IEICE TRANS. INF. & SYST., VOL.E99–D, NO.6 JUNE 2016

LETTER

DynamicAdjust: Dynamic Resource Adjustment for Mitigating Skew in MapReduce

Zhihong LIU[†,††a)], **Aimal KHAN**[††], ***Nonmembers***, **Peixin CHEN**[†], ***Student Member***, **Yaping LIU**[†], ***and*** **Zhenghu GONG**[†], ***Nonmembers***

SUMMARY MapReduce still suffers from a problem known as *skew*, where load is unevenly distributed among tasks. Existing solutions follow a similar pattern that estimates the load of each task and then rebalances the load among tasks. However, these solutions often incur heavy overhead due to the load estimation and rebalancing. In this paper, we present DynamicAdjust, a dynamic resource adjustment technique for mitigating skew in MapReduce. Instead of rebalancing the load among tasks, DynamicAdjust adjusts resources dynamically for the tasks that need more computation, thereby accelerating these tasks. Through experiments using real MapReduce workloads on a 21-node Hadoop cluster, we show that DynamicAdjust can effectively mitigate the skew and speed up the job completion time by up to 37.27% compared to the native Hadoop YARN.

key words: *MapReduce, resource-aware scheduling, skew mitigation*

1. Introduction

MapReduce has proven itself as a popular and powerful tool for large-scale data processing. By breaking down a data-intensive job into a number of small tasks and executing them in parallel across multiple machines, MapReduce can significantly decrease the job running time. However, MapReduce still suffers from a problem known as *skew*, where load is unevenly distributed among tasks. It can make a number of tasks take dramatically longer to complete than others, thereby slowing down the entire job.

Many solutions have been proposed recently to mitigate skew in MapReduce. Most of them [1]–[5] follow a similar pattern that profiles the distribution of the intermediate key-value pairs and then reassigns the key-value pairs among reduce tasks to achieve a better balance. However, in order to gather the information of the key-value pairs, these solutions either have to wait for the completion of all map tasks [1]–[3] or execute an additional sampling phase before the actual job starts [4], [5]. Some other solutions [6], [7] speculatively execute replica tasks for slow running tasks, hoping the replica tasks can finish faster than the original tasks. However, the replica tasks have the same load as the original tasks. Executing them elsewhere may not improve the job completion time when skew exists.

More recently, a run-time skew mitigation technique Skewtune [8] has been proposed. Skewtune repartitions the remaining load of a slow running task and launches an additional MapReduce job to process the remaining load. However, it will impose a heavy overhead (as reported in [8], 30s is imposed for mitigating the skew). The solutions in [9], [10] dynamically allocate resources for reduce tasks according to their estimated load. However, these solutions focus on the skew in the reduce stage, but cannot mitigate the skew in the map stage, which limits their applicability.

In this paper, we present DynamicAdjust, a dynamic resource adjustment technique for mitigating skew in MapReduce. Different from existing work, DynamicAdjust monitors the progress of each task at run-time and detects skewed tasks based on a phase-aware task remaining time prediction technique. After that, DynamicAdjust adjusts resources for the skewed tasks using idle resources. DynamicAdjust makes no assumption regarding the cause of the skew. As a result, it can mitigate any kind of skew, both in map and reduce stages. Through experiments using real workloads on a 21-node Hadoop cluster, we show that DynamicAdjust can shorten the job completion time by up to 37.27% compared to the native Hadoop YARN 2.4.0.

2. Background and Motivation

2.1 Resource Allocation Scheme in Hadoop YARN

Hadoop YARN [11] is a widely-used MapReduce implementation. In YARN, the *container* is used to manage the resources. It provides specific resource accountings (*e.g.* <2 GB RAM, 1 CPU>) and enforces the resource limits on the task. Nevertheless, YARN fails to consider that the run-time resource requirement varies from task to task, and does not support dynamic resource adjustment for an allocated container. As a result, the tasks that require more resources (*e.g.* skewed tasks) may run slower because their resources are limited by the allocated containers, thereby prolonging the job completion time.

