Exploiting Time-Malleability in Resource Allocation for Cloud Computing

Luo Mai
Imperial College London

Evangelia Kalyvianaki
Imperial College London

Paolo Costa
Microsoft Research Cambridge

Abstract

Cloud computing has revolutionized the IT landscape, providing small and large companies as well as individual users with the ability of running "big data" jobs on virtually unlimited resources. Recently, there have been several proposals, both by researchers and practitioners, to enable users to specify high-level goals, e.g., job completion time, and let the cloud provider decide which resources to allocate to jobs to meet these goals. However, existing approaches constrain users into specifying a single completion time, often accompanied with a single relative price. This drastically limits the flexibility of the provider for job allocation, thus hampering its ability to accommodate jobs and, hence, reducing its revenue.

To address these shortcomings, in this paper we propose a novel flexible pricing model and allocation plans that aim to maximize job acceptance ratio and provider's revenue. Our approach exploits the property that most batch-processing jobs are *time-malleable*, i.e, their completion time depends on the number of resources allocated to them, which can be changed at run-time. Based on this property we propose a flexible pricing model where users specify a range of acceptable completion times spanning from a *soft* up to a *hard* deadline and the eventual price depends on the actual completion time of a job. Preliminary results show that compared to today's practices our approach can increase revenue by up to 75 % and can accept up to 62% more jobs.

1 Introduction

Most cloud computing platforms support the Infrastructure-as-a-Service (IaaS) model where users specify required resources for their jobs and the provider charges them accordingly. While simple and intuitive, this interface can become cumbersome for users, as in many cases they lack the necessary IT skills to accurately identify the resources needed. It also limits the flexibility of the provider to perform resource multiplexing across users. This negatively affects resource utilization, leading to lower revenue.

To address these shortcomings, recent systems, including both research prototypes [5–8, 10–12] and commercial products such as Amazon DynamoDB [13], support higher-level interfaces where users specify job-specific performance goals, e.g., application throughput or job completion deadlines, and the provider decides what resources to use to meet these goals.

Hereafter, we focus on *deadline-based* batch-processing systems, e.g., [5, 8], in which users specify the desired completion time of their jobs and the provider allocate resources accordingly. This approach is mutually beneficial. First, it simplifies user experience by providing a simple and intuitive interface, even for non-IT experts, to specify performance goals. Second, it increases provider's options on devising allocation plans and, hence, its ability to accept more jobs and increase its revenue. A major drawback of these systems, however, is that the provider is still bound to a *rigid* pricing model. In this model the provider accepts the job only if there are enough resource to meet the desired deadline, otherwise the job is rejected.

In contrast, we propose a *flexible* pricing model, in which users specify a desired *soft* deadline of the completion time along with a *hard* deadline. Our approach exploits the property that MapReduce jobs are *time-malleable*, i.e., it is possible to change their resource allocation at runtime [3]. Furthermore, due to their batch nature, typically MapReduce jobs do not have strict real-time requirements and some slack is generally tolerated. To create the right incentives, the later the job is completed, the lower the users pay. This is beneficial both for the users to reduce their costs, and for the providers to exploit this flexibility and accommodate more jobs.

In this paper, we explore the opportunities and trade-offs offered by a flexible pricing model. In Section 2, we introduce a pricing model to capture the tradeoffs between computation and revenue. Next, in Section 3, we formulate the allocation problem as a mixed integer program to maximize provider's revenue. When new jobs arrive, we solve the maximization problem and, if *i)* resources are available and *ii)* the total revenue increases, we revise the prior allocation plan to accept the new job. Our preliminary results in Section 4, show that our approach significantly improves both acceptance ratio and provider revenue.

2 Flexible Pricing Model

We employ a new *flexible pricing model* where users pay the provider according to the actual completion time of their jobs rather than whether their jobs finish before a fixed, pre-defined deadline. Our pricing model exploits time-malleability in batch-processing applications in the sense that it is acceptable to stretch the job completion time over time across a *range* of deadlines.

