An Improved Speculative Strategy for Heterogeneous Spark Cluster

Pengfei Zhang^{1,2,*}, Zonghuai Guo^{1,2}

¹Chongqing University of Posts and Telecommunications, 400065, Nan'an District, Chongqing, P.R.China

²Chongqing Mobile Internet Application Engineering Technology Research Center, 400065, Nan'an District, Chongqing, P.R.China

Abstract. Apache Spark is an open-source in-memory cluster-computing framework. Spark decomposes an application into numerous tasks and assigns them to computing nodes for higher efficiency. However, in heterogeneous environments, some tasks become stragglers because of poor performance of some computing nodes, data skew, etc. These stragglers can affect cluster performance seriously since a job completes just when the last undertaking completions. To mitigate stragglers, Spark uses speculative execution which recognizes slow tasks and picks the node to run speculative task, but the low accuracy in identification and simple way of backing up will further extend the execution time. Then we develop an improved speculative strategy, DBMTPE (Data-Based Multiple Phases Time Estimation), which selects stragglers by estimating their remaining time and chooses a proper way to run speculative task according to the cause. Experiment results show that DBMTPE can run applications up to 10.5% faster over Spark-Native and save computing resource at the same time.

1 Introduction

Apache Spark started as a research project at UC Berkeley's AMPLab in 2009, with a goal to design a unified engine that supports wider class of applications than MapReduce [1] for distributed data processing. Spark provides a similar programming model to MapReduce but extends it with a new distributed memory abstract model called "Resilient Distributed Datasets" (RDDs) [2]. It is the RDDs that allow Spark to cover almost all the typical big data computing mode, including iterative computing, batch computing, flow computing, etc. Using memory computing to ensure the low latency, Spark can run programs up to 100x faster than Hadoop. Due to the above advantages, Spark has been becoming more and more popular for big data processing.

Spark decomposes an application into numerous tasks and assigns them to computing nodes for higher efficiency. Stragglers are the tasks that run much slower than other tasks and since an application completes just when the last undertaking completions. [3] Little number of stragglers can further extend the execution time of an application. Stragglers are normally caused by heterogeneous resource capacity of worker nodes, network traffic, data skew and so on. Therefore, how to mitigate the impact of stragglers is an urgent problem to be solved.

To enhance the performance in heterogeneous environments, we identify some pitfalls in the native speculative mechanism and then present an improved speculative execution strategy DBMTPE for spark in this paper. DBMTPE estimates the remaining time of each task to identify whether it is slow, then finds out the cause of a straggler to determine which way will be adopted to handle it.

2 Related work

Although Spark covers the shortage of MapReduce for some certain class of applications like iterative jobs and interactive analysis, both are faced with the challenge of stragglers. The improved strategies among other distributed systems like MapReduce are instructive in the study of mitigating the impact of stragglers in Spark.

The native speculation of Hadoop uses the progress of a task and simply compares the progress of each task with the average of all tasks to identify stragglers, then starts the backups when there is no new map or reduce task to assign, finally mark a task as completed as long as one of the attempts completes. Longest Approximate Time to End (LATE) [4] aims to address the problems caused by the Hadoop scheduler in heterogeneous environments, it calculates the progress rate of tasks and estimates their approximate

* Corresponding author: 474644669@qq.com

remaining time, the one with longest remaining time has the highest priority to back up. The speculative execution strategy of Mantri [5] tries to avoid worthless backups result from data skew by estimating a task's remaining time based on the process bandwidth, meanwhile focuses more on computing resource saving. MCP [6] uses both progress rate and the process bandwidth to select slow tasks and determine which task to backup based on a cost-benefit model. Zheng [7] proposed a time prediction method named RISKI by collecting history information of a task and a novel speculative execution mechanism called SkewSeize to identify whether the straggler is caused by data skew and makes a choice between moving straggler and non-straggler.

The existing research mainly focuses on recognizing straggler by estimating the remaining time of tasks and choosing a proper way to back up according to the cause.

3 Background

3.1 Causes of Stragglers

Stragglers are the tasks that take an unusually long time to finish. In heterogeneous environments, we can classify the causes of stragglers into internal and external reasons. Internal reasons can be solved by service provider, but the external cannot. The internal reasons include heterogeneity among hardware and the contention for the computing resources on the same node within an application. It's obvious that the variety of data processing speed between nodes may leads to heterogeneous performance. By changing system setting, we can reduce the impact of resource competition within an application. The external reasons contain resource competition due to co-hosted, data skew, network congestion and so on. Most situations can be solved effectively by speculative execution, but data skew which means some tasks have huge amount data to process compared to other tasks, cannot be.

