud platforms provide data computing services, one of the most commonly used is the Hadoop [1] which is an open source project under the Apache organization. Hadoop is a distributed parallel programming framework for huge data processing and data-analysis. It can be deployed in the cloud platform resource pool and mainly focus on the huge amounts of data storage and computing.

But it is essential to use a resource manager to coordinate and schedule the Hadoop cluster resources. Yet Another Resource Negotiator (Yarn) is commonly used as a resource manager. There are three kinds of schedulers, simple First-In-First-Out (FIFO) Scheduler, Capacity Scheduler [2] and Hadoop Fair Scheduler (HFS) [3], which can only schedule resource based on the strategy of "FIFO" in each queue and be unaware of the Running information of jobs in Hadoop.

As users always setting deadline for Hadoop MapReduce job, it requires the scheduler has the ability to allocate resource according to the job's deadline. But now all the schedulers for Hadoop cannot allocate resource in terms of the job deadline. In

S. Wang and A. Zhou (Eds.): CollaborateCom 2016, LNICST 201, pp. 615–626, 2017.
DOI: 10.1007/978-3-319-59288-6_62

this paper, we propose an on demand scheduler for Hadoop MapReduce job with deadline. For a given MapReduce job with deadline, the scheduler tries its best to allocate enough resources to meet the deadline of job. Meanwhile, this new kind of scheduler can judge whether the resource of cluster can meet the deadline of all the jobs in cluster. If not, the scheduler should request resources from Cloud platform to make all the jobs in cluster finished before deadline. This paper makes the following contributions.

A strategy to estimate the progress of Hadoop MapReduce job is proposed, which is the basis of on demand scheduling and resource judging strategy. The strategy uses the information of map tasks, shuffle, reduce tasks.

A concept of on demand resource scheduling is included, which describes the demand for resource. The resource demand is related to the deadline of job, the progress of job and the used resources of job.

An algorithm to judge whether the resources of cluster can meet the deadline of all the jobs in cluster is presented. This algorithm uses the information of resource demand, deadline, cluster resources and so on. In this algorithm, we also introduce the concept of work. When this algorithm judges cluster resource is not enough, it requests resources from Cloud platform.

The rest of paper is organized as follows. Section 2 reviews the related work. Section 3 gives details on the model of on demand resource scheduler. Section 4 shows the experimental results. Section 5 concludes the paper.

2 Related Work

In recent years, many researchers attempt to improve resource utilization of Hadoop cluster.

Polo *et al*. [4] introduce a new task scheduler that dynamically collects the performance data of MapReduce jobs and adjusts the resource allocation accordingly. But they only focus on the map phase and have no control over the reduce phase. Jockey [5] is designed for single job to maximize its economic utility. So Jockey is effective at guaranteeing job latency and minimizing the impact on the data parallel clusters. But it lacks scheduling mechanism among multiple jobs. A. Verma [6] proposes a framework, called ARIA, to estimate and allocate appropriate number of map and reduce slots for MapReduce applications so that they can meet their required deadlines. However, all of these studies only support resource inference and allocation and give no consideration to overtime budget and energy cost.

Gunho Lee *et al*. [7] present a architecture to allocate resources to a data analytical cluster in the cloud, and a scheduling scheme that uses progress share as a new fairness metric to fit the heterogeneous environment. In their resource allocation strategy, they actually divided nodes into two pools to reduce the cost brought by fluctuating resource demands of data analytical workloads. In our work, we also propose a multi-layer node model to reduce the cost of scale process and increase the flexibility of the cluster maintained by the datacenter. However, we propose an auto-scaling algorithm to automatically adjust each node layer instead of using only static threshold and single indicator which may cause fluctuation.

Wei Zhang *et al*. [8] refine the scheduler of virtualization layer to minimize interference on high priority interactive services. Toomas *et al*. [9] also propose a method named auto-scaling in Hadoop clusters, but their auto-scaling is only based on the workload of the Hadoop cluster which won't apply to various services environments. There are also efforts to study heterogeneous clusters in [10, 11]. These authors address the poor performance of MapReduce on heterogeneous clusters and propose a heterogeneous aware scheduling scheme. Although their work improves the performance of heterogeneous cloud environments, the auto-scaling algorithm in our paper pays more attention to the heterogeneous resource demands rather than complex physical environments.

