Brief Announcement: Deadline-Aware Scheduling of Big-Data Processing Jobs

Peter Bodík
Microsoft Research
Redmond, WA
peterb@microsoft.com

Ishai Menache
Microsoft Research
Redmond, WA
ishai@microsoft.com

Joseph (Seffi) Naor*
CS Department, Technion
Haifa, Israel
naor@cs.technion.ac.il

Jonathan Yaniv
CS Department, Technion
Haifa, Israel
jyaniv@cs.technion.ac.il

ABSTRACT

This paper presents a novel algorithm for scheduling big data jobs on large compute clusters. In our model, each job is represented by a DAG consisting of several stages linked by precedence constraints. The resource allocation per stage is *malleable*, in the sense that the processing time of a stage depends on the resources allocated to it (the dependency can be arbitrary in general). The goal of the scheduler is to maximize the total value of completed jobs, where the value for each job depends on its completion time. We design an algorithm for the problem which guarantees an expected constant approximation factor when the cluster capacity is sufficiently high. To the best of our knowledge, this is the first constant-factor approximation algorithm for the problem. The algorithm is based on formulating the problem as a linear program and then rounding an optimal (fractional) solution into a feasible (integral) schedule using randomized rounding.

Keywords

Scheduling algorithms; big data; deadline-aware scheduling

1. INTRODUCTION

Background and Motivation. Big data processing is increasingly receiving more attention today, as many companies process huge amounts of data to gain valuable insight into data patterns and behavior that previously were not observable. Frameworks such as MapReduce [3] or Cosmos [2] run on tens of thousands of machines and schedule jobs that process terabytes or even petabytes of data. These jobs are often used for business critical decisions and have strict deadlines associated with them. For example, outputs of some jobs are used by business analysts; delaying job completion would significantly lower their productivity. In other cases, a job computes the charges to customers in cloud computing settings, and delays in sending the bill might have serious business consequences. If the output of a job is used by external customers, missing a deadline often results in actual financial penalty.

In practice, production jobs vary significantly in many aspects. First, they vary in *urgency*: some jobs cannot suffer delays, whereas other jobs have looser time constraints and can be pushed back. Second, they vary in *utility*: some jobs are more important than others, and as a result users have higher value assessments for their jobs meeting predefined deadlines. Finally, they vary in *structure*. Big data jobs typically have complex internal structure which makes their scheduling challenging. Jobs are composed of computation *stages*, where each stage represents a logical operation on the data, such as extraction of raw data, filtering of data, or aggregation of certain columns. The number of stages varies between jobs; for example, MapReduce jobs have only two stages, while several large production jobs in Cosmos can have up to hundreds of stages. Stages are linked by input-output dependencies that induce precedence constraints between stages. These constraints form a directed acyclic graph (DAG) structure that must be preserved when scheduling the job.

However, currently used schedulers in production typically do not support hard or soft deadlines. In most cases, a user simply submits a job with a certain resource requirement which is not necessarily matched with a concrete desired completion time. The purpose of this paper is to design a deadline-aware scheduler with provable performance guarantees for executing complex job structures on large computation clusters.

The question is, up to what level of granularity should jobs be broken into? Stages themselves typically consist of numerous vertices (also known as subtasks or worker nodes) that also induce a DAG structure. In principle, one can think of a scheduler that bypasses the stage hierarchy, treating each job as a large DAG of vertices and assigning vertices to compute slots. While a vertex-level scheduler could lead in principle to efficient allocations, it might not scale to multiple big-data jobs having millions of vertices each, which are fairly common. Consequently, allocating resources at stage-level becomes an appealing scalable alternative. On the other extreme, allocating resources at the job level would significantly reduce the scale of the problem, but might lead to very inefficient schedules. For example, different stages in a job require different level of parallelism and process different amounts of data (up to five or more orders of magnitude) and should thus be treated differently by the scheduler. Rather than allocating a fixed amount of resources

*Work supported in part by the Technion-Microsoft Electronic Commerce Research Center, and by ISF grant 954/11.

SPAA'14, June 23–25, 2014, Prague, Czech Republic.

to the entire job, a stage-level scheduler can assign a different number of resources for each stage, based on the number of vertices, the amount of data to be processed within the stage, etc. Towards this end, stages are treated as *malleable* tasks, i.e., tasks that can be allocated different amounts of resources, such that increasing the number of allocated slots can reduce the stage processing time.

