Need for Speed: CORA Scheduler for Optimizing Completion-Times in the Cloud

Zhe Huang[1], Bharath Balasubramanian[2], Michael Wang[2], Tian Lan[3], Mung Chiang[2], and Danny H.K. Tsang[1]
[1]The Hong Kong University of Science and Technology, Hong Kong,
[2]Princeton University, NJ, USA,
[3]George Washington University, DC, USA
Emails: {ecefelix, eetsang}@ust.hk, {bharathb, mwseven, chiangm}@princeton.edu, tlan@gwu.edu

***Abstract*—There is an increasing need for cloud service performance that can be tailored to customer requirements. In the context of jobs submitted to cloud computing clusters, a crucial requirement is the specification of job completion-times. A natural way to model this specification, is through client/job utility functions that are dependent on job completion-times. We present a method to allocate and schedule heterogeneous resources to jointly optimize the utilities of jobs in a cloud. Specifically: (i) we formulate a completion-time optimal resource allocation (CORA) problem to apportion cluster resources across the jobs that enforces max-min fairness among job utilities, and (ii) starting with an integer programming problem, we perform a series of steps to transform it into an equivalent linear programming problem, and (iii) we implement the proposed framework as a utility-aware resource scheduler in the widely used Hadoop data processing framework, and finally (iv) through extensive experiments with real-world datasets, we show that our prototype achieves significant performance improvement over existing resource-allocation policies.**

I. Introduction

In a computing cloud, efficient allocation of cluster resources (e.g., CPU-time and memory) across multiple jobs determines system performance and fairness. We address key challenges in resource allocation with a specific focus on the client's sensitivity to job completion-times. The cloud service provider will greatly benefit by prioritizing resources towards critical jobs that need to finish by a certain time as opposed to jobs that are insensitive to completion-times. While there has been extensive work on resource-allocation in data centers [1]–[3] and data processing frameworks [4]–[7], none of them addresses individual job requirements with respect to completion-times. This brings forth two hard problems: *(i)* how do we model job sensitivity to completion-times in a practically relevant manner, and *(ii)* given such a model, how can we perform optimal resource-allocation? We address these questions and formulate a resource-allocation problem with job utilities that depend on job completion-times and further, we present an efficient optimal solution to this problem.

In cloud data analysis frameworks such as Hadoop [8] and Dryad [9], multiple jobs concurrently share cluster resources such as CPU-time and memory. The state-of-the-art in resource-allocation among jobs for such systems is based on simple policies. For example, the default Hadoop scheduler uses FIFO scheduling while the most popular Hadoop scheduler, the fair scheduler [10], allocates computational resources proportional to the priority of the jobs. Since these schedulers ignore job completion-times, they may often make wrong scheduling decisions. Even deadline-aware scheduling schemes such as earliest-deadline-first (EDF) are insufficient to address our problem [11], [12]. For example, consider two jobs, where the first job specifies an early target completion time, but can tolerate longer execution time and the second job specifies a later, but much more critically important target completion time. In such a scenario, EDF will simply schedule the first job ahead of the second job, which may cause the critical job to miss its target completion time. Clearly, resource allocation schemes need to account for this heterogeneity with respect to jobs' sensitivity to completion time.

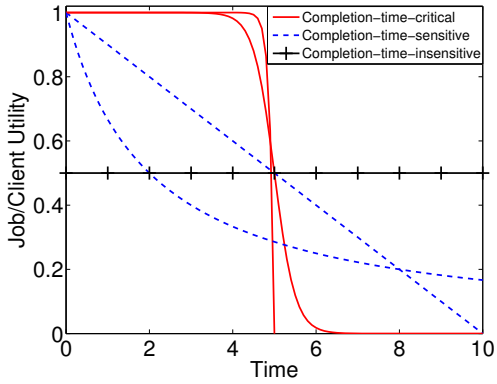

Fig. 1. Five examples of completion-time-dependent utility functions.

We can capture this sensitivity by modeling job utilities directly as a function of job completion-times. For example, a client can describe a completion-time-critical job by specifying a utility function that decays rapidly to zero (but not below as utilities are modeled as nonnegative values) after the job's target completion-time. In Figure 1, both the red curves model such utilities. On the other hand, if the job is sensitive to completion-time without being critically dependent on it, the client could specify a utility curve that decays gradually with time (e.g., blue-dashed lines in Figure 1). Finally, if the job is completely insensitive to completion-time, the client can simply specify a flat curve. Based on such job utilities, this is

This work was in part supported by DARPA grant FA8750-11-C0254, a gift from Google and a gift from HP.