Dachen Zhang [12] designs a model to meet the deadline of all jobs with limited resources, while we focus on how to provide resources for urgent jobs that will meet the deadline in a short time.

3 Model Description

In this section, we introduce the details of on demand resource scheduler. The scheduler consists of three functions which includes estimating progress of MapReduce jobs, scheduling resources and judging whether the resources of cluster are enough. Figure 1 shows the whole process.

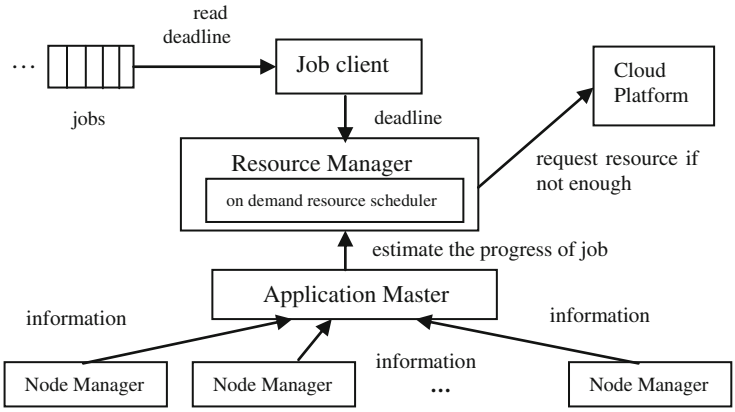

Fig. 1. The process of on demand resource scheduler

We can see that the Job Client read the deadline firstly, and then send it to the Resource Manager. Application Master collects the running information of Node Manager and sends it to Resource Manager. Resource Manager schedules the resource according to the strategy we propose, and judges whether cluster resource is enough. If cluster resource is not enough, it requests resource from the cloud platform.

3.1 Estimate Progress of MapReduce Job

We analyze the execution of the Hadoop MapReduce job firstly. According to the framework of Hadoop, MapReduce job is divided into many map tasks and reduce tasks, these tasks can be done on different nodes in cluster concurrently. In these tasks, map tasks focus on processing the split of original data and put the intermediate results on local nodes. These intermediate results will be transformed to the corresponding reduce tasks after the process of shuffle. Finally, reduce tasks produce the final result. In sum, the progress of a job consists of three parts, map, reduce and shuffle. Fig. 2 shows the specific execution of the Hadoop MapReduce job.

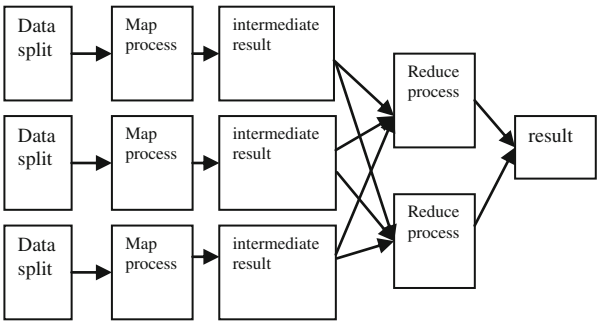

Fig. 2. The specific execution of the Hadoop

We use a linear Eq. (1) to fit the progress of MapReduce job.

$$p = \alpha p_{map} + \beta p_{red} + \gamma p_{sf} \tag{1}$$

Where p is the progress of job, p_{map} is the progress of all map tasks, p_{red} is the progress of all reduce tasks,p_{sf} is the progress of shuffle and α, β, γ are parameters.

Naturally, we need to estimate the handling time of each map task and the handling time of each reduce task in order to get the progress of general Map task and the progress of general Reduce task.

The handling time of map task and reduce task can be estimated by computing the average handling time of corresponding completed tasks. We can figure out the progress of map task at a cluster node over Eq. (2) when the nodes don't have failure. And if the node or job fails, the corresponding value is equal to 0.

$$p_{mapi} = \frac{t_{mapruni}}{t_{mapave}} \tag{2}$$

Where p_{mapi} is the progress of i-th map task, $t_{mapruni}$ is the time of the i-th map task has run for. t_{mapave} is the average time that completed map task used. The progress of a reduce task at a cluster node can be estimated by Eq. (3). Similarly, if the node or job fails, the corresponding value is equal to 0.

$$p_{redi} = \frac{t_{redruni}}{t_{redave}} \tag{3}$$

Where p_{redi} is the progress of i-th reduce task, $t_{redruni}$ is the time of the i-th reduce task has run for, t_{redave} is the average time that completed reduce task used.