We describe the flexible pricing model using Figure 1. At first, for every new job denoted by j, the user specifies the earliest and longest acceptable job's completion times. We refer to the earliest time as the *soft* deadline, denoted by s_j and to the latest time as the *hard* deadline, denoted by d_j. The hard deadline must be at least as long as the soft headline. The user pays the provider if and only if his job finishes at any point in time between the job submission time and the job's hard deadline d_j. However, given that the user accepts a range of completion times, the provider charges the user commensurate to the actual completion time as shown in Figure 1: i.e., the closer the completion time is to the hard deadline the less the user pays the provider.

More formally, for every new job the provider and the user agree on a pricing model expressed by a monotonically decreasing function P_j, on the job completion time t. We consider piecewise functions of the form:

$$P_j(t) = \begin{cases} p_{\max}, & \text{if} \quad t < s_j \\ p_j, & \text{if} \quad s_j \le t < d_j. \end{cases}$$

We explain P_j using Figure 1. The pricing function divides into two parts. The first part ($t < s_j$) relates to the maximum price $p_{\max}$ the user pays if the job finishes at any time before the soft deadline s_j. For this time region the cost is *constant* and enables the user to cap his maximum payment to the provider. For the region where the completion time is between the soft and hard deadlines ($s_j \le t < d_j$) the price is strictly decreasing to t. For the latter region, Figure 1 illustrates a convex function, but concave or linear types of functions can be considered as well. Compared to a rigid pricing model, which assigns a single price value on a single deadline, our model extends payment across a range of prices.

Recently, there have been several proposals, which use offline profiling and analytic performance models, to derive the resource needed by a MapReduce job to complete within a given deadline, e.g., [5, 8, 11, 12]. We

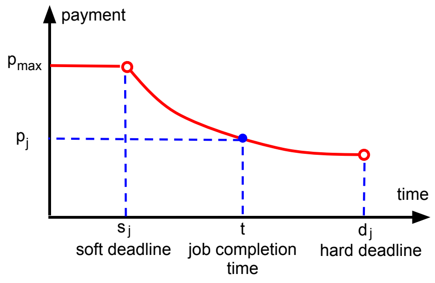


Figure 1: Example of a pricing function P_j of job j.

leverage these effort to compute the range of resources needed to complete the job across the interval of valid completion times specified by the user.

The proposed model is advantageous to both the user and the provider. It provides great incentives for the user to define a long hard deadline at the premises of a reduced price. At the same time, this model also benefits the provider to devise flexible planning decisions among all jobs. Given that a hard deadline is at least as long as a soft one, the provider can extend a job's completion time to accept more jobs and eventually increase its revenue. In the next section, we present a novel job planning approach that uses the above pricing model to increase both total revenue and acceptance ratio of incoming jobs when compared to traditional planning strategies.

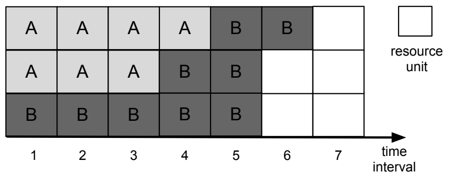
3 Cloud allocation for time-malleable jobs

Our approach exploits time malleabality of MapReduce jobs [3] to dynamically allocate resources to jobs for the provider to increase its revenue and accept more jobs.

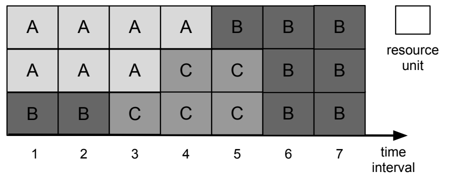
Figure 2 overviews our approach using an example with three jobs. We assume a total cloud capacity of three *units of resources* at any given time; a resource unit can be for example a VM, a CPU core or a MapReduce slot. We also divide time into fixed time intervals.