2.2 Skew in MapReduce

There are mainly two types of skew in MapReduce: 1) *Computational Skew*, which occurs due to the fact that some expensive records take longer to process, and it happens both in map and reduce stages; 2) *Partitioning Skew*, which occurs due to the hash function used for distributing the in-

Manuscript received December 15, 2015.
Manuscript revised February 1, 2016.
Manuscript publicized March 7, 2016.

[†]The authors are with the College of Computer, National University of Defense Technology, Changsha, Hunan, China.
[††]The authors are with the David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada.
a) E-mail: zeroun.liu@uwaterloo.ca

DOI: 10.1587/transinf.2015EDL8255

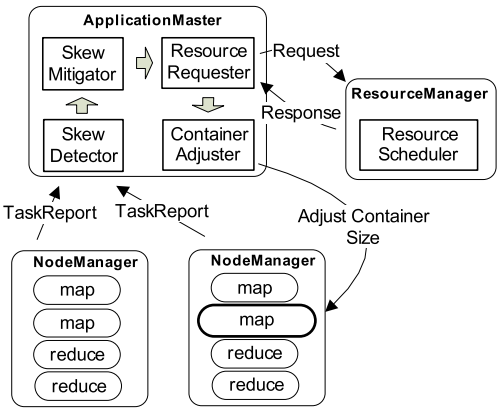


Fig. 1 Architecture of DynamicAdjust

termediate data among reduce tasks, and this type of skew arises especially in the reduce stage.

3. Proposed Method

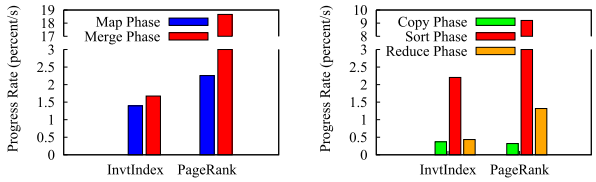
3.1 System Architecture

The architecture of DynamicAdjust is shown in Fig. 1. There are five main components: the Skew Detector, the Skew Mitigator, the Resource Requester, the Container Adjuster and the Resource Scheduler. The workflow of DynamicAdjust consists of following steps:
(1) The Skew Detector collects the task reports from the running tasks, detects the skew according to the task progress and notifies the Skew Mitigator when skew is detected.
(2) The Skew Mitigator determines how much resources are needed for the skewed tasks and notifies the Resource Requester to ask resources from the ResourceManager. Subsequently, the Resource Requester sends the resource adjustment requests to the Resource Scheduler.
(3) After the Resource Scheduler receives the requests, it allocates resources to the corresponding ApplicationMaster when resources in the cluster become available.
(4) Once the ApplicationMaster receives the response that the requested resources have been allocated, it notifies the Container Adjuster to execute the resource adjustment in the NodeManager.

3.2 Skew Detection

We consider that if the estimated duration of the task (denoted by *estimatedDuration*) is greater than the historical average duration (denoted by *averageDuration*) by θ_{slow}, then this task is a candidate of skewed tasks. Also, it is not beneficial to mitigate the skewed task near completion, even though its estimated duration is long. Therefore, the task that has the greatest remaining time among the candidates is selected as the task need to be mitigated each time.

The solution in LATE [6], assumes the progress rate is stable in each phase. Using this assumption, LATE estimates the task remaining time by $\frac{1-progressScore}{progressRate}$ and then



(a) Map Stage (b) Reduce Stage

Fig. 2 Progress rate of each phase

speculatively launches replica tasks based on the task remaining time estimation. However, the progress rate is not necessarily stable during the task execution. Figure 2 compares the progress rate of each phase in PageRank and InvertedIndex jobs. It is clear that different phases have different progress rates and the fluctuation is dramatic. LATE uses an unchanged progress rate to predict the task remaining time, which will increase the prediction error.

Hence, we propose a phase-aware task remaining time prediction technique. It predicts the task remaining time by summing up the remaining time in current and following phases. For the remaining time in the current phase, we use the progress rate measured in the current phase. For the remaining time in following phases, we use the average progress rates calculated from finished tasks. Since we will not start the skew mitigation after 10% of tasks have completed, there will be sufficient historical statistics for calculating the per phase average progress rate.