The average progress of all map tasks can be estimated by Eq. (4).

$$p_{map} = \frac{\sum_{i=1}^{n} p_{mapi}}{n} \tag{4}$$

Where p_{map} is the average progress of all map tasks, p_{mapi} is the progress of i-th map task. n is the number of map tasks in cluster.

The average progress of all reduce tasks can be estimated by Eq. (5).

$$p_{red} = \frac{\sum_{i=1}^{m} p_{redi}}{m} \tag{5}$$

Where p_{red} is the average progress of all reduce tasks, p_{redi} is the progress of i-th reduce task and m is the number of reduce tasks in cluster.

The time of shuffle is the period of time from the end of map task to the start of reduce task. In this period, the data is transmitted from map task to reduce task. So we can approximately think that the start of a reduce task means that a process of shuffle has been finished. So the execution progress of shuffle can be estimated by Eq. (6).

$$p_{sf} = \frac{n_{redrun}}{n_{redtot}} \tag{6}$$

Where p_{sf} is the progress of shuffle, n_{redrun} is the number of reduce tasks has been launched and n_{redtot} is the number of all reduce tasks in cluster.

After many tests for this mathematical model, we can get optimized parameters. That is $\alpha = 0.45, \beta = 0.45, \gamma = 0.1$. Because the size of data that map task need to process is equal to the reduce task, so α is equal to β. It should be noted that these values are obtained with wordcount jobs. So the specific values may not be suitable for other types of job.

3.2 Strategy of Scheduler

At present, several existing schedulers are all based on the strategy of "FIFO". These strategies can't assign resource to the job which has urgent need for resource in time, but only assign resource based on the order of jobs in the queue. If a scheduler can assign resource reasonably with the information of running job, it is as far as possible to ensure that all jobs in cluster can be finished in its deadline.

We can simplify the problem. When the scheduler allocates resource, it allocates resource to the job with highest resource demand. The resource demand is an abstract

concept. It describes the extent of demand for resource. The resource demand is related to the deadline of job, the progress of job and the used resources of job. The shorter job's deadline is, the higher the resource demand of job is. The less resource job has used, the higher the resource demand of job is. The smaller progress of job is, the higher the resource demand of job is.

In order to use specific value to describe resource demand, we need to make the following assumptions and analysis. We regard the resource that a map task or a reduce task uses as a unit resource. Because the running speed of job which has two child tasks is two times as fast as the job which has only one child task and each child task has equal resource, the running speed of job is proportional to the numbers of unit resource. On the premise of this conclusion, we have the following analysis: we set the unit running speed of job is s, the number of unit resource the job has is m, the handling time of job is t_1 and the progress of job is p. We can get Eq. (7).

$$smt_1 = p \tag{7}$$

The job can exactly be finished at the deadline t2, if the job can get n unit resource at present. We can get Eq. (8).

$$s(m+n)(t_2 - t_1) = 1 - p \tag{8}$$

We can get Eq. (9) according to the Eq. (7).

$$s = \frac{p}{mt_1} \tag{9}$$

We can get Eq. (10) according to the Eqs. (8) and (9).

$$n = \frac{(1-p)mt_1}{p(t_2 - t_1)} - m \tag{10}$$

We define the resource demand R is the number of unit resource n divided by the rest of the time to job's deadline t2–t1. We can get Eq. (11).

$$R = \frac{n}{t_2 - t_1} = \frac{(1-p)mt_1}{p(t_2 - t_1)^2} - \frac{m}{t_2 - t_1} \tag{11}$$

3.3 Design Algorithm to Judge Whether the Resource of Cluster Are Enough

The resources of cluster may not meet the deadline of all the jobs in cluster. We need to design an algorithm to judge whether we should request resources from cloud platform.

The algorithm that judges whether the resources of cluster are enough is related to many factors. For example, the size of cluster resource, the number of jobs in cluster,

the size of data in each job, the deadline of each job, the current progress of each job, the size of resource each job has used and so on.

