Scheduling Jobs across Geo-Distributed Data Centers

Project for Algorithm and Complexity

Bo Li

Department of Computer Science and Engineering,
The Hong Kong University of Science and Technology

Abstract. This project introduces the problem of scheduling jobs across geo-distributed data centers. It includes the introduction, background and challenges of this problem. Finally, it lists all the requirements and rules for each group. Please read this document carefully and complete the corresponding tasks.
Keywords: Geo-distributed Data Center Networks, Graph Algorithm, Scheduling.

1 Data Analytic Job Scheduling Problem

It has now become commonly accepted that the volume of data —from end users, sensors, and algorithms —has been growing exponentially, and mostly stored in geographically distributed data centers around the world. *Big data processing* refers to applications that apply machine learning algorithms alike to process such large volumes of data, typically supported by modern data-parallel frameworks such as MapReduce and Spark. Big data processing has become a routine in governments and multinational corporations, especially those in the business of social media and Internet advertising.

A data analytic *job* typically proceeds in several computation stages where there are precedence constraints among them. Each stage may consist of a number of computation tasks that are executed in parallel. For example, there are only one stage for both job A and job B in Figure 1, and each stage contains two parallel tasks. To start a new computation stage, intermediate data from the preceding stage needs to be obtained, which may initiate multiple network flows.

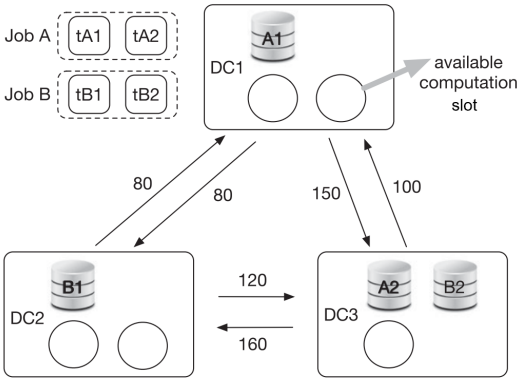

Fig. 1. An example of scheduling multiple jobs across geo-distributed data centers, retrieved from [1].

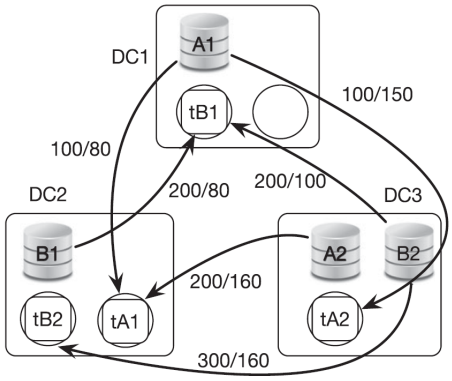

Fig. 2. A possible assignment of jobs for the problem in Figure 1, retrieved from [1].

Figure 1 shows an example of scheduling multiple jobs across geo-distributed data centers. For job A, both of its tasks, tA1 and tA2, require 100 MB of data from input dataset A1 stored in DC1 (Data Center 1), and 200 MB of data from A2 located at DC3. For job B, the amounts of data to be read by task tB1 from dataset B1 in DC2 and B2 in DC3 are both 200 MB; while task tB2 needs to read 200 MB of data from B1 and 300 MB from B2. These tasks are to be assigned to available computing slots in the

three data centers, each with two slots, two slots, and one slot, respectively. The amounts of available bandwidth of inter-datacenter links are illustrated in the figure, with the unit of MB/s. The bandwidth of inner-datacenter links, though not shown in the figure, is 1000 MB/s. The goal is to find a placement of data, intermediate results and computation tasks so that the average completion time of two jobs is minimized.

Figure 2 gives a feasible solution of the problem. The job completion time is determined by the network transfer time and execution time of each task. For simplicity, we assume that all tasks take 1 second to execute. For the assignment of job A, task tA2 is assigned to the only available computing slot in DC3, and tA1 is placed in DC2, which result in a network transfer time of $\max\{100/80, 200/160, 100/150\} = 1.25$ seconds. For the assignment of job B, DC1 and DC2 are selected to distribute task tB1 and tB2, respectively, resulting in the transfer time of $\max\{200/80, 200/100, 300/160\} = 2.5$ seconds. The average completion time is thus $(2.25 + 3.5)/2 = 2.875$ seconds.

Traditionally, we transfer all the data that are geographically distributed to be processed to a single data center, so that they can be processed in a centralized fashion. However, this wisdom becomes practically unfeasible nowadays. First, it may not be practical to move user data across country boundaries, due to legal reasons or privacy concerns. Second, the cost, in terms of both bandwidth and time, to move large volumes of data across geo-distributed data centers may become prohibitive as the volume of data grows exponentially.

A better design may be to move computation tasks to where the data is, so that data can be processed locally within the same data center. Of course, the intermediate results after such processing may still need to be transferred across data centers. In this case, designing the best possible task assignment strategy to assign tasks to data centers is important, since different strategies lead to different flow patterns across data centers, and ultimately, different job completion times.