978-1-4799-8381-0/15/$31.00 ©2015 IEEE

the main problem we address in this paper:

Given a set of jobs in a cloud cluster, each with heterogeneous resource requirements and a completion-time-sensitive utility, how do we optimally apportion cluster resources across the jobs?

This is a difficult problem for three reasons: (i) there is considerable heterogeneity in the cloud in terms of both resource requirements and completion-time sensitivity. An optimal allocation scheme needs to carefully balance such needs across jobs. (ii) Since job utilities are a function of completion-time, resource allocation needs to be optimized across jobs as well as time. This type of optimal allocation over the time-axis involves discrete optimization, which is often challenging. (iii) To capture client sensitivity to completion-time in a practical manner, we have to allow utilities that are neither convex nor concave (e.g., sigmoid). It further complicates the optimization problem beyond the standard literature on convex minimization. We tackle these challenges and make the following contributions:

- We formulate a resource allocation problem (referred to as CORA) in cloud computing clusters to maximize the minimum utility achieved across all the jobs in the cluster, where the job utilities are functions of their completion times. A solution to this problem apportions cluster resources across jobs, in line with job resource requirements and cluster capacity.
- While CORA is an integer programming problem, we transform the objective function to a separable convex function and exploit the total unimodularity structure of the solution space to reformulate CORA as an equivalent linear programming problem, which can be solved efficiently and optimally to achieve max-min fairness across job utilities.
- We implement the proposed framework as a new job-utility-aware resource scheduler in the Hadoop data processing framework (code is available at [13]).
- We compare the performance of our scheduler in Hadoop with commonly used scheduling policies such as FIFO scheduling, Hadoop fair scheduler scheduling, earliest-deadline-first (EDF) scheduling and utility-based risk-reward heuristic (RRH) scheduling [14]. Through extensive experiments performed on our Hadoop cluster, with the PUMA suite [15] for MapReduce, we illustrate the efficacy of our scheduler, in terms of both utility and adherence to completion-times.

The rest of the paper is organized as follows. In section II we define our main problem and solution technique. In section III we describe the implementation of our scheduler in the Hadoop framework and present the evaluation results. In section IV we discuss other areas of work relevant to this paper. Finally we conclude the paper and present directions for future work in section V.

II. Completion-Time-Optimal Resource Allocation

In this section, we first describe our system model, followed by the problem formulation and our efficient optimal solution technique. We consider a cloud cluster with different resources, denoted by the set $\mathcal{R}$, and C^r amount of each resource type $r \in \mathcal{R}$. For example, $\mathcal{R}$ could be $\{\text{cores, memory}\}$ with the cluster containing $C^{cores} = 100$ and $C^{memory} = 400,000$. A set of jobs, $\mathcal{N} = \{1, ..., N\}$, are submitted to this cluster at different times, with job i requiring R_i^r amount of type $r \in \mathcal{R}$ resource to complete. We assume a slotted-time model, in which each job is allocated a share of cluster resources in each time slot[1]. A job finishes executing when it receives the necessary amount of resources across the time slots. The completion-time of job i, denoted by T_i, is the number of time slots it takes for the job to receive all the resources it requires.

Each job i also specifies a utility function, $U_i(T_i)$ which captures its sensitivity to its completion-time. For example, the utility function of a completion-time-critical job will drop very fast in value after the job's execution time passes the target completion-time. Since each job would prefer an earlier completion-time, the utility function is strictly non-increasing. Note that we make no other assumption on the utility function apart from this. This flexibility allows our proposed model to handle a wide range of completion-time-sensitivities.

The cluster allocates $x_{i,t}^r$ resources of type r to job i in time slot $t \in \mathcal{T}$, where $\mathcal{T}$ is the set of time slots during which the cluster operates. Its size (i.e., $|\mathcal{T}|$) is chosen to be large enough to finish all the jobs. We denote the vector of $x_{i,t}^r$ by $\mathbf{X}$. Since the resources allocated to a job cannot be infinitesimal, without loss of generality, we assume that the minimum unit of resource-allocation is one.