In order to analysis this problem, we can see a simple example first. We assume that there are three jobs in cluster. The current number of unit resource that the first job has used is 0, there is 1 unit time from current time to the first job's deadline. The first job can exactly be finished in deadline if first job gets 2 unit resource at present. The current number of unit resource that the second job has used is 0, there are 2 unit times from current time to the second job's deadline. The second job can exactly be finished in deadline if second job gets 2 unit resource at present. The current number of unit resource that the third job has used is 0, there are 3 unit times from current time to the third job's deadline. The third job can exactly be finished in deadline if third job gets 1 unit resource at present. There are three unit resource in cluster at present. Fig. 3 shows the problem. The length of rectangle is the deadline of job, the width of rectangle is resource that job should get at present, the area of rectangle is total remain work of job.

To judge whether the cluster resources can meet the deadline of the three applications, we introduce a concept of work firstly. We define a work for the progress of

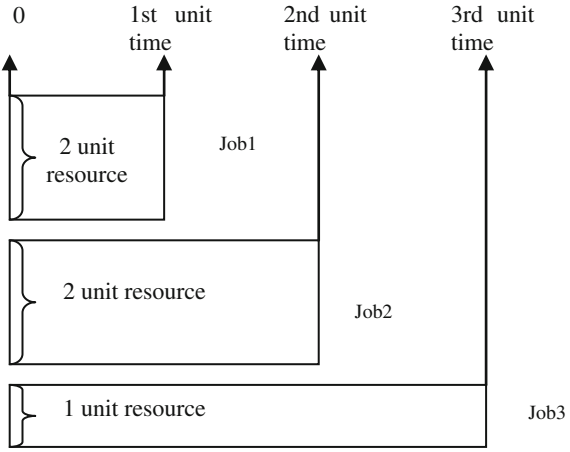

Fig. 3. Describe of the example

job which runs for a unit time using a unit resource. Because work is proportional to the running time, we set the value of work equal to the product of the size of resource and the length of running time. In the beginning, we can assign 2 unit resource to the first job and assign 1 unit resource to the second job. The first job will be finished and the second job will finish 1*1 = 1work at the point of first unit time. At this time, we assign 2 unit resource which the first job frees to the second job. when coming at the point of second unit time, the second job has finished 1*1 + 3*1 = 4work equal to the total work of the second job 2*2 = 4, so the second job can exactly be finished at its deadline. At this time, we assign 3 unit resource which the second job frees to the third job. when coming at the point of third unit time, the third job has finished 3*1 = 3work equal to the total work of the third job 1*3 = 3, so the third job can also exactly be

finished at its deadline. Through the above analysis, we can see that the cluster resources can exactly meet the deadline of three jobs in cluster.

Then we make a general analysis. The number of jobs in cluster is n and we sort the n jobs in ascending order of deadline. The i-th job can exactly be finished in deadline, if it get $m_1[i]$ unit resource at present. The i-th job has used $m_2[i]$ unit resource at present. There are t[i] unit time from current time to the deadline of the i-th job, the remain unit resource in cluster is R_m unit resource at present. Based on the rule of Eq. (12), we do n times to judge. If n times of judgement all meet the Eq. (12), the cluster resources are enough to meet the deadline of all jobs in cluster. On the contrary, it should require resource for the cloud platform.

$$\sum_{i=1}^{k} m_1[i] * t[i] \leq R_m * t[k] + \sum_{i=1}^{k} m_2[i] * (t[k] - t[i])(1 \leq k \leq n) \tag{12}$$

4 Experiment and Result

In this section, we evaluate on demand resource scheduler performance using word-count jobs in Hadoop 2.6. The whole cluster is deployed in a physical server, and nodes are virtual machines (VM).

First is whether cluster can request resources from cloud platform. Simply, we count the number of VMs as a symbol of requesting resources. Each VM is representative for requesting resources once. We take two experiments to verify the relationship between the size of resource scheduler requires and the size of job's data, the deadline of job. The first experiment is to make the job's deadline always equal to 2 min and increase the size of job's data. The second experiment is to make the size of job's data always equal to 16 G and increase the deadline of job. The Figs. 4 and 5 show the results. From the two forms, we can see that the more the size of job's data is, the more resource the scheduler requires. The shorter job's deadline is, the more resource the scheduler requires.