Your team is assigned to design an algorithm to tackle this challenge. To do so, you will have to consider many questions. How hard is it to find an optimal scheduling algorithm? How to determine the data and job placement across geo-distributed data centers? How to balance fairness in terms of resource allocation and efficiency when scheduling multiple jobs? Please keep theses questions in mind and accomplish the detailed tasks in Section 2.

2 Single Job Scheduling

Before we tackle the multi-job scheduling problem, let us first learn some wisdom from the algorithms for single-job scheduling. We introduce the DAG scheduling model as follows.

A data analytic job, which is essentially a parallel program, can be represented by a directed acyclic graph (DAG) $G = (V, E)$, where V is a set of v nodes and E is a set of e directed edges. A node in the DAG represents a task of the job. The weight of a node v_i is called the computation cost and is denoted by w_i. The edges in the DAG, each of which is denoted by (v_i, v_j), correspond to the communication messages and precedence constraints among the nodes. The weight of an edge is called the communication cost of the edge and is denoted by $c(v_i, v_j)$. To make it clear, assume there is another task tA3 in job A in the problem introduced in Figure 1, and tA3 needs 100 MB from intermediate result of tA1. If we place tA3 in the empty slot of DC1 in Figure 2, then $c(v_{tA1}, v_{tA3})$ would be $100/80 = 1.25$ seconds. The source node of an edge is called the parent node while the sink node is called the child node. A node with no parent is called an entry node and a node with no child is called an exit node.

The precedence constraints of a DAG dictate that a node cannot start execution before it gathers all of the messages from its parent nodes. The communication cost between two tasks assigned to the same processor (i.e. abstracted to be computation slots in Figure 1 and 2, we now use the term *processor* and *slot* interchangeably) is assumed to be zero. If node v_i is scheduled to some processor, then $ST(v_i)$ and $FT(v_i)$ denote the start-time and finish-time of v_i, respectively. After all the nodes have been scheduled, the schedule length is defined as $\max_i\{FT(v_i)\}$ across all processors. The goal of scheduling is to minimize $\max_i\{FT(v_i)\}$. Note that this goal of min-max is different from the previous one.

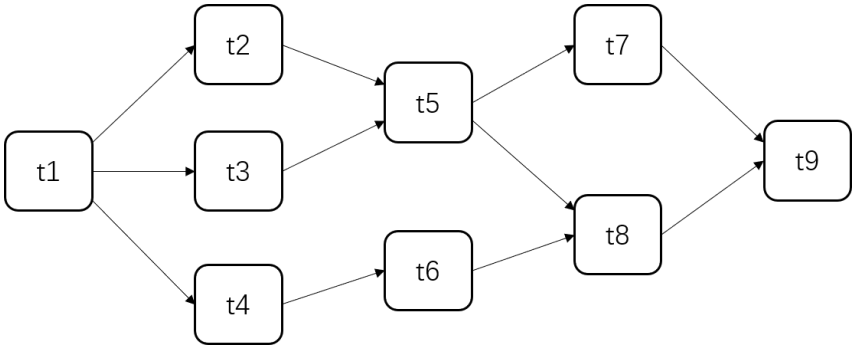

Fig. 3. An example of a data analytic job consisting of multiple stages.

Figure 3 shows a data analytic job whose precedence constraints are more complicated than the previous one. It has in all 7 tasks and 5 stages. Most importantly, only when an old stage is finished may a new stage start to be computed.

In DAG scheduling, it is assumed that the processors do not share memory and communication relies solely on message-passing. Note that different processors in the same data center can also raise communication cost, since the inner bandwidth is large yet not infinity. For simplicity, suppose all processors to be homogeneous, having the same speed and processing capability. The processors are connected by an interconnection network with a certain topology. The topology may be fully connected or of a particular structure such as a hypercube or mesh. Figure 1 is essentially a fully connected network for the processors.

3 Task and Requirements

3.1 Task: Multi-Job Scheduling with Max-Min Fairness

Consider you are scheduling multiple jobs with limited computing slots in each data center. In this case, different jobs may be competing for resources. We want the resource to be allocated *fairly*. To quantify fairness, there is a notion called *max-min fairness* [5,8].

Max-Min Fairness Consider a network composed of a set V of nodes, a set E of links with given capacities C_e ($e \in E$), and a set D of connections (demands). Each connection d ($d \in D$) is assigned a predefined path P_d between its end nodes; each such path is identified with the set of links it traverses, i.e., $P_d \subseteq E$, $d \in D$. Now let x_d denote the bandwidth allocated to path P_d and $\mathbf{x} = (x_d : d \in D) \in \mathbb{R}^{|D|}$ be the corresponding allocation vector. We are interested in finding a feasible allocation vector $\mathbf{x} \in X$ (the set of all feasible allocation vectors will be denoted by X) which is fair.