Consider jobs A and B with the following deadlines and resource requirements ($s_A = 4,\ d_A = 6,\ r_A = 7$) and ($s_B = 5,\ d_B = 7,\ r_B = 9$) where r_j denotes the total resource units required by job j. We assume offline analysis methods for calculating a job's resource requirements, e.g., [5, 11]. Figure 2(a) shows the *resource allocation plans* for A and B. We define the *allocation plan* of a job to be the number of resource units allocated to the job per time interval. We consider a plan to be *feasible* when the estimated completion time of the job lies between its soft and hard deadlines and the job is assigned with resources as requested. At this stage of the work, we ignore any potential causes of delays such as task outliers, machine or network faults [4]. Under these assumptions, both plans for A and B in Figure 2(a) are feasible.

Consider a new job C arriving at $t = 2$ with requirements ($s_C = 2,\ d_C = 3,\ r_C = 5$). For the current work, we



(a) Allocation plan for jobs A and B.



(b) Allocation plan for jobs A, B and C.

Figure 2: Allocation plans on flexible pricing models.

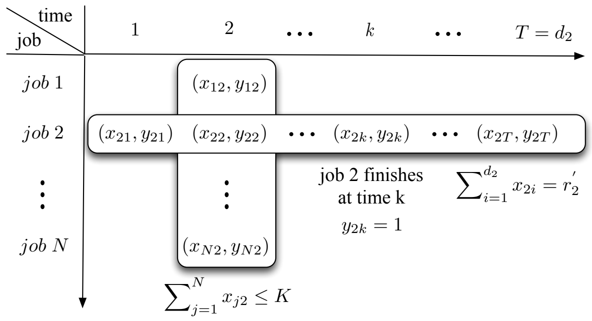


Figure 3: Example illustration of the constraints and the decision variables x and y of the maximization problem.

always denote a deadline in relative time intervals counting from the interval the job begins execution. Given the plans of Figure 2(a) job C cannot be accommodated, since by the time B completes at $t = 6$ and resources become free, it already past C's hard deadline, i.e., the sum of C's arrival time and d_C or $t = 2 + d_C = 5$ in Figure 2. Rather, our approach is to extend the completion time of running jobs to release resources towards C. To this end, one such feasible plan is to delay B by one time interval to accommodate C as shown in Figure 2(b).

Finding feasible plans to accommodate thousands of jobs is non-trivial. The problem further aggravates since we aim to find, among all possible feasible plans, the one that maximizes the total revenue and solutions should be devised within short times from the arrival of new jobs.

3.1 Time-malleable job plans

We address the job planning problem that maximizes the aggregate cloud revenue over time. We first provide an overview of our approach and discuss the admission control policy for new jobs.

Consider $N-1$ jobs running under certain allocation plans and a new job arriving and indexed by N. To find a plan for the new job, our approach also revises existing allocation plans. Assume that the new job can be accommodated as new feasible plans are found for some or even all of the N jobs. Nevertheless, it is not guaranteed that the new solution will give higher revenue over the current one of the $N-1$ jobs. This is because, with flexible pricing models, there is a tradeoff between accepting a new job and delaying already running ones. To this end, if such feasible plans are found to accommodate the N job *and* the predicted revenue is higher than the current one then, the provider accepts the new job. If however, the provider cannot find feasible plans for all N jobs or, if the new predicted revenue of the feasible plans is less than the current one then the new job is rejected.

Maximization problem formulation. To find a set of feasible plans we formulate the mixed integer programming (MIP) problem with the objective (1) to maximize the total revenue from all jobs over time, subject to constraints (2)-(6). We solve the maximization problem for a duration of $T = \arg\max_j d_j$ intervals to include the longest hard deadline among all N jobs. We use t to index intervals for the next T intervals starting from the current one pointed by $t = 1$, i.e., $t \in \{1, \ldots, T\}$. We ensure that the solution includes feasible plans via the constraints explained below. We also use Figure 3 to illustrate key points of the constraints.