Second is about deadline and on demand resource scheduler is called as Adaptive Scheduler. We test the improvement on the job with deadline. We use two schedulers, Adaptive Scheduler is scheduled to meet deadline at utmost, and Capacity Scheduler follows the rule of FIFO. The whole cluster of this experiment has 5 child nodes, each node has 2 G memory and 2.2 MHz. 18 jobs with different deadline are submitted to the cluster. We set 2 min as the deadline for the former 14 jobs while the rest is 3 min. Actual time the job used and job's deadline are recorded and displayed in Figs. 6, 7, 8 and 9. In Fig. 6, we can see the time cost between Adaptive Scheduler and Capacity Scheduler. It is obvious that the time cost can be reduced 30% at least in most jobs. As shown in Figs. 7 and 8, Adaptive Scheduler is better than Capacity Scheduler. There are almost 28% jobs which can be finished before its deadline. But all jobs in Capacity Scheduler are finished after the deadline.

Because the deadline is not the same, we use data normalization to the time cost based on deadline. We compare the degree of improvement using Adaptive Scheduler

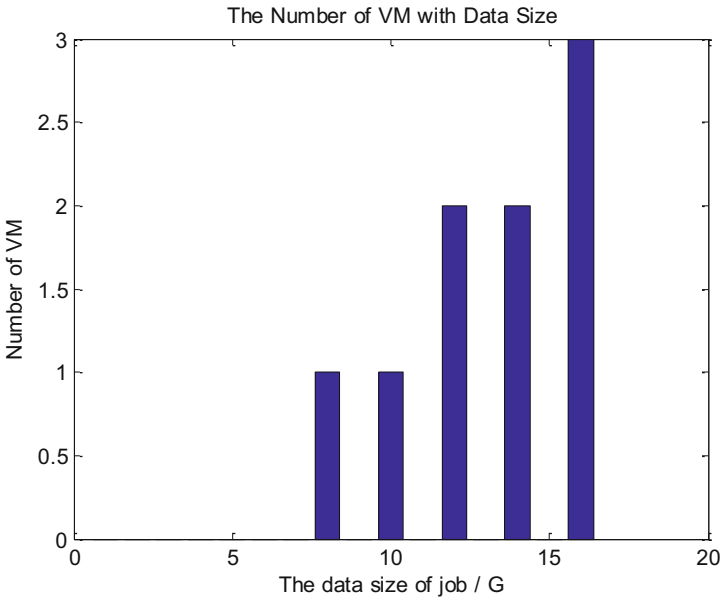

Fig. 4. The number of VM with data size

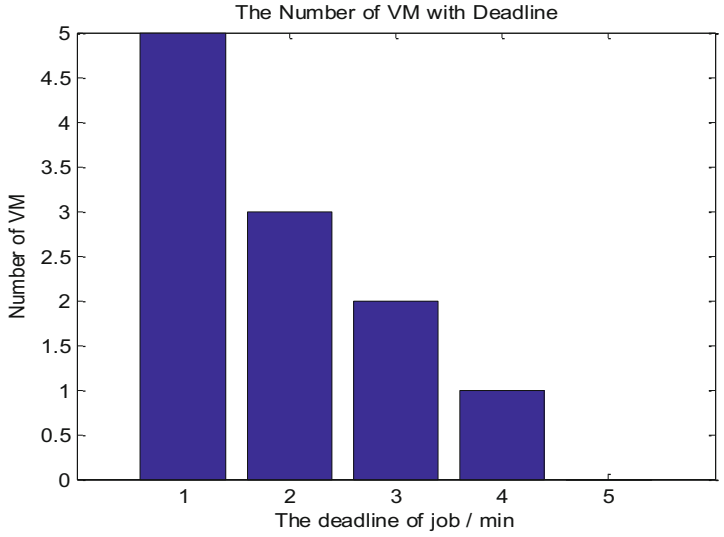

Fig. 5. The number of VM with deadline

and Capacity Scheduler. The ratio of deviation to deadline are showed in Fig. 9, where a positive number represents the actual time job used is less than job's deadline in vertical axis. The deviation of Adaptive Scheduler is less than 1, and all jobs can be finished before double deadline. The out part of the deadline is from requiring