In this paper, we consider the problem of scheduling a set of big-data jobs on a cluster with C identical computing resources (e.g., a server or a core within a server). We summarize below the main aspects of our model.

1. Jobs consist of stages that are DAG-structured: A directed dependency edge between two stages symbolizes a precedence constraint, that is, a stage cannot be processed before its dependencies have been completed.

2. Allocations are malleable, assuming arbitrary speedups: Each stage can be allocated different amounts of resources. For example, the user may specify per stage low and high thresholds on the required amount of resources. The number of resources per stage is fixed during the execution of the stage. As mentioned, we assume that speedups can be arbitrary, i.e., the relation between the number of allocated resources and stage processing time can arbitrary. Our allocation model also enables the processing time to depend not only on the number of resources dedicated to processing the stage, but also on the specific time during which the stage started its processing. For example, the processing time can increase during times when the cluster is known to be congested.

3. Value gained by job completion depends on completion time: Each job j is associated with a value function $v_j(t)$ which represents the value gained by completing job j at a time t. The value functions can be arbitrary. Note that our model generalizes the single deadline scenario[1], as it allows for several "soft" deadlines when delayed completion time is still somewhat valuable to the user. The objective of the scheduler is to find a feasible assignment maximizing the total value extracted from fully completed jobs.

4. Jobs are known in advance (offline allocation model): In practice, deadline-bound jobs tend to be *recurring*, i.e., they are scheduled periodically, e.g., on an hourly or daily basis. One can thus use the execution statistics from past instances of these jobs to produce per-stage response curves which can serve as input to the scheduler. For example, [4] describes a method for estimating the duration of a stage with n vertices, by forming empirical distributions for each vertex which rely on past executions of the stage, and then estimating the stage duration via Monte Carlo simulations, where vertex latencies are drawn from the per vertex distributions.

Related Work. Scheduling problems of DAG-structured jobs have been widely studied in the parallel processing literature (see [9] for a survey). More related to our work, papers on scheduling DAG jobs with malleable tasks focus mainly on global system objectives such as makespan minimization (see [5, 8] and references therein). Recently, a deadline-aware scheduler for malleable tasks has been proposed [4], however the paper only proposed heuristics for the single job case. To the best of our knowledge, the objective of maximizing aggregate (completion-time) values has not been considered in the literature.

The value maximization objective has been previously considered by [6, 7, 10, 11, 1]; however, these papers treated each job as a single entity, while abstracting away inner stage dependencies and stage malleability. Such simplifications might result in inefficient resource allocation, especially for jobs with heterogenous stage profiles.

[1] A single strict deadline d_j can be modeled by setting $v_j(t)$ to some constant for $t \leq d_j$ and 0 otherwise.

2. PROBLEM STATEMENT

System. The system consists of a computing cluster containing C identical compute units (we use the terms compute units, resources, servers interchangeably). We assume that the timeline is divided into a discrete set of slots $1, 2, \ldots, T$ and that all of the servers are available throughout each time slot. The cluster receives a set of job processing requests. We consider the offline allocation model, in which all jobs are fully known in advance, and the goal is to schedule the jobs on the C servers during the time slots $1, 2, \ldots, T$.

Jobs. Each job j submitted to the system is described by a directed acyclic graph $G_j = (V_j, E_j)$. Nodes of the graph represent stages of the job, while edges represent the dependencies between stages. For clarity, we use the term "node" instead of "stage" to maintain consistency with graph notation. Each job j consists of n_j nodes, denoted $V_j = \{1, 2, \ldots, n_j\}$. We assume that the nodes in G_j are topologically ordered, such that $(v, v') \in E_j$ implies $v < v'$. An edge $(v, v') \in E_j$ in the graph symbolizes a precedence constraint between nodes, meaning, node v' cannot begin its execution before node v has been completed. We define the *width* ω_j of a directed acyclic graph G_j as the largest number of nodes in G_j that can be simultaneously processed without violating precedence constraints. Denote by $n = \max_j\{n_j\}$ and $\omega = \max_j\{\omega_j\}$ the largest graph size and graph width of the input jobs.

