ReB: Balancing Resource Allocation for Iterative Data-Parallel Jobs

Dongsheng Li, Zhiyao Hu, Deke Guo, Yiming Zhang
National University of Defense Technology
Changsha, China
dsli@nudt.edu.cn,huzhiyao14@nudt.edu.cn,guodeke@nudt.edu.cn,ymzhang@nudt.edu.cn

ABSTRACT

The under-allocation of computation resource would slow down the completion of data-parallel jobs like Hadoop and Spark applications. Hence, users are inclined to speed jobs by requiring more computation resource. In this paper, we reveal an unexpected fact that the over-allocation of computation resource would also lengthen the job completion time (JCT). We attribute the over-allocation problem to the underlying system overheads and utilize machine learning techniques to predict the optimal resource allocation for a given job. Via the prediction model, the resource manager prevents jobs from demanding excessive computation resource. Furthermore, we discuss the optimal multi-job resource allocation if the JCT of one job is known in advance as a function in terms of the resource allocation.

CCS CONCEPTS

• **Computer systems organization** Distributed architectures;

KEYWORDS

data-parallel iterative job, resource over-allocation, optimal resource allocation, performance prediction

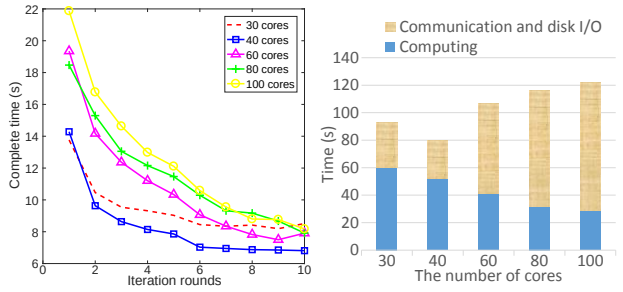
ACM Reference Format:
Dongsheng Li, Zhiyao Hu, Deke Guo, Yiming Zhang. 2018. ReB: Balancing Resource Allocation for Iterative Data-Parallel Jobs. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

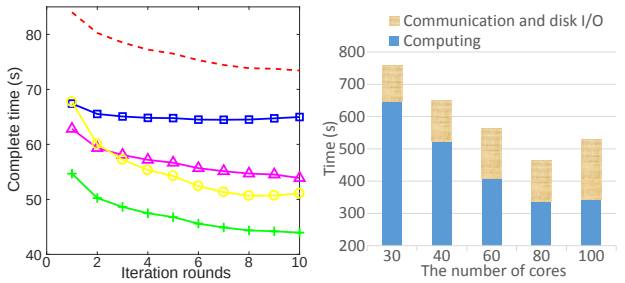
For data-parallel jobs like Spark and Hadoop applications, resource requirements are customized by the user and the JCT is associated with the amount of allocated computation resource. It is well-known that the under-allocation of computation resource slows down the completion time. Hence, many users are inclined to speed up jobs simply by requiring more computation resource. However, the over-allocation of computation resource brings nonnegligible system overheads for the network communication and disk I/O operations. As a result, the JCT may increase. Previous research [4] also shows the performance reduction when the scale of distributed system expands, although the calculation process could be speeded with increasing allocation of computation resource. Moreover, the over-allocation of partial jobs consumes considerable resource, which may cause that other jobs suffer the under-allocation, finally making the average JCT worse. To the best of our knowledge, there is no relevant research to solve the over-allocation problem. In this paper, we focus on characterizing and solving the *optimal resource allocation* (ORA) problem for single-job and multi-job cases.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn



(a) The duration per iteration of PageRank. 40 cores are the optimal.
(b) The component of computation and other time for PageRank in Fig. 1(a).



(c) The duration per iteration of Regression. 80 cores are the optimal.
(d) The component of computation and other time for Regression in Fig. 1(c).

Figure 1: When the number of cores increases from 30 to 100, SparkPageRank and SparkRegression.

