BAS: Branch-Aware Scheduling for Iterative Data-Parallel Jobs

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

ABSTRACT

A data-parallel job consists of multiple dependent stages and coflows. DAG scheduling like the critical path method aims at scheduling stages according to the duration of stages. Coflow scheduling like Varys and Aalo only focuses on the coflow completion and is agnostic to the stage semantics. The divergence between the DAG scheduling and coflow scheduling brings challenges on decreasing the job completion time (JCT). To address this problem, we propose the branch as the abstraction including both the stage and coflows. For decreasing the JCTs, we utilize the job-level semantics of branches to design the branch scheduling method.

CCS CONCEPTS

• **Computer systems organization** Distributed architectures;

KEYWORDS

DAG scheduling, coflow scheduling, branch semantics

ACM Reference Format:
Zhiyao Hu, Yiming Zhang, Dongsheng Li, Deke Guo, Ziyang Li. 2018. BAS: Branch-Aware Scheduling 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 distributed computing such as Hadoop and Spark, a data-parallel job consists of multiple stages which are connected in a directed acyclic graph (DAG). Along with the execution of a data-parallel job, the calculation and network communication take place by turn. Without loss of generality, the calculation is referred to the execution of intra-stage parallel tasks. However, current DAG scheduling methods like the critical path method [4] focus on scheduling tasks according to the task completion time but don't consider the network communication between tasks.

On the other hand, since massive amounts of intermediate data are delivered between successive tasks, such network transfers have a significant impact on the job performance, accounting for more than 50% of job completion time (JCT) [1]. The recently proposed coflow abstraction represents a group of parallel flows in a shuffle transfer and realizes the flow-level synchronization of all involved flows. However, coflow scheduling methods simply assume that JCTs consist of only coflow transmissions but don't consider the real stage duration. Since coflows are originated by completed stages, the stage duration impacts the start time of coflow transmission.

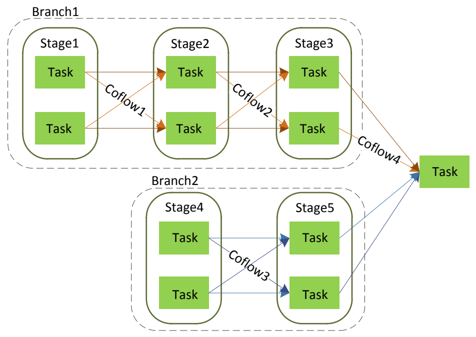


Figure 1: An illustrative example of job multi-stage semantics.

We observe that stages are highly heterogeneous in terms of their durations and thus multiple coflows emerge at different time. This makes it less of practical significance to schedule multi-stage coflows without considering the stage duration.

In coflow scheduling problem, before we schedule some a coflow, all upstream stages must be completed first. Similarly, in DAG scheduling problem, only after receiving all coflows, the downstream stage can be executed. Such a mutual restriction between coflow and DAG scheduling would incur suboptimal JCTs. For example, Fig. 1 illustrates the directed acyclic graph (DAG) of one job, which consists of 5 stages and 4 coflows. Given that Coflow3 is the shortest, existing coflow scheduling methods would schedule Coflow3 with a higher priority. However, in order to complete the job as soon as possible, the critical path algorithm would give a higher priority to Coflow1 and Coflow2. The illustrative example shows the divergence between DAG job scheduling and coflow scheduling.

2 BRANCH ABSTRACTION

To solve the problem, we propose a new abstraction, the branch. Compared with coflows, the branch considers both the stage duration and the coflow duration.

DEFINITION 1 (BRANCH). *For any DAGs $G(V,E)$ which have fork and merge nodes like Fig. 2, one branch is a DAG component with one or more nodes which are connected with each other. Each branch is a disjoint path which is a concatenation of nodes.*

The confluence of branches form the branch synchronization, which imposes significant impacts on the JCT. For example, Sycronization3 in Fig. 2 is the ultimate branch synchronization and its duration is equal to the JCT.

DEFINITION 2 (BRANCH SYNCHRONIZATION). *In the DAG, one directed edge represents the group of data flows. For any node $v \in V$ which is connected with multiple upstream nodes $\{p_i | i >= 1\}$, the execution of v is triggered only after data flows from $\{p_i | i >= 1\}$ are received. For the node v, the branch synchronization is the process that all involved branches of v are executed until they are completed. Note that, the branch synchronization can be nested.*

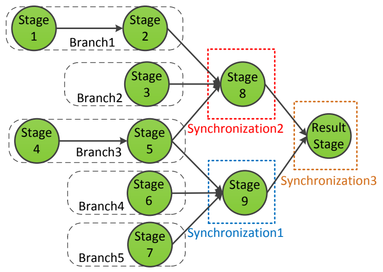


Figure 2: An illustrative example of branches and branch synchronizations.

Such an abstraction doesn't separate the stage and coflow communication. It reconciles the mutual restriction between coflow and DAG scheduling. Via the scheduling preferences of branches, we know which coflows or stages should be scheduled with higher priorities. Moreover, the branch can exhibit the job-level semantics like the urgency of branches. The scheduling preference of branches influnces the exceeding time of the ultimate branch synchronization i.e. the completion time of one DAG. In this paper, they're used as heuristic rules to design the branch scheduling method.