3.2 Spark Speculative Mechanisms

Spark Speculative Mechanisms is a health-check procedure that checks for tasks to be speculated whose running time is greater in a stage than the threshold - median of all successfully completed tasks in a task set multiply by the parameter "spark.speculation.multiplier". Such slow tasks will be re-submitted to another worker. It will not stop the slow tasks but run a new copy in parallel and take the

fast results. The mechanisms starts with spark.speculation enabled. It executes periodically every "spark.speculation.interval" after a certain amount of task is done.

3.3 Pitfalls in Spark Speculation

Spark speculative mechanisms mitigates the impact of stragglers in heterogeneous environments to a certain degree, but there are still two pitfalls as follows:

i) Pitfalls in selecting slow tasks - simply compare the running time with the threshold rather than estimate the remaining time of the tasks, some tasks may have been entered their final stage and will complete soon, this comparison is unfair to these tasks. Meanwhile, this way does not suit for heterogeneous environments, some tasks seems to take a long time to execute but they will finish earlier because of the better performance of the node.

ii) Pitfalls in backing up - directly add the slow tasks to the task queue and re-submit them without taking the causes into account, this strategy may lead to unnecessary speculative execution since the copies of slow tasks caused by data skew cannot finish earlier.

4 Our design

By identifying slow tasks more accurate then automatically choosing a proper way to run backup task according to the cause, we come up with a method named DBMTPE (Data-Based Multiple Phases Time Estimation) to alleviate the influence of stragglers in heterogeneous cluster. DBMTPE can be divided into two steps: i) Predict the remaining time of running tasks to select slow tasks; ii) Evaluate the difficulties of backing them up and re-submit them to other workers. Next, we will explain these two steps in detail and present the pseudo code of DBMPTE in Table1.

4.1 Predict the Remaining Time

Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for an application. A stage consists of many tasks. Tasks are sent to the executors to run. Here are the four running steps of each task: i) De-serialize the task object; ii) De-serialize RDD(s) needed and dependency or function according to the type of the task; iii) Use metadata of RDD(s) to get the actual partition data then calculate results; iv) Serialize the results. Then we define that Step i) combine with ii) to form the de-serializing phase (P1), step iii) is the computing phase (P2) and step iv) is the serializing phase (P3) [8]. The

execution time of a task mainly depends on the input data size and the computing performance of the machine that the task runs on. We will not start the prediction until P2 because the de-serializing time is short and the input data size can be got after P1.

Owing to the differences in the performance between nodes in a heterogeneous cluster, it's inaccurate to estimate remaining time of a task by using historical records of all successful tasks within the stage. We refer to historical records of those finished tasks running on a same host to predict the remaining time. The record contains the execution time of each phase, the input and output data size of a task.

In DBMPTE, the remaining runtime of a task rem_time' is the sum of remaining time of each phase. What we can predict is the total time of a phase $total_time'_{p_i}$, the remaining time of current phase $rem_time'_{cp}$ is the difference between $total_time'_{cp}$ and the time run_time_{cp} ; In the following phases, $rem_time'_{fp}$ equals to $total_time'_{fp}$ because the task has not entered those phases yet. As shown in formula (1):

$$\begin{aligned} rem_time' &= rem_time'_{cp} + \sum rem_time'_{fp} \\ &= total_time'_{cp} - run_time_{cp} + \sum total_time'_{fp} \end{aligned} \quad (1)$$

We use regression analysis when estimating the runtime of P2. $input_{run}$ indicates the input data volume of the running task to be estimated, $input_{his_i}$ represents the data size of a successful task i on the same host within the same stage, and $total_time_{p2_i}$ is the runtime of P2 in task i . Let δ and K be constants while both are greater than zero, and let N be the number of task i whose $input_{his_i}$ is δ -close to $input_{run}$ (formally, $|input_{his_i} - input_{run}| < \delta$), if N is greater than K which means there are enough tasks similar to the current one, then $total_time'_{p2}$ is the average runtime of P2 in those tasks; otherwise we use linear regression to characterize the relationship between $input_{his}$ and $total_time'_{p2}$, this relationship will make a prediction through $input_{run}$ and ω is a linear coefficient. As shown in formula (2):

$$total_time'_{p2} = \begin{cases} \sum_{i=1}^N total_time_{p2_i} / N & (N \geq K) \\ \lambda * input_{run} + \omega & (N < K) \end{cases} \quad (2)$$