In this model, the main problem we consider is the following: when a scheduling decision needs to be made, how should the scheduler allocate resources among the jobs (i.e., decide on $x_{i,t}^r$), to achieve lexicographic max-min fairness among jobs' utilities. In other words, we wish to maximize the worst-case utility across all the jobs, and then sequentially maximize the next-worst utility as long as it does not affect the previous worst-case utility. We formally model this as the following completion-time optimal resource-allocation (CORA) problem:

$$\text{(CORA)}$$

$$\underset{\mathbf{X}}{\text{lexmax}} \quad \min_i (U_i(T_i)) \tag{1}$$

$$s.t.$$

$$T_i = \max\{t | x_{i,t}^r > 0,\ \forall r \in \mathcal{R}, \forall t \in \mathcal{T}\}, \forall i \in \mathcal{N} \tag{2}$$

$$\sum_{i \in \mathcal{N}} x_{i,t}^r \leq C^r, \qquad \forall t \in \mathcal{T}, \forall r \in \mathcal{R} \tag{3}$$

$$\sum_{t \in \mathcal{T}} x_{i,t}^r = R_i^r, \qquad \forall i \in \mathcal{N}, \forall r \in \mathcal{R} \tag{4}$$

$$x_{i,t}^r \in \mathbb{Z}^+, \qquad \forall i \in \mathcal{N}, \forall r \in \mathcal{R}, \forall t \in \mathcal{T}. \tag{5}$$

[1]In section III, we describe how this slotted-time model can be applied to a system like Hadoop which executes in continuous time.

While constraint (2) defines job completion-times, constraints (3) and (4) capture the cluster capacity and job resource requirements respectively. The CORA problem **is an integer programming problem with non-linear constraint (2)**. In this paper, we leverage the structure of our problem to transform it into an equivalent **linear programming problem (LP)**, to obtain the resource-allocation vector $\mathbf{X}$. Here, we summarize the steps in the transformation:

- Step 1: We eliminate the non-linear constraint (2) by reformulating the objective function in (1) to be dependent on the resources allocated, $x_{i,t}^r$, rather than the completion time T_i. We denote this new problem as R-CORA.
- Step 2: We then replace the objective function of R-CORA with a separable convex function [16] that preserves lexicographic max-min fairness and prove that the constraints form a totally unimodular matrix [17] (both these terms are explained in section II-B). This enables us to transform R-CORA into an equivalent LP.
- Step 3: The transformation to LP in the previous step results in $C^r + 1$ additional decision variables for each variable $x_{i,t}^r$ in R-CORA. We transform this LP into another equivalent LP with just three additional variables for each original variable in R-CORA. This reduction in variables is crucial since C^r, the resource capacity, could be in the order of tens of thousands.

We now elaborate each of these steps in the following subsections. Due to space constraints, all the proofs in the paper are presented in the appendix of our technical report [18].

A. Eliminate Non-linearity in the Constraints (Step 1)

We eliminate the non-linear constraint (2) in CORA by representing the job's finishing time using an integer indicator function $I : \mathbb{Z}^+ \rightarrow \{0, 1\}$ so that $I(x_{i,t}^r) = 1$ when $x_{i,t}^r$ is a positive integer and $I(x_{i,t}^r) = 0$ otherwise. Let u_{it} denote the utility value of job i if it finishes at time t (i.e., $u_{it} = U_i(t)$). Since $U_i(t)$ is non-increasing, we have $u_{it} \leq u_{it'}$ if $t \geq t'$. Therefore, by selecting the minimum term of $u_{it}I(x_{i,t}^r)$ for all $t \in \mathcal{T}$ and $r \in \mathcal{R}$, we obtain the utility value of job i at its completion-time. Since scaling the utility function does not change the optimal solution, when u_{it} is rational, we multiply it by a constant coefficient to convert it into an integer with negligible quantization error. This representation links the utility of jobs to their resource-allocation and eliminates the nonlinear definition of job completion-time from the constraints. With the above notations, we transform the CORA problem as follows:

$$\text{(R-CORA)}$$
$$\underset{\mathbf{X}}{\text{lexmax}} \quad \min_{i,r,t}(u_{it}I(x_{i,t}^r))$$
$$s.t.$$
$$\text{Constraints (3), (4) and (5).}$$

B. Transform to Linear Programming Problem (Step 2)

To transform R-CORA into an LP we exploit two fundamental concepts: a separable convex objective function and the total unimodularity of the constraints.