1.1 Rethinking the single-job resource allocation

For a single data-parallel job, the amount of allocated computation resources imposes both positive and negative impacts on the JCT. The increasing amount of computation resource like CPU cores could reduce the JCT by executing more tasks of the job concurrently. However, each task suffers the network and disk I/O overheads resulting from the data shuffle and the storage of intermediate results. Such overheads significantly grow up along with the increase of the allocated computation resource. This would finally slow down the job completion. Thus, when the amount of allocated computation resource increases, there exists a balance between increasing overheads and decreasing computation time, i.e. the *optimal* allocation of computation resource. However, the optimal allocation of computation resource varies with the workload, job configurations, and other settings. Thus, it remains open to find the optimal allocation of one job. We aim at identifying the optimal allocation which achieves the shortest JCT.

We measure the JCT of PageRank jobs when the number of cores increases from 30 to 100. For each tested job, the input data is also the same and pre-stored in HDFS. The number of tasks always keeps the same as that of cores. Each job involves 10-round iterations. Fig. 1(a) shows the duration per iteration. We see 100 cores result in the longest duration in the 1st round iteration. This indicates that the over-allocation of CPU cores incurs the performance reduction. When the allocated cores increases from 60 to 100 cores, the more resource is allocated, the JCT is longer. On the other hand, the job

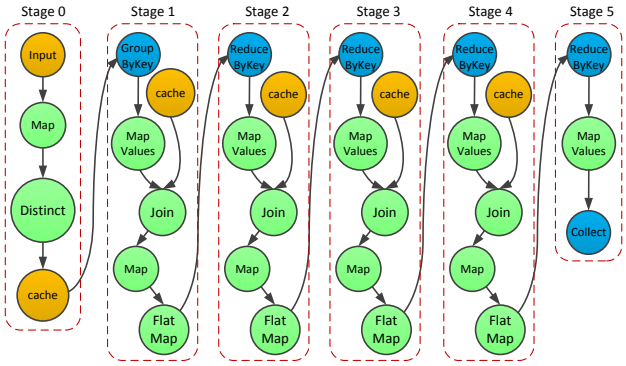


Figure 2: An illustrative workflow of PageRank which consists of 6 stages. Each node denotes one operation. Green and blue nodes involve computation and communication operations, respectively.

allocated 30 cores suffers the under-allocation. 40 cores are the optimal not only in the 1st round but also in the later iteration rounds. Fig. 1(c) shows the similar problem for Regression but the optimal allocation is 80 cores. This indicates that the optimal allocation varies with application types. Moreover, Fig. 1(b) and Fig. 1(d) show that the component of JCTs. We can see that although computation time decreases with increasing cores, the total JCT doesn't always decrease owing to increasing communication and disk I/O time.

1.2 Feature selection

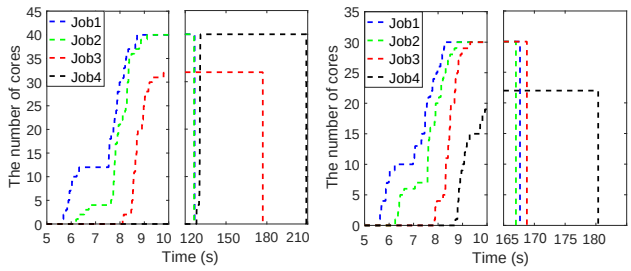
Fig. 1 shows that the JCT is a function in terms of the resource allocation. We infer that the duration of data-parallel jobs is dominated by not only computation time and disk I/O overheads, but also the highly dynamic network [1]. This makes modeling the JCT function for prediction very challenging. Previous prediction methods [2, 3, 5] adopt high-level job settings like input sizes and the number of virtual machines. However, these methods don't consider the cause for underlying overheads like communication and disk I/O, which are associated with the types of application. In fact, such overheads are originated by **operation**s, which serve as underlying programming interfaces such as Map and Reduce.