$$\underset{x,y,p}{\text{maximize}} \sum_{j=1}^{N} p_j \tag{1}$$

$$\text{subject to } \sum_{i=1}^{d_j} y_{ji} = 1 \quad \forall j, \tag{2}$$

$$\sum_{i=1}^{d_j} x_{ji} = r'_j \quad \forall j, \tag{3}$$

$$\sum_{j=1}^{N} x_{jt} \leq K \quad \forall t, \tag{4}$$

$$0 \leq x_{jt} \leq K \times \sum_{i=t}^{T} y_{ji} \quad \forall j, t, \tag{5}$$

$$p_j \leq P_j \left(\tilde{t_j} + \sum_{i=1}^{d_j} i \times y_{ji} \right) \quad \forall j, \tag{6}$$

$$y_{ji} \in \mathbb{B}^{N \times T}, \; x_{ji} \in \mathbb{N}^{N \times T}, \; p_j \in \mathbb{R}_+, \tag{7}$$

$$j \in \{1, \ldots, N\}, \quad t \in \{1, \ldots, T\}. \tag{8}$$

Deadline feasibility constraint. All jobs should finish before their hard deadlines shown by constraint (2) where the binary variable $y_{ji} \in \mathbb{B}^{N \times T}$ is assigned to 1 if job j completes at time i, otherwise is set to 0, also shown in Figure 3. For every job j there should be only one $y_{ji} = 1$ at a time before its hard deadline d_j.

Resource feasibility constraints. All jobs should be given resource units according to their requirements shown by constraint (3) and exemplified in Figure 3. The decision variable $x_{ji} \in \mathbb{N}^{N \times T}$ denotes the allocation of resource units per job j at interval i. Also, $r'_j \leq r_j$ denotes the remaining resource units needed to allocate for job j given that j might be an existing running job and has already spent some resource units. The sum of allocations across jobs per interval must not exceed the total cloud capacity denoted by K and shown by constraint (4). Furthermore, the resource allocation x_{ji} itself cannot exceed the total capacity of the cloud as shown by constraint (5).

Finally, constraint (6) denotes the payment of a job given its completion time and where $\tilde{t_j}$ denotes the past running time of job j. This payment is bounded by function $P_j(t)$ whose shape in the region between deadlines can vary to policies. For instance, the provider can employ a second-degree polynomial pricing function with a derivative larger than 0, i.e., a convex function, in order to encourage users to stretch their jobs' hard deadlines by significantly reducing their cost. Or, the derivative can be smaller than 0, i.e., a concave function, when the provider doesn't wish to sacrifice the payment of a job too much if stretching its completion time.

As the search for an optimal solution for a MIP has shown to be NP-complete, we use the CPLEX state-of-the-art optimization solver [1] to find approximated solutions within short times. To address a given MIP, CPLEX employs a parallel branch-and-bound algorithm. Using the CPLEX solver we can support linear and second-degree polynomial pricing functions, while other solvers [2] support more general non-linear functions.

4 Preliminary Results

We use a discrete event simulator [14] to assess the potential of the flexible pricing model and the maximization problem formulation introduced above. While preliminary, our results indicate that by allowing users to specify both soft and hard deadlines and by exploiting the time-malleability property of MapReduce jobs, our approach increases both acceptance ratio and provider revenue.

4.1 Setup

Given the unavailability of a reference workload that fits our pricing model, we adopt a synthetic workload. While admittedly naive, we believe that this workload is a good approximation of what a real workload might be.

Deadlines. We consider a total cloud data center capacity of $K = 1,000$ units of resources. For each job j, the ratio $d^j_{min} = \frac{r_j}{K}$ indicates the shortest completion time possible, i.e., the time taken by a job if *all* cloud resources are assigned to it. We assign the initial resource requirements r_j of job j to be a randomly chosen integer between 2,500 and 7,500 resource units. This ensures that all jobs take at least more than two units of time to complete (i.e., $d^j_{min} > 2$ for all jobs). We assume that tenants will select a soft deadline that is at most a few times of d_{min}, e.g., $s_j = 5 \cdot d^j_{min}$. Similarly, we assume that a tenant is willing to extend its completion time by at most a small factor, e.g., by three. More formally, for any given job j we have $s_j \in [d^j_{min}, 5 \cdot d^j_{min}]$ and $d_j \in [s_j, 3 \cdot s_j]$. Note that the latter constraint implies that some jobs may have no flexibility at all, i.e., $s_j = d_j$. Finally, for simplicity we consider a linear pricing function P_j. We also experimented with different function shapes and different ranges for s_j and d_j, observing similar trends to the results reported below.