1) Separable Convex Objective Function: A function is referred to as separable convex if it can be represented as a summation of multiple single variable convex functions. Our first goal is to replace the objective function of R-CORA with a separable convex function that preserves the lexicographic max-min fairness. To mathematically define lexicographic order, we introduce the following definitions. Let $\mathbf{Y}$ and $\mathbf{W}$ be two arbitrary k dimensional vectors in $\mathbb{Z}^k$. We define $\overrightarrow{\mathbf{Y}}$ and $\overrightarrow{\mathbf{W}}$ to be the vectors of $\mathbf{Y}$ and $\mathbf{W}$ sorted in nondecreasing order. $\mathbf{Y}$ is said to be lexicographically greater than $\mathbf{W}$, denoted by $\mathbf{Y} \succ \mathbf{W}$, if the first non-zero component of $\overrightarrow{\mathbf{Y}} - \overrightarrow{\mathbf{W}}$ is positive. Consequently, an allocation vector $\mathbf{Y}$ is said to be lexicographically no less than $\mathbf{W}$, denoted by $\mathbf{Y} \succeq \mathbf{W}$, if $\overrightarrow{\mathbf{Y}} - \overrightarrow{\mathbf{W}} = 0$ or $\mathbf{Y} \succ \mathbf{W}$. Consider a convex function $h : \mathbb{Z}^k \rightarrow \mathbb{R}$ which has the form of $h(\mathbf{Y}) = \sum_{j=1}^{k} k^{y_i}$, where y_i is the i-th element of vector $\mathbf{Y}$. The following lemma shows that $h(\mathbf{Y})$ preserves the lexicographical order of the integer vectors in $\mathbb{Z}^k$.

Lemma 1. *For* $\mathbf{Y}, \mathbf{W} \in \mathbb{Z}^k$, $\mathbf{Y} \succeq \mathbf{W} \Longleftrightarrow h(\mathbf{Y}) \geq h(\mathbf{W})$.

We use lemma 1 to translate the lexicographic max-min objective function into a separable convex objective function. Let $k = |\mathcal{N}||\mathcal{T}||\mathcal{R}|$. Note that there are k number of $u_{it}I(x_{i,t}^r)$ terms in the lexicographic max-min objective function of R-CORA. Let $\mathbf{F_X}$ be the vector of $u_{it}I(x_{i,t}^r)$ terms calculated using $\mathbf{X}$. The objective function in R-CORA is equivalent to $lexmin_{\mathbf{X}}(\max(-\mathbf{F_X}))$. Hence, the objective function in R-CORA is searching for the smallest $-\mathbf{F_X}$ vector according to the lexicographic order $\succeq$. From Lemma 1, we preserve the same lexicographic order by comparing the value of $h(-\mathbf{F_X})$. Hence, the objective function of R-CORA is replaced by

$$\min h(-\mathbf{F_X}) = \sum_{i \in \mathcal{N}} \sum_{t \in \mathcal{T}} \sum_{r \in \mathcal{R}} k^{-u_{it}I(x_{i,t}^r)} \tag{6}$$

Since each term of the summation in (6) consists of an exponential function (which is convex) of a single decision variable $x_{i,t}^r$, the function is a separable convex function.

2) Total Unimodularity: An $l \times n$ matrix $\boldsymbol{A}$ is totally unimodular if and only if $\boldsymbol{A}$ has all its entries selected in $\{-1, 0, 1\}$ and every subset of the row indexes (i.e., $I \subseteq \{1, 2, ..., l\}$) can be divided into two disjoint sets, I_1 and I_2, such that $|\sum_{i \in I_1} a_{ij} - \sum_{i \in I_2} a_{ij}| \leq 1, \forall j \in \{1, 2, ..., n\}$ [17]. We show that the constraints of R-CORA are totally unimodular.

Lemma 2. *The coefficients of constraints (3) and (4) form a totally unimodular matrix.*

The total unimodularity property of linear constraints defines a feasible solution polyhedron which has integral extreme points. When an integer programming problem has a separable convex objective function and totally unimodular constraints, there exists a method to transform the integer programming problem into a linear programming problem which has the same optimal solutions [16]. We describe this transformation in the next section.