With the task remaining time predicted, we can obtain the *estimatedDuration* by summing up the task elapsed time and remaining time. Note that DynamicAdjust leverages the task status report mechanism inherent in Hadoop, determining the *estimatedDuration* and *averageDuration* will not incur noticeable overhead over Hadoop.

3.3 Skew Mitigation

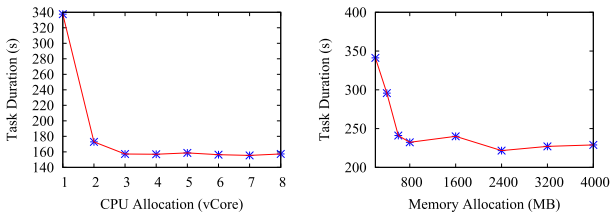
Since DynamicAdjust will adjust the resources for the skewed tasks, we need to know the relationship between the resource allocation and the task running time. To this end, we compare the task running times by varying the CPU allocation from 1 *vCore* to 8 *vCores* with the memory allocation fixed to $1GB$. Furthermore, we also compare the task running times by varying the memory allocation (JVM heap size)† from 200 MB to 4000 MB with the CPU allocation fixed to $1vCore$. Figure 3 shows the results for a random task from InvertedIndex. We can see that with the resource allocation increased, the task running time decreases until the resources become sufficient for the task. Similar results can also be observed in other workloads.

Therefore, in terms of CPU adjustment, we scale up the allocation to $\frac{estimatedDuration}{averageDuration} Alloc_{old}^{cpu}$, where $Alloc_{old}^{cpu}$ is the previous CPU allocation; in terms of memory adjust-

†There are two settings in YARN for memory allocation: logical memory limit and JVM heap size. The former one is used for the ResouceManager to manage the cluster resources, and the latter one is the actual memory limit that the container can use.

Table 1 Skew Detection for InvertedIndex (235 map tasks and 60 reduce tasks in total)

Strategy	Map Stage				Reduce Stage			
	True Positive	True Negative	False Positive	False Negative	True Positive	True Negative	False Positive	False Negative
Hadoop-LATE	5	205	25	0	0	55	0	5
DynamicAdjust	5	225	5	0	4	55	0	1



(a) CPU (b) Memory

Fig. 3 Relationship between task duration and resource allocation in InvertedIndex

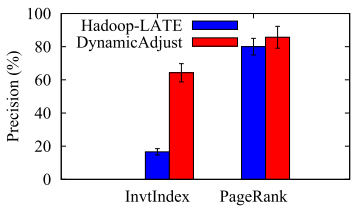


Fig. 4 Comparison of the skew detection between Hadoop-LATE and DynamicAdjust

ment, we do not change the memory allocation at run-time since YARN is using JVM based container, and it is not able to change the JVM heap size dynamically. We set the JVM heap size to 80% of the logical memory limit while launching a task. The intention is to try to maximum the usable memory, but still subject to the resource bounding.

Overall, this strategy is simple yet works well in practice. It roughly estimates the resource requirement for accelerating the skewed tasks to catch up with the normal tasks, and privileges these tasks by guaranteeing their resource allocation. Even though we do not adjust the memory at runtime, we found that $800MB^{\dagger}$ of the JVM heap size is actually enough for a task in our experiments.

4. Evaluation

In this section, we evaluate the performance of DynamicAdjust using a real Hadoop cluster with 21 virtual machines (VMs) in the SAVI Testbed [12]. Each VM has four 2 GHz cores, 8 GB RAM and 80 GB disk. We deploy the ResourceManager and NameNode on same VM, and the remaining 20 VMs are used as workers. Each worker is configured with 8 vCores and 7GB RAM (leaving 1GB for background processes). The HDFS block size is set to 128 MB, and the replication level is set to 3. We activate the CGroups configuration and also enable the map output compression. We evaluate DynamicAdjust using 1) InvertedIndex, with a 30 GB Wikipedia dataset; 2) PageRank, with a 13 GB ClueWeb09 dataset.