We use the output data volume $output_{run}$ and the average serializing speed V_{ser} to calculate the runtime of P3, as shown in formula (3). If the task to be estimated has not entered P3, use the average ratio ρ

of output data size to input data size of all finished tasks to estimate $output_{run}$ by $input_{run} * \rho$. Otherwise we can get $output_{run}$ directly. V_{ser} comes from the average ratio of output data size to the runtime of P3 of a task in those finished tasks.

$$total_time'_{p3} = output_{run} / V_{ser} \quad (3)$$

4.2 Evaluate the failure Rate of backups

The greatest challenge for the speculative execution is that it does not guarantee the backup task complete earlier than the original task. If the original straggler ends earlier, we call this execution failed. In the worst case, all the speculative execution failed thus computing resources were wasted with no reduction in the execution time of an application. Two main factors impeding the execution are taken into consideration in this paper:

i) Input data skew; input data skew may be inevitable in big data processing. Some tasks have more data to process compared to the other tasks within a same stage they are more likely to be stragglers in the backup nodes and doomed to be slow. The speculative execution normally has nothing to do with this type of task.

ii) Execution progress of a task; it is essential to consider the current execution progress completed for a task for effective replica generation. A speculative duplicate should be created only if it is likely to end prior to the straggler. When a task has entered the late phase, it still has a great chance to finish earlier than the new duplicate.

By considering these two factors, we use formula (4) evaluate FR the failure rate of generating a replica.

$$FR = \alpha * S + (1 - \alpha) * P \quad (4)$$

Let $input_{median}$ be the median input of those finished tasks when speculative mechanisms start to work. S is defined as $input_{run} / input_{median}$; P indicates the ratio of the execution time of a straggler to the estimated total time.

Table 1. Pseudo code of DBMPTE.

Algorithm 1 DBMPTE
Input: taskInfos: Records of all tasks in this stage
MD: Median duration of all successful tasks in the stage
MP1: Mapping between host and the taskInfos of finished tasks on the host
MP2: Mapping between host and the serialize speed of the output
$input_{median}$: The median input of those finished tasks when speculative mechanisms start
ρ : The average ratio of output data size to input data size of all finished tasks

K, δ, ψ, α : model coefficients

Output: speculatableTasks sorted in ascending order of FR

```

1: for task in taskInfos do
2:   if task has not finished then
3:      $RT \leftarrow 0$ 
4:     if task has entered P2 then
5:        $N \leftarrow MP1.get(host).filter(|input_{his_i} - input_{run}| < \delta$ 
6:     if  $N \geq K$  then
7:        $total\_time_{p2}' = \sum_{i=1}^N total\_time_{p2_i} / N$ 
8:     else
9:        $total\_time_{p2}' = \lambda * input_{run} + \omega$ 
10:    end if
11:     $RT \leftarrow$ 
 $total\_time_{p2}' - runtime_{p2} + input_{run} * \rho / V_{ser}$ 
12:    else if task has entered P3 then
13:       $RT \leftarrow output_{run} / V_{ser} - runtime_{p3}$ 
14:    end if
15:    if  $RT \psi * MD$  then
16:      speculatableTasks += task
17:    end if
18:  end if
19: end for
20: for task in speculatableTasks do
21:    $total\_time' \leftarrow RT + task.runtime$ 
22:    $FR \leftarrow \frac{\alpha * input_{run} / input_{median} + (1 - \alpha) * RT}{total\_time'}$ 
23: end for
24: sort tasks in speculatableTasks in ascending order of FR and find the median  $FR_{median}$ 
25: filter out those tasks whose  $FR > \psi * FR_{median}$  launch backups for the rest of tasks

```

5 Evaluation

Our heterogeneous cluster consists of 3 virtual machines: 8 logical cores (2.4GHz Intel(R) Xeon(R) E5620), 12G RAM and 500G disks; 10 logical cores (2.4GHz Intel(R) Xeon(R) E5620), 12G RAM and 500G disks; 4 logical cores (3.2GHz Intel(R) Core(TM) i5-6500), 8G RAM and 1T disks.