We analyze the DAG of jobs and extract the operations as features to train a prediction model. For one data-parallel job, the type and number of operations determines the time for computation, communication and disk-I/O. For example, communication-intensive applications are inclined to frequently use network operations like *Reduce*, *GroupByReduce* and *SortByReduce*. Fig. 2 shows the DAG of PageRank. Each node representes one operation. Main operations includes *Map*, *FlatMap*, *Join*, *ReduceByKey* and *GroupByKey*. Blue operations indicate communication overheads such as the data shuffle. Green operations indicate computation time. We are inspired to exploit these operations for training the prediction model which is aware of the underlying overhead.

1.3 Rethinking the multi-job resource allocation

Although the prediction model searches the optimal resource allocation for each job, the resource capacity is insufficient to optimize multi-job resource allocation. Thus, the allocation of only individual jobs are optimized and other jobs have to starve. Intuitively, the allocation of the shortest job should be the optimal with higher priorities than long jobs. We define the naïve method as the **shortest-job-optimal** method. However, our micro-benchmark demonstrates that the naïve shortest-job-optimal method is inefficient.

We conduct multi-job allocation experiments in a small-scale cluster which has 120 cores. Fig. 3 shows the number of cores which are allocated to four PageRank jobs. The four jobs are totally identical with the PageRank job in Fig. 1(a) and submitted at the same



(a) The shortest-job-optimal method allows the optimal allocation of 40 cores. Job1, Job2 and Job3 are allocated 40, 40 and 32 cores, respectively. However, Job4 is delayed.

(b) One heuristic method replaces the optimal allocation with the near-optimal under-allocation. Job1, Job2 and Job3 are allocated 30 cores. Job4 are allocated 22 cores.

Figure 3: A micro benchmark shows that the shortest-job-optimal method is suboptimal compared with one heuristic method. The same 4 jobs (Job1, Job2, Job3 and Job4) are the job in Fig. 1(a) and submitted at the same time. The optimal allocation is 40 cores.

time. Seeing from Fig. 1(a), we know that the optimal allocation is 40 cores per job. We compare the shortest-job-optimal method with another heuristic method.

Fig. 3(a) shows details of the shortest-job-optimal method. We can see that not all jobs are optimally-allocated because the total cores aren't sufficient. Job1, Job2, and Job3 are all allocated at the beginning. However, Job4 is delayed owing to insufficient resource and doesn't start until Job1 completes and releases the resource. Fig. 3(b) shows that another heuristic method which gives up the optimal allocation. The heuristic method under-allocates 30 cores to each job and performs better than the shortest-job-optimal method. The above evaluation reveals the shortest-job-optimal method is suboptimal under insufficient resource. We attribute it to low *allocation efficiency*, namely, although individual jobs consumed considerable computation resource, their JCTs wouldn't be decreased a lot.

2 FUTURE WORK

First, we compare different machine learning techniques such as the decision tree, random forest and neural network methods to predict the JCT given the resource allocation of one job. Two challenges are to collect data and to enhance the accuracy of the cross-application prediction. Second, we define the allocation efficiency of one job so as to control the amount of resource allocation. Then, we design one multi-job resource allocation algorithm under the support of the prediction model. The multi-job resource allocation algorithm aims at maximizing the total allocation efficiency. We also consider design an online resource allocation method with a small time complexity. We plan to implement the resource allocation method in the Apache Spark Platform and compare it with the state-of-the-art YARN scheduler.

REFERENCES

[1] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: simplified data processing on large clusters. In *Conference on Symposium on Opearting Systems Design & Implementation*. 10–10.

[2] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon:QoS-aware scheduling for heterogeneous datacenters. *Acm Sigarch Computer Architecture News* 48, 4 (2013), 77–88.

[3] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: resource-efficient and QoS-aware cluster management. In *International Conference on Architectural Support for Programming Languages and Operating Systems*. 127–144.

[4] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and M. West. 1987. Scale and performance in a distributed file system. In *Eleventh ACM Symposium on Operating Systems Principles*. 1–2.

[5] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. 2011. CloudScale: elastic resource scaling for multi-tenant cloud systems. In *ACM Symposium on Cloud Computing*. 5.