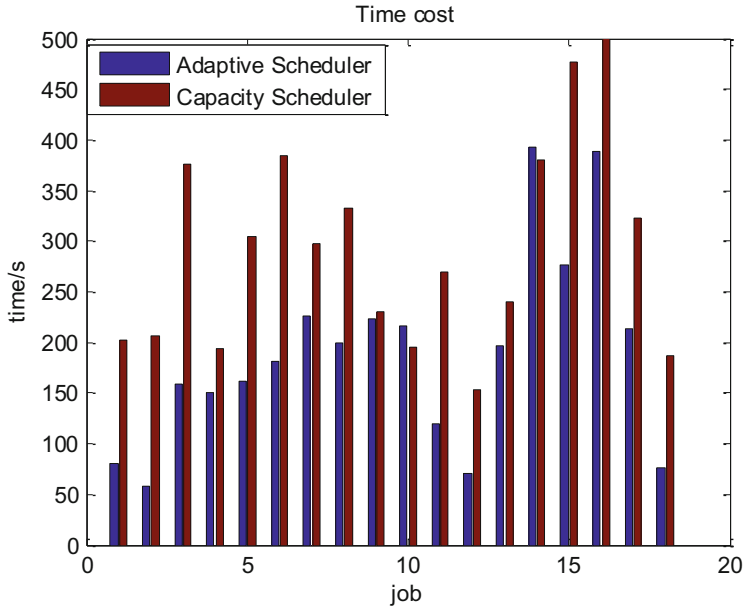

Fig. 6. Time cost of scheduler

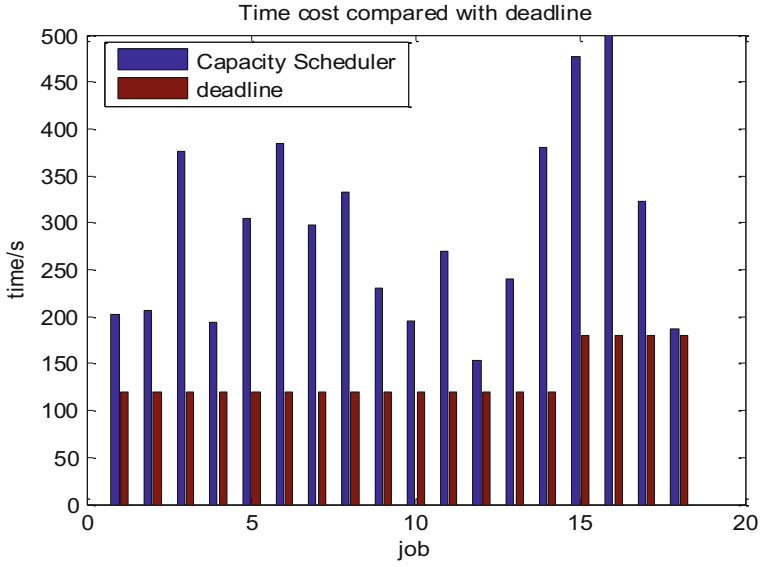

Fig. 7. Time cost compared with deadline of capacity scheduler

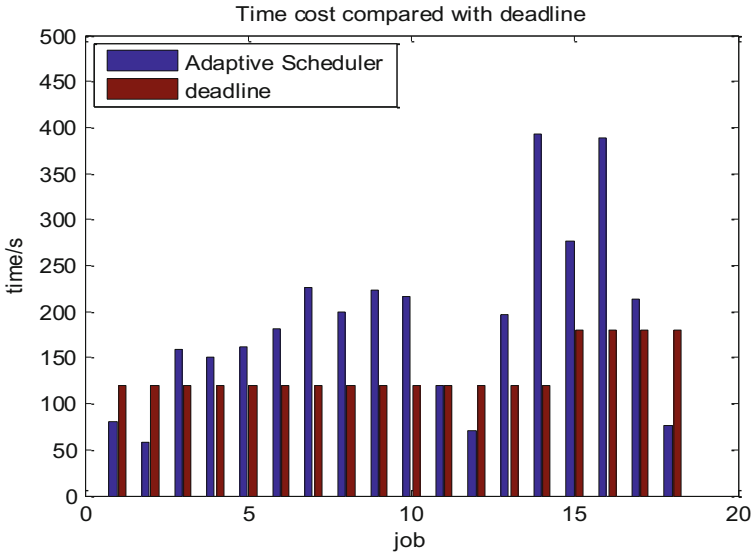

Fig. 8. Time cost compared with deadline of adaptive scheduler

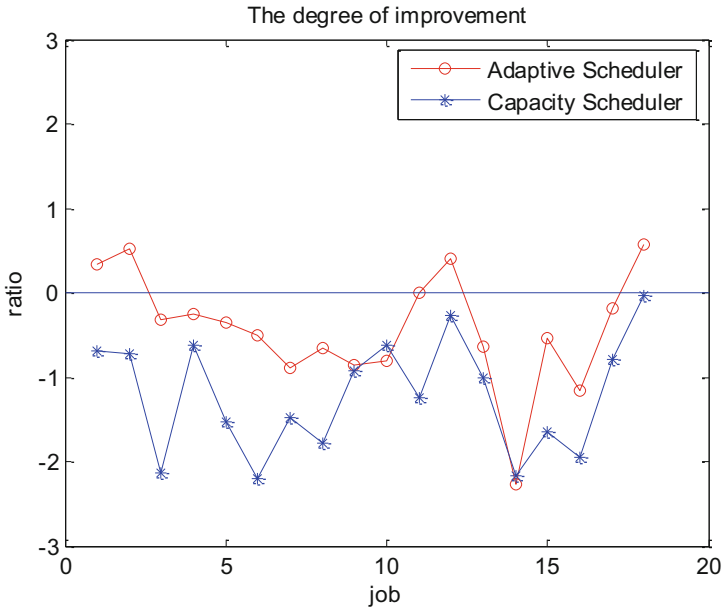

Fig. 9. The degree of improvement

resources which needs time. Therefore some jobs cannot be finished in its deadline. While Capacity Scheduler is more than double time, even up to threefold time or higher without time of requiring resources.

5 Conclusion

In this paper, we come up with on demand resource scheduler, which is aware of jobs' deadline and tries to guarantee the deadline by requesting resources from cloud platform. And results show the scheduler is useful for jobs with deadline. Even the model is not perfect, there must be altered to a more perfect model to make jobs meet the deadline. As more users submit jobs with its deadline, cloud platform still allocates fixed resources according to users' requirement. This mode can not match deadline with the amount of fixed resources. On demand resource scheduler can give a solution to this problem.

References

1. ApacheHadoop. http://hadoop.apache.org/
2. Capacity Scheduler. http://hadoop.apache.org/docs/r0.20.205.0/fairscheduler.html
3. Fair Scheduler. http://hadoop.apache.org/docs/r0.20.205.0/fairscheduler.html