Each job is associated with a value function $v_j(t)$ that specifies the value gained by fully completing job j at any time t. The completion time of a job is defined as the latest completion time of all its nodes. The value functions are given explicitly to the scheduler, which attempts to maximize the total gained value.

Node Allocations. Allocations of resources to nodes are shaped as rectangles. A rectangle A describes an allocation of resources to a node during a time slot interval $[s(A), e(A)]$, where $s(A)$ and $e(A)$ are the processing start and end times, respectively. The height of the rectangle $k(A)$ represents the number of resources allocated to the node. A tuple $(k(A), s(A), e(A))$ is termed a *node allocation* and is denoted by A. For a time slot t, we shorten the notation of $t \in [s(A), e(A)]$ to simply $t \in A$.

Each job j specifies a set $\mathcal{A}_{j,v}$ of feasible node allocations per node[2] $v \in V_j$. To allocate node v, the system must choose exactly one node allocation from $\mathcal{A}_{j,v}$. We note that we make no additional assumptions on the relation between the number of allocated resources and the node processing time, though in practice the processing time typically decreases in the number of allocated resources. We denote by $k = \max\{k(A) \mid j, v, A \in \mathcal{A}_{j,v}\}$ the largest number of resources that may be allocated to a node.

3. DAG SCHEDULING

We present the first offline approximation algorithm for scheduling DAG-structured jobs with malleable stages to maximize the total value of completed jobs. The algorithm guarantees an expected constant approximation factor when $C = \Omega(\omega k \log n)$. The approximation algorithm is based on a randomized rounding technique, where a relaxed fractional formulation of the problem is optimally solved and then rounded to a feasible schedule of the DAG-structured jobs via randomized methods. The algorithm consists of four steps, each summarized next.

[2] Each set $A_{j,v}$ is specified by the job owner and is part of the input to the problem. We note that the set $\mathcal{A}_{j,v}$ need not necessarily include all possible node allocations; in practice it may include only a small subset of "attractive" allocation options as perceived by the job owner.

Step 1: Linear Program. We first solve a relaxed formulating of the DAG scheduling problem as a linear program. We define a variable $x(A) \in [0,1]$ for each node allocation $A \in \mathcal{A}_{j,v}$ of job j and node $v \in V_j$, denoting the fractional allocation of A.

$$\begin{aligned}
\max \quad & \sum_{j} \sum_{A \in \mathcal{A}_{j,n_j}} v_j(e(A)) \cdot x(A) \\
\text{s.t.} \quad & \sum_{A \in \mathcal{A}_{j,v}} x(A) \;\leq\; 1 && \forall j, v \\
& \sum_{j,v} \sum_{\substack{A \in \mathcal{A}_{j,v}: \\ t \in A}} k(A) \cdot x(A) \;\leq\; C && \forall t \\
& \sum_{\substack{A \in \mathcal{A}_{j,v}: \\ e(A) < t}} x(A) \;\geq \sum_{\substack{A' \in \mathcal{A}_{j,v'}: \\ s(A') \leq t}} x(A') && \forall j, (v,v') \in E_j, t \\
& x(A) \;\geq\; 0 && \forall j, v, A \in \mathcal{A}_{j,v}
\end{aligned}$$

The first two set of constraints are standard demand and capacity constraints, and the final set of constraints are the *precedence constraints* of the linear program. We define the value of a fractional solution x as the value of the objective function obtained by x.

Rounding a fractional solution x can be very difficult due to the inherit structure of x. Specifically, the support of x may contain two node allocations that cannot coexist in a feasible schedule, since their existence violates a dependency constraint; see Fig. 1 for an example. To overcome this difficulty, we first extract meaningful job allocations from x (see step 3 for definition) and then round the job allocations (step 4). Before decomposing x, we apply a preliminary correcting step called balancing (step 2).

Step 2: Balancing. The balancing step is a preliminary step used to simplify the decomposition of a fractional solution x, as described in step 3. For a fractional solution x, job j and node $v \in V_j$, we define $x_{j,v} = \sum_{A \in \mathcal{N}_{j,v}} x(A)$ as the total completed fraction of node v according to x.