3 BRANCH SCHEDULING PROBLEM

We observe that multiple branches in a branch synchronization are usually heterogeneous in term of their durations, which provides nice opportunities to optimize branch scheduling. We're inspired to delay short and less urgent branches in one long synchronization so that the resource can be utilized to execute other jobs.

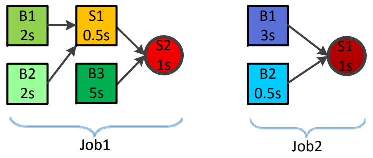
Fig. 3 shows an illustrative example of scheduling two jobs. Their DAGs are shown in Fig. 3(a). For simplicity, we assume that the resource consumption of all branches is one-third of the total resource. That is, at most three branches are concurrently executable. Note that, Job1 and Job2 are submitted at 0 and 0.5s. Partial resource which is marked with gray color isn't available at the beginning. Fig. 3(b) illustrates the critical path method. We can see that the average JCT is 6s.

Fig. 3(c) illustrates the branch scheduling method. Differently, the branch scheduling method determines not to execute B2 of Job1 at 0.5 second. Although we execute B1 and B2 of Job1 in serials, Job1 wouldn't be prolonged because the expected shortest completion time of Job1 is determined by B3. Similarly, the scheduler determines to execute B2 of Job2 rather than B2 of Job1 at the 2nd second. If B2 of Job1 is executed at the 2nd second, the expected shortest completion time of Job1 doesn't decrease but that of Job2 would be prolonged. As a result, the heuristic method exhibits the shorter average JCT than the critical path method.

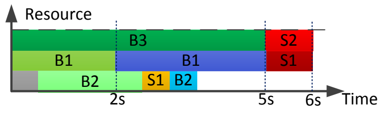
The above example shows it's unnecessary to concurrently execute all branches of a job, and short and less urgent branches should be delayed to save the resource for other jobs. The key insight of the branch scheduling is to complete a job as in serials as possible and meanwhile to achieve the expected shortest completion time of the job. To sum up, the branch scheduling method aims at viewing one DAG as an ultimate branch synchronization and scheduling it with the least exceeding time.

4 ENHANCEMENT

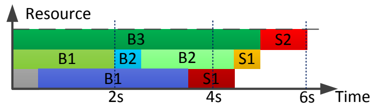
In practice, we also concern the resource utilization. If the sum of resource requirements of concurrent branches is less than the resource capacity, branches are parallelizable and can be packed together. However, the resource requirement of branches is time-varying and



(a) DAGs of Job1 and Job2 are represented as branches. Job1 and Job2 are submitted at 0 and 0.5s, respectively. We assume that all branches consume one-third of the total resource for simplicity so that discussions are mainly concentrated to the dependency.



(b) The critical path method exhibits the average JCT of 6 seconds. In Job1, branches on the critical path are given higher priorities than other branches, namely, long branches are scheduled prior to short branches. The gray label means the bsuy resource which is occupied by other jobs.



(c) The branch scheduling method exhibits the average JCT of around 5 seconds. Branch2 of Job1 is delayed because it's not urgent to synchronize with Branch3 of Job1. Thus, we execute Branch2 and Branch3 of Job1 in series without increasing the duration of Job1. This save resource for Job2.

Figure 3: Comparing the critical path method with branch scheduling method for two jobs.

changes with the running stage in the branch. This makes it intractable to pack branches in our branch scheduling method. Prior work [2] calculated a dot product between the task resource vector and the machine's available resource vector and packed tasks to certain machines. Nevertheless, the method considers the resource vector of tasks is time-invariant. To pack branches, we express the resource requirement of one branch in the form of a time-varying resource vector. If the sum of time-varying resource vectors of multiple branches is less than the resource capacity anytime, we pack these branches together in theory. In this way, we guarantee that the available resource is sufficient for concurrently-running stages of these branches.

5 FUTURE WORK

First, we propose a DAG analyzing algorithm to convert a DAG job to a set of branches. Second, we define the expected shortest completion time of one job as its timing budget and quantify the urgency of one branch. Then, we design one prediction model for the branch duration, solve the dynamical resource demands of branches and the explicit branch scheduling method which involves the branch packing and matching. We plan to implement the branch scheduling method in the Apache Spark Platform and compare it with the state-of-the-art DAG scheduling for data-parallel jobs, CARBYNE [3].

REFERENCES

[1] Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I Jordan, and Ion Stoica. 2011. Managing data transfers in computer clusters with orchestra. *Acm Sigcomm Computer Communication Review* 41, 4 (2011), 98–109.
[2] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2014. Multi-resource packing for cluster schedulers. In *ACM Conference on SIGCOMM*. 455–466.
[3] Robert Grandl, Mosharaf Chowdhury, Aditya Akella, and Ganesh Ananthanarayanan. 2016. Altruistic Scheduling in Multi-Resource Clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.* 65–80.
[4] James E. Kelley Jr. 1959. Critical-Path Planning and Scheduling: Mathematical Basis. *Operations Research* 9, 3 (1959), 296–320.