4. Polo, J., Carrera, D., Becerra, Y., et al.: Performance-driven task co-scheduling for MapReduce environments. In: Network Operations and Management Symposium, pp. 373–380. IEEE (2010)
5. Ferguson, A.D., Bodik, P., Kandula, S., et al.: Jockey: guaranteed job latency in data parallel clusters. In: Eurosys Proceedings of the European Conference on Computer Systems, pp. 99–112 (2012)
6. Verma, A., Cherkasova, L., Campbell, R.H., et al.: ARIA: automatic resource inference and allocation for MapReduce environments. In: International Conference on Autonomic Computing (2011)
7. Lee, G., Chun, B., Katz, H., et al.: Heterogeneity-aware resource allocation and scheduling in the cloud. In: Cloud Computing (2011)
8. Zhang, W., Rajasekaran, S., Wood, T., et al.: MIMP: Deadline and interference aware scheduling of hadoop virtual machines. In: Cluster Computing and the Grid (2014)
9. Romer, T.: Autoscaling Hadoop Clusters. Master's thesis of University of Tartu faculty of mathematics and computer science (2010)
10. Rao, B.T., Sridevi, N.V., Reddy, V.K., et al.: Performance issues of heterogeneous Hadoop clusters in cloud computing. Comput. Sci. (2012)
11. Ahmad, F., Chakradhar, S.T., Raghunathan, A., et al.: Tarazu: optimizing MapReduce on heterogeneous clusters. ACM Sigarch Comput. Archit. News **40**(1), 61–74 (2012)
12. Cheng, D., Rao, J., Jiang, C.J., et al.: Resource and deadline-aware job scheduling in dynamic hadoop clusters, pp. 956–965 (2015)