3) Equivalent LP: To locate the equivalent LP of R-CORA, we use the λ-representation technique [16] in which, for a single integer variable $x \in [0, U] \cap \mathbb{Z}$, an integer convex function $f : [0, U] \cap \mathbb{Z} \rightarrow \mathbb{R}$ can be linearized using the following representation:

(λ-representation)

$$f(x) = \sum_{j \in \mathbf{D}} f(j)\lambda_j \tag{7}$$

$$\sum_{j \in \mathbf{D}} j\lambda_j = x \tag{8}$$

$$\sum_{j \in \mathbf{D}} \lambda_j = 1 \tag{9}$$

$$\lambda_j \in \mathbb{R}^+, \forall j \in \mathbf{D}. \tag{10}$$

In the above λ-representation, $\mathbf{D}$ is the integer set of all the potential values of x. Constraints (8) and (9) define a convex combination set for x using linear variables λ_j. We apply this λ-representation to each convex function $k^{-u_{it} I(x^r_{i,t})}$ in (6). For each variable $x^r_{i,t}$ in the range of $[0, C^r]$, we define a set $\mathbf{D}^r_{i,t} = [0, C^r] \cap \mathbb{Z}^+$ to contain the potential values of $x^r_{i,t}$. According to the λ-representation, extra variables $\lambda^{r,j}_{i,t}$ are introduced for every $j \in \mathbf{D}^r_{i,t}$. Note that the single variable function is sampled at each point $j \in \mathbf{D}^r_{i,t}$, when $j = 0$, the sample value of $k^{-u_{it} I(x^r_{i,t})}$ at $x^r_{i,t} = j$ is simply one. When $j > 0$, the sample value of $k^{-u_{it} I(x^r_{i,t})}$ at $x^r_{i,t} = j$ is $k^{-u_{it}}$. As a result, the R-CORA can be rewritten as the following mixed integer LP:

(Linearized-CORA)

$$\min_{\mathbf{X}, \boldsymbol{\lambda}} \sum_{i \in \mathcal{N}} \sum_{t \in \mathcal{T}} \sum_{r \in \mathcal{R}} \Big(\sum_{j \in \mathbf{D}^r_{i,t} \setminus \{0\}} k^{-u_{it}} \lambda^{r,j}_{i,t} + \lambda^{r,0}_{i,t} \Big)$$

$s.t.$

$$\sum_{j \in \mathbf{D}^r_{i,t}} j\lambda^{r,j}_{i,t} = x^r_{i,t}, \qquad \forall i \in \mathcal{N}, \forall t \in \mathcal{T}, \forall r \in \mathcal{R}$$

$$\sum_{j \in \mathbf{D}^r_{i,t}} \lambda^{r,j}_{i,t} = 1, \qquad \forall i \in \mathcal{N}, \forall t \in \mathcal{T}, \forall r \in \mathcal{R}$$

$$\lambda^{r,j}_{i,t} \in \mathbb{R}^+, \forall j \in \mathbf{D}^r_{i,t}, \qquad \forall i \in \mathcal{N}, \forall t \in \mathcal{T}, \forall r \in \mathcal{R}$$

$$\mathbf{D}^r_{i,t} = [0, C^r] \cap \mathbb{Z}^+, \qquad \forall i \in \mathcal{N}, \forall t \in \mathcal{T}, \forall r \in \mathcal{R}$$

Constraints (3), (4) and (5).

The above problem can be solved by linear programming through relaxing the integer constraints [16]. The total unimodularity property guarantees that the optimal $x^r_{i,t}$ values are integral. Hence, the integral optimality is preserved. However, from equation (10), it is clear that this linearization introduces $|\mathbf{D}^r_{i,t}| = C^r + 1$ additional decision variables for each original variable $x^r_{i,t}$. In the following section, we reduce the size of $\mathbf{D}^r_{i,t}$ to improve the speed of the Linearized-CORA problem without any loss in optimality.