Inter-arrival rates. We simulate job requests arriving over time for a total duration of 10,000 seconds. We assume Poisson job arrivals with a mean arrival rate λ varying between 0.12 and 0.26 jobs/c. These values yield a data center utilization between 60% and 100% when we allocate resources using our approach, hereafter referred to as MIP.

Baselines. We compare the performance of MIP against three baselines, representative of common techniques used in today's systems:

1. SOFT: this allocation plan assigns enough resources to a job to meet its *soft* deadline s_j;

2. HARD: this allocation plans assign enough resources to a job to meet its *hard* deadline d_j;

3. EDF: this baseline implements an earliest deadline first (EDF) approach [9]. Jobs are prioritized by their deadlines and resources are assigned accordingly. Job allocations can be modified at runtime if new jobs, with tighter deadlines, are submitted.

The SOFT and HARD baselines are examples of *static* allocation plans as allocations cannot be changed at runtime. EDF, instead, is an example of a *dynamic* allocation plan (like MIP), as the resources assigned to a job can be increased (resp. decreased) when an existing job terminates (resp. a new job is submitted).

Allocation time. We run our experiments on a server with 16 Intel Xeon E5-2690 cores and 32 GB of RAM, using CPLEX version 12.4. We set the upper bound of the CPLEX execution time per job to 100 s and we set the CPLEX error rate to 1%. Across all the runs, the median CPLEX execution time is 2 s and the 95th percentile is 6 s. Since this needs to be run when a job is submitted, the overhead introduced is negligible.

4.2 Results

Figure 4 shows the percentage of jobs accepted by each method.[1] We observe that the dynamic allocation plans (MIP and EDF) can accept significantly more jobs than the static ones (SOFT and HARD). The reason is that, by being unable to re-allocate resources as new jobs come in or existing jobs terminate, the ability of static allocation plans to accept jobs is drastically reduced. This highlights the problem with today's setup and motivates our effort to explore alternative solutions.

[1]We average the results across 5 runs and the standard deviation for all experiments in this section is within 3%.

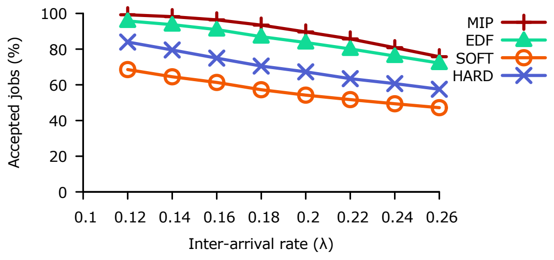


Figure 4: Jobs' acceptance ratio.

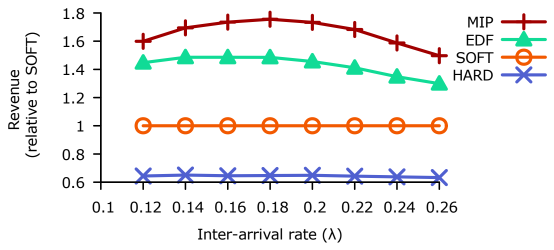


Figure 5: Total revenue (relative to SOFT).

The second metric used in our evaluation is the total revenue, shown in Figure 5. We plot the results relative to SOFT as the latter closely resembles the approach adopted in existing deadline-based systems, in which only one deadline is considered.

MIP achieves much higher revenue than SOFT (between 49.7% and 75.5%) and HARD (between 136% and 171%). This is because of MIP's ability to dynamically re-assign resources and, hence, accept more jobs. Interestingly, MIP also outperforms EDF (between 10.4% and 19.2% higher revenue), although their percentage of accepted jobs is close. The reason is that MIP admits a new job only if this increases the total revenue. This means that in some cases, even though there would be enough idle resources to accept a job, MIP can still decide to reject it, if this is not cost-effective. In contrast, if resources are available, EDF always accepts a new job, although this might be detrimental in the long term. This shows the importance of considering the total revenue as a first-class citizen in the admission control.