Clearly, vector $\mathbf{x}$ is feasible if $\mathbf{x} \geq \mathbf{0}$ and

$$\sum_{d \in D: e \in P_d} x_d \leq C_e, \quad e \in E,$$

which is called the capacity constraint; it assures that for any link e its load does not exceed its capacity. We say that an allocation vector $\mathbf{x}^0$ is said to be max-min fair in X if $\mathbf{x}^0 \in X$ and $\mathbf{x}^0$ fulfills the following property:

For any allocation vector $\mathbf{x} \in X$ *and for any connection* d *such that* $x_d > x_d^0$ *there exists a connection* d' *such that* $x_{d'} < x_{d'}^0 \leq x_d^0$.

To get more intuition of the definition, let us look at a simple example. Three persons want to share 100cl of beer in three glasses of different volumes, respectively 25, 40, and 50cl. Obviously the most fair sharing given the volumes of glasses would be the one given in the Figure 4. All people get the amount

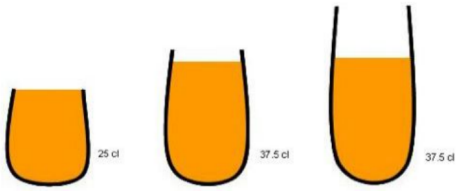

Fig. 4. Example of fair beer sharing between three persons.

not exceeded than required, and for those whose demand is not satisfied, the allocation among them is fair.

Now back to the scheduling problem. The goal is to minimize the **average** completion time of all jobs while maintaining max-min fairness on computation resource. Please accomplish following tasks:

1. Formalize the multi-job scheduling problem with notations. Especially, formalize the max-min fairness on computation resource in this problem.
2. How hard is this problem? Is it in P, or NP, or NP-Complete?
3. Please design an algorithm for this problem and analyze its complexity.
4. Test you algorithm on the attached toy data. Visualize your result to illustrate your design of algorithm if possible.
5. Test the efficiency of your design by simulations. You can collect data from open-source websites or generate data based on your understanding of the problem. If you collect data, please state where you find it and explain why it is suitable for testing. If you generate data, please briefly explain how you generated data based on your assumptions.

4 Report Requirements

You need to submit a report for this project, with the following requirements:

1. Your report should have the title, the author names, IDs, email addresses, the page header, the page numbers, figure for your simulations, tables for discussions and comparisons, with the corresponding figure titles and table titles.
2. Your report is English only, with a clear structure, divided by sections, and may contain organizational architecture like itemizations, definitions, or theorems and proofs.
3. Please include reference section and acknowledgment section. You may also include your feelings, suggestion, and comments in the acknowledgment section.
4. Please define your variables clearly. If needed, a symbol table is strongly recommended to help readers catch your design.
5. Please also include your latex source and simulation codes upon submission.

Acknowledgment

This problem is motivated by Prof. Bo Li (bli@cse.ust.hk) from Department of Computer Science and Engineering at The Hong Kong University of Science and Technology, designed by Prof. Xiaofeng Gao (gao-xf@cs.sjtu.edu.cn) from Department of Computer Science and Engineering at Shanghai Jiao Tong University. Tianyao Shi from CS2016@SJTU (sthowling@sjtu.edu.cn) scribed and finalized the project. Yihao Xie from JI2016@SJTU(crystalys@sjtu.edu.cn) and Haolin Zhou from SPEIT@SJTU (kozilelo@sjtu.edu.cn) helped on literature review. Runbo Ni from SPEIT@SJTU (pqross@sjtu.edu.cn) provided many suggestions.

References

1. Li Chen, Shuhao Liu, Baochun Li, and Bo Li, Scheduling Jobs across Geo-Distributed Datacenters with Max-Min Fairness, IEEE Transactions on Network Science and Engineering (TNSE), 6(3):488-500, 2019.
2. Chen Chen, Wei Wang, and Bo Li, Performance-Aware Fair Scheduling: Exploiting Demand Elasticity of Data Analytics Jobs, IEEE International Conference on Computer Communications (INFOCOM), 504-512, 2018.
3. Yu-Kwong Kwok and Ishfaq Ahmad, Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors, ACM Computing Surveys, 31(4):406–471, 1999.
4. Jeffrey Dean and Sanjay Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, Communications of the ACM (CACM), 51(1):107–113, 2008.
5. Dritan Nace and Michal Pióro, Max-Min Fairness and its Applications to Routing and Load-Balancing in Communication Networks: A Tutorial, IEEE Communications Surveys and Tutorials, 10(1-4): 5-17, 2008.
6. Zhiming Hu, Baochun Li, and Jun Luo, Flutter: Scheduling Tasks Closer to Data across Geo-Distributed Datacenters, IEEE International Conference on Computer Communications (INFOCOM), 1-9, 2016.
7. Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica, Low Latency Geo-distributed Data Analytics, ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), 421–434, 2015.
8. Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica, Dominant Resource Fairness: Fair Allocation of Multiple Resource Types, USENIX Symposium on Networked Systems Design and Implementation (NSDI), 1-14, 2011.