4.1 Accuracy of Skew Detection

In this set of experiments, we activate the skew detection, but do not perform the skew mitigation. Thus, we let the labeled tasks run as usual and verify whether they are truly skewed in the end. Similar to [7], we consider the tasks whose durations are 1.5 times longer than the average as skewed tasks. Figure 4 compares the precision†† of the skew detection between Hadoop-LATE (the Hadoop implementation of LATE) and DynamicAdjust. It is clear that DynamicAdjust outperforms Hadoop-LATE significantly. In particular, DynamicAdjust can improve the precision by 47.61% and 5.7% for InvertedIndex and PageRank, respectively.

In order to understand the rationale behind the gain, we show the detailed skew detection results for InvertedIndex in Table 1. We can see that in the map stage, both strategies can detect all the skewed tasks. However, Hadoop-LATE misjudges 25 normal tasks as skewed tasks. In comparison, the number of False Positive for DynamicAdjust is 5. In the reduce stage, Hadoop-LATE cannot detect any of the skewed tasks, however, DynamicAdjust identifies 4 out of 5 skewed tasks. Besides, we can also observe that there is little improvement for PageRank. That is because in PageRank the skewed tasks are extremely slow, and the normal tasks hover over at the average. Thus, both Hadoop-LATE and DynamicAdjust can detect the skewed tasks accurately.

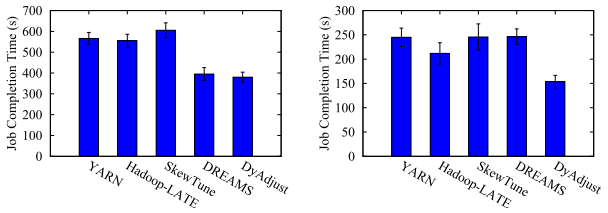
4.2 Performance of Skew Mitigation

In this section, we want to validate how well DynamicAdjust can mitigate skew. We compare DynamicAdjust against 1) native Hadoop YARN 2.4.0; 2) the speculation-based straggler mitigation approach (Hadoop-LATE); 3) the repartition-based skew mitigation approach (SkewTune) and 4) the resource-aware skew mitigation approach (DREAMS [9]). Note that SkewTune is implemented on Hadoop version 1, which is slot-based and there is no resource isolation between slots. In order to fairly compare the approaches mentioned above, we have implemented isolation between slots in Hadoop version 1 (0.21.0) and installed SkewTune on top of it. We configure each worker with 6 map slots and 2 reduce slots for SkewTune.

Figure 5 shows the job completion time for each job while using different mitigation strategies. As we can see

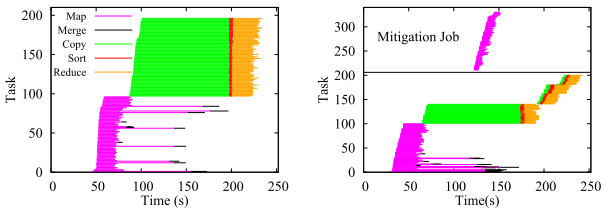
†The minimum logical memory limit is 1 GB by default, and 80% of that is 800 MB.

††The precision is calculated by $\frac{True\ Positive}{True\ Positive+False\ Positive}$.

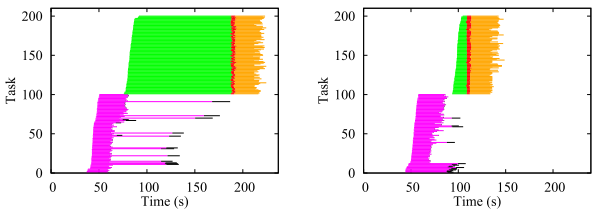


(a) InvertedIndex (b) PageRank

Fig. 5 Job Completion time comparison



(a) YARN (b) SkewTune



(c) DREAMS (d) DynamicAdjust

Fig. 6 Job Execution timeline comparison

from the figure, DynamicAdjust outperforms other skew mitigation approaches in all case. In particular, DynamicAdjust improves the job completion time by 32.86% and 37.27% in InvertedIndex and PageRank, respectively. There is no improvement observed for Hadoop-LATE and SkewTune. These two strategies sometimes perform worse than YARN in our experiments. DREAMS can improve the job completion time for InvertedIndex. However, for PageRank which has skew in the map stage, DREAMS cannot bring any benefit.