DEFINITION 1. *A fractional solution x is called* balanced *if $x_{j,v} = x_{j,v'}$ for every job j and nodes $v, v' \in V_j$.*

LEMMA 3.1. *Every fractional solution x can be balanced in polynomial time without changing the value of x.*

Step 3: Decomposing a Balanced Solution. We decompose x^* into *job allocations*, which are eventually used to construct the rounded solution. A decomposition can be viewed as an alternate representation of a fractional solution in which node allocations in x are grouped into job allocations; see Fig. 1 for an example.

DEFINITION 2. *A* job allocation *of a job j is a set of node allocations $J = \{A_1, A_2, \ldots, A_{n_j}\}$, one allocation $A_v \in \mathcal{A}_{j,v}$ per node $v \in V_j$, which satisfies allocation precedence constraints. Formally, for every dependency edge $(v,v') \in E_j$ and corresponding node allocations $A_v, A_{v'} \in J$, we have $e(A_v) < s(A_{v'})$.*

DEFINITION 3. *A decomposition of a balanced fractional solution x is a tuple $(\mathcal{S}, y)$. The decomposition consists of a set $\mathcal{S}$ of job allocations and a mapping $y : \mathcal{S} \to [0,1]$ that satisfies for every job j, node $v \in V_j$ and node allocation $A \in \mathcal{A}_{j,v}$:*

$$x(A) \;=\; \sum_{J \in \mathcal{S}: A \in J} y(J). \tag{1}$$

LEMMA 3.2. *A decomposition $(\mathcal{S}, y)$ of a balanced fractional solution x can be generated in polynomial time.*

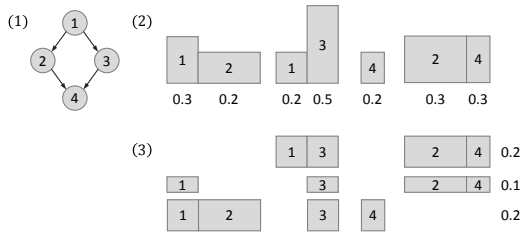

Figure 1: The figure shows an example of a job j and its fractional allocation. (1) DAG structure of job j; (2) fractional solution x (only job j shown); (3) weighted decomposition of x.

Step 4: Randomized Rounding. Our algorithm decomposes an optimal fractional solution x^* and randomly selects job allocations from the decomposition. We note that randomly generating a feasible solution, while obtaining an expected high value, is non-trivial and requires several algorithmic insights.

THEOREM 3.3. *The DAG scheduling problem admits a randomized approximation algorithm that obtains an expected approximation ratio of $\alpha(\lambda)$ for every $\lambda > 0$, where:*

$$\alpha(\lambda) \;\triangleq\; \frac{1}{\lambda} \cdot e^{-\frac{1}{\lambda}} \cdot \left[1 - e^{-\frac{\left(1-\frac{1}{\lambda}\right)C-k}{2\omega k} \cdot \ln\left(\lambda \cdot \left(1-\frac{k}{C}\right)\right)}\right]^n .$$

4. REFERENCES

[1] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM (JACM)*, 48:1069–1090, 2001.

[2] R. Chaiken, B. Jenkins, P. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE: Easy and Efficient Parallel Processing of Massive Datasets. In *VLDB*, 2008.

[3] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI*, 2004.

[4] A. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca. Jockey: guaranteed job latency in data parallel clusters. In *ECCS*, pages 99–112. ACM, 2012.

[5] E. Günther, F. G. König, and N. Megow. Scheduling and packing malleable and parallel tasks with precedence constraints of bounded width. *J Combinatorial Optimization*, 27:164–181, 2012.

[6] N. Jain, I. Menache, J. Naor, and J. Yaniv. A truthful mechanism for value-based scheduling in cloud computing. In *SAGT*, pages 178–189, 2011.

[7] N. Jain, I. Menache, J. Naor, and J. Yaniv. Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters. In *SPAA*, pages 255–266, 2012.

[8] K. Jansen and H. Zhang. Scheduling malleable tasks with precedence constraints. *Journal of Computer and System Sciences*, 78(1):245–259, 2012.

[9] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, 31(4):406–471, 1999.

[10] B. Lucier, I. Menache, J. Naor, and J. Yaniv. Efficient online scheduling for deadline-sensitive jobs: extended abstract. In *SPAA*, pages 305–314, 2013.

[11] C. A. Phillips, R. N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *SODA*, pages 879–888, 2000.