In our experiments, we calculate the average temperature of each year from 2000 to 2016 based on the nearly 20GB NCDC dataset. We use SKEW to measure the skew degree of the dataset, which is the ratio of the sum data size of 50% of the years with largest input to total data size. This application includes two stages: i) Stage0 load data from HDFS, then run filter, map and groupByKey; ii) Stage1 get the output through shuffle from the previous stage then run reduce. We keep the default configuration for

HDFS, so the block size of HDFS is 128MB. There is no sign of straggler problem in Stage0, we focus on the straggler problem in Stage1.

5.1 Execution time of a stage

To verify the effectiveness of the DBMPTE in a heterogeneous cluster, we compare the average execution time of Stage1 in Spark with no speculation, native speculation and DBMPTE. We perform the experiment 5 times for each Strategy and task the average under different SKEW. From Fig.1, we can see our algorithm can improve the stage execution time up to 10.5% over that of Spark-Native and by 19.6% over that of Spark-None. DBMPTE predicts the remaining time and evaluates the failure rate of backup based on the monitoring information of tasks, this involves some involves some network transmission and computation. So DBMPTE is inferior to Spark-Native when the dataset has a uniform distribution, and outperforms Spark-Native when the benefits from these overhead exceeds the costs as more stragglers caused by data skew.

Fig. 1. The execution time of Stage1.

5.2 Accuracy of straggler identification

If the speculative duplicate ends ahead of the straggler task, we call that Spark make a correct identification. When SKEW is set to 89.3%, conduct the experiment 10 times for the native strategy and DBMPTE. Fig.2 shows our algorithm has higher identification accuracy than the native in all cases. We attribute the improvement to i) the reference to the historical records on the same host which leads to a better prediction ii) the evaluation of data skew and task progress which lets us avoid conducting unnecessary backups.

Fig. 2. The identification accuracy of Stage1.

5.3 Consumption of computing resource

To further evaluate the DBMPTE, we give an analysis of the CPU and memory consumption during the execution by Ganglia. We have the same setting as the previous experiment. As shown in Fig.3 and Fig.4, the usage of CPU and memory hold stably from the beginning to 75%, and then start to fall. DBMPTE declined faster than the native speculative mechanisms because DBMPTE can save computing resource by avoiding conducting unnecessary backups.

Fig. 3. CPU usage of the cluster.

Fig. 4. Memory usage of the cluster.

In summary, based on time prediction and backup failure rate evaluation, DBMPTE can effectively shorten the execution time of an application with data skew in heterogeneous cluster, and save computing resource at the same time.

5 Conclusion

Spark uses speculative execution to mitigate the impact of straggler problem on an application's execution. In this paper, we have analyzed the Spark-original speculative mechanisms and its pitfalls, and propose an improved speculative strategy DBMPTE based on time prediction and backup failure rate evaluation to further shorten the execution time. It can estimate the remaining time of unfinished tasks and identify the potential main causes of the slow tasks, then evaluate the difficulties of backing them up based on the causes, finally back up those stragglers with the minimum difficulty. Experiment results show that DBMPTE can run applications faster and save computing resources at the same time. DBMPTE is more suitable for the input data skew applications in heterogeneous cluster.

In the future, we will focus more on assigning the backup tasks, avoiding scheduling replicas to another slow node.

References

1. J. Dean, S. Ghemawat. Communications of the ACM, **51**(1), pp.107-113.(2008)
2. M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, and I. Stoica. *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* pp.2-2(2012)
3. U. Kumar, J. Kumar. *International Journal of Grid and Distributed Computing*.7(4), pp.139-148 (2014)
4. M. Zaharia, A. Konwinski, A.D. Joseph, R.H. Katz, and I. Stoica. *Osd*. **8**,4, pp.7-7(2008)
5. G. Ananthanarayanan, S. Kandula, A.G. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. *Osd*. **10**,1, p.24(2010)
6. Q. Chen, C. Liu, and Z. Xiao. *IEEE Transactions on Computers*, **63**,4,pp.954-967(2014)
7. S. Zheng. Master's thesis. National University of Defense Technology, Changsha, China (2014)
8. H. Yang, X. Liu, S. Chen, Z. Lei, H. Du, and C. Zhu. *Audio, Language and Image Processing*, pp.28-33.(2016)