To explain why DynamicAdjust outperforms other skew mitigation strategies, we plot the execution timeline while running PageRank with different strategies[†] in Fig. 6. We can see in Fig. 6 (a) that several tasks take much longer than other tasks in the map stage. As shown in Fig. 6 (b), SkewTune repartitions one of skewed tasks and launches an additional job to process this skewed task using free cluster resources. However, SkewTune cannot mitigate another skewed task while there is a mitigation job in progress (see Sect. 3.2 in [8]). Therefore, it misses the best time to mitigate concurrent skewed tasks. DREAMS focuses on partitioning skew, but cannot handle the skew that arises in the map stage. Thus, we can see that there is no improvement in Fig. 6 (c). In Comparison, DynamicAdjust adjusts resources for skewed tasks both in map and reduce stages. As shown in Fig. 6 (d), the durations of the skewed map tasks are decreased dramatically, thereby improving the job completion time.

[†]Due to the space limit, we do not show the timeline for Hadoop-LATE.

5. Conclusion

In this paper, we presented DynamicAdjust, a framework that provides run-time skew mitigation. In contrast to existing solutions, we adjust resources dynamically for tasks that need more computation, thereby accelerating these tasks. In our performance evaluation using real workloads on a 21-node Hadoop cluster, we demonstrated that DynamicAdjust can improve the precision of skew detection by up to 47.64% compared to Hadoop-LATE. We also showed that DynamicAdjust can improve the job completion time by up to 37.27% in comparison to the native Hadoop YARN.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (No.61472438).

References

[1] J. Son, H. Choi, and Y.D. Chung, "Skew-tolerant key distribution for load balancing in MapReduce," IEICE Trans. Inf. & Syst., vol.E95-D, no.2, pp.677–680, 2012.
[2] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, and S. Wu, "Handling partitioning skew in MapReduce using LEEN," Peer-to-Peer Networking and Applications, vol.6, no.4, pp.409–424, 2013.
[3] N. Zacheilas and V. Kalogeraki, "Real-time scheduling of skewed MapReduce jobs in heterogeneous environments," ICAC, pp.189–200, USENIX, 2014.
[4] S.R. Ramakrishnan, G. Swart, and A. Urmanov, "Balancing reducer skew in MapReduce workloads using progressive sampling," Proc. Third ACM Symposium on Cloud Computing - SoCC '12, pp.1–14, 2012.
[5] W. Yan, Y. Xue, and B. Malin, "Scalable and robust key group size estimation for reducer load balancing in MapReduce," 2013 IEEE International Conference on Big Data, pp.156–162, 2013.
[6] M. Zaharia, A. Konwinski, A.D. Joseph, R.H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," OSDI, p.7, 2008.
[7] Q. Chen, C. Liu, and Z. Xiao, "Improving MapReduce performance using smart speculative execution strategy," IEEE Trans. Computers, vol.63, no.4, pp.954–967, 2014.
[8] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "SkewTune: mitigating skew in Management applications," SIGMOD, pp.25–36, ACM, 2012.
[9] Z. Liu, Q. Zhang, M.F. Zhani, R. Boutaba, Y. Liu, and Z. Gong, "Dreams: Dynamic resource allocation for mapreduce with data skew," IFIP/IEEE IM 2015, pp.18–26, Ottawa, Canada, May 2015.
[10] Z. Liu, Q. Zhang, R. Boutaba, Y. Liu, and B. Wang, "Optima: Online partitioning skew mitigation for MapReduce with resource adjustment," Springer JNSM, 2016.
[11] "Apache hadoop yarn," http://hadoop.apache.org/
[12] "Savi testbed," http://www.savinetwork.ca