Although MIP achieves higher acceptance ratio and revenue, it does not significantly affect the job completion time for most of the jobs. At low load ($\lambda = 0.12$), the median job completion time increase (relative to s_j) is less than 1% (95^{th} percentile is 46%). At high load ($\lambda = 0.26$), instead, the median is 26% (95^{th} percentile is 177%). This indicates that MIP exploits time malleability only for a few jobs, while for the vast majority of them, it strives to keep completion times close to the soft deadlines. This is a consequence of the incentives set by our pricing model: the provider has a strong incentive to finish the job as fast as possible but at the same time can delay a few jobs if this increases the overall revenue, which is not possible in today's setups.

5 Discussion

In this paper, we describe a new approach for cloud-based batch-processing jobs. Unlike current proposal that support a rigid pricing model in which users can only specify one desired completion time, we propose a flexible pricing model in which users can specify both a soft and hard deadline.

In this preliminary work, we made several simplifying assumptions, including the ability to accurately derive the resources needed by a given job and its completion time, the ability to arbitrarily reassign the resources at runtime, and the absence of failures or outliers in general. While solutions to these assumptions have recently appeared in literature e.g., [3–5, 8], we have not fully evaluated them yet in the context of our work.

Even with these caveats in mind, our results are promising and show the potential benefits that our approach can bring, both to users and providers.

References

[1] IBM, ILOG CPLEX. `http://www.ibm.com`.

[2] Knitro. `http://www.ziena.com/knitro.htm`.

[3] ANANTHANARAYANAN, G., DOUGLAS, C., RAMAKRISHNAN, R., RAO, S., AND STOICA, I. True Elasticity in Multi-Tenant Data-Intensive Compute Clusters. In *ACM SoCC* (2012).

[4] ANANTHANARAYANAN, G., KANDULA, S., GREENBERG, A., STOICA, I., LU, Y., SAHA, B., AND HARRIS, E. Reining in the Outliers in Map-Reduce Clusters using Mantri. In *OSDI* (2010).

[5] FERGUSON, A., BODIK, P., KANDULA, S., BOUTIN, E., AND FONSECA, R. Jockey: Guaranteed Job Latency in Data Parallel Clusters. In *EuroSys* (2012).

[6] HENZINGER, T. A., SINGH, A. V., SINGH, V., WIES, T., AND ZUFFEREY, D. FlexPRICE: Flexible Provisioning of Resources in a Cloud Environment. In *IEEE Cloud Computing* (2010).

[7] HERODOTOU, H., DONG, F., AND BABU, S. No One (Cluster) Size Fits All: Automatic Cluster Sizing for Data-intensive Analytics. In *ACM SoCC* (2011).

[8] JALAPARTI, V., BALLANI, H., COSTA, P., KARAGIANNIS, T., AND ROWSTRON, A. Bridging the Tenant-Provider Gap in Cloud Services. In *ACM SoCC* (2012).

[9] LIU, C. L., AND LAYLAND, J. W. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM 20*, 1 (1973).

[10] TSAKALOZOS, K., KLLAPI, H., SITARIDI, E., ROUSSOPOULOS, M., PAPARAS, D., AND DELIS, A. Flexible Use of Cloud Resources through Profit Maximization and Price Discrimination. In *ICDE* (2011).

[11] VERMA, A., CHERKASOVA, L., AND CAMPBELL, R. H. ARIA: Automatic Resource Inference and Allocation for MapReduce Environments. In *ACM ICAC* (2011).

[12] WIEDER, A., BHATOTIA, P., POST, A., AND RODRIGUES, R. Orchestrating the Deployment of Computations in the Cloud with Conductor. In *NSDI* (2012).

[13] Amazon DynamoDB. `http://aws.amazon.com/dynamodb/`.

[14] OmNet++ Simulator. `http://www.omnetpp.org/`.