C. Reduce the Number of Decision Variables (Step 3)

The λ-representation represents the value of a single variable convex function $f(x)$ using a convex combination of function values sampled at the integer points in the set of $\mathbf{D}$. If $f(x)$ is a **nonlinear** single variable convex function $f(x)$, then for integers a, b, c such that $a, c \in \mathbf{D}$, $b \notin \mathbf{D}$ and $b \in [a, c]$, the λ-representation represents the value of b using convex combination $b = \lambda_a a + \lambda_c c$ with $\lambda_a + \lambda_c = 1$. Because of the convexity of the nonlinear function $f(x)$, we have $f(b) < \lambda_a f(a) + \lambda_c f(c)$. As a result, when $b \notin \mathbf{D}$, the λ-representation overestimates the value of $f(x)$ at b. Hence, for nonlinear convex f, $\mathbf{D}$ must include all the possible values of x. On the other hand, if $f(x)$ is **linear** in the range of $[a, c]$ then, $f(b) = \lambda_a f(a) + \lambda_c f(c)$ where $b = \lambda_a a + \lambda_c c$ with $\lambda_a + \lambda_c = 1$. In this case, we can remove all the integers in the range of (a, c) from $\mathbf{D}$ without creating function value overestimation. To mathematically define the equivalence of the reduced $\mathbf{D}$ set, we present the following lemma.

Lemma 3. *For an optimization problem that optimizes a separable convex function over an integral lattice, replacing $\mathbf{D}$ with $\mathbf{D}' \subset \mathbf{D}$ for each single variable convex function $f(x)$ in the λ-representation preserves integral optimality if $\forall b \in \mathbf{D} \setminus \mathbf{D}'$, $\exists a, c \in \mathbf{D}'$ such that $b \in [a, c]$ and $f(x)$ is linear in $[a, b]$.*

Due to the indicator function, the single variable convex function $k^{-u_{it} I(x^r_{i,t})}$ in (6) is linear when $x^r_{i,t} \in [1, C^r]$. Therefore, all the integer value points in $(1, C^r)$ can be removed from $\mathbf{D}^r_{i,t}$. As a result, for each $x^r_{i,t}$ variable, only three variables denoted $\lambda^{r,0}_{i,t}$, $\lambda^{r,1}_{i,t}$ and $\lambda^{r,c}_{i,t}$ corresponding to the integer sampling points, 0, 1 and C^r, should be considered in the λ-representation. In this way, the Linearized-CORA problem can be greatly simplified as follows:

(FINAL)

$$\min_{\mathbf{X}, \boldsymbol{\lambda}} \sum_{i \in \mathcal{N}} \sum_{t \in \mathcal{T}} \sum_{r \in \mathcal{R}} (\lambda^{r,0}_{i,t} + k^{-u_{it}} (\lambda^{r,1}_{i,t} + \lambda^{r,c}_{i,t}))$$

$s.t.$

$$\lambda^{r,1}_{i,t} + C^r \lambda^{r,c}_{i,t} = x^r_{i,t}, \qquad \forall i \in \mathcal{N}, \forall t \in \mathcal{T}, \forall r \in \mathcal{R}$$

$$\lambda^{r,0}_{i,t} + \lambda^{r,1}_{i,t} + \lambda^{r,c}_{i,t} = 1, \qquad \forall i \in \mathcal{N}, \forall t \in \mathcal{T}, \forall r \in \mathcal{R}$$

$$\lambda^{r,0}_{i,t}, \lambda^{r,1}_{i,t}, \lambda^{r,c}_{i,t} \in \mathbb{R}^+, \qquad \forall i \in \mathcal{N}, \forall t \in \mathcal{T}, \forall r \in \mathcal{R}$$

Constraints (3), (4) and (5).

The FINAL problem is an LP with $4|\mathcal{N}||\mathcal{T}||\mathcal{R}|$ decision variables as compared to the $|\mathcal{N}||\mathcal{T}||\mathcal{R}|(C^r + 2)$ variables of Linearized-CORA. Since, the resource capacity C^r can be in the order of tens of thousands, this is a significant reduction. The following theorem completes the transformation of CORA into the LP presented as the FINAL problem.

Theorem 1. *An optimal solution to the FINAL problem is an optimal solution to the CORA problem.*

Efficient LP solvers (e.g., Lindo [19], Mosek [20]) can be applied to solve the FINAL problem. For convenience, we refer to the scheduler based on a solver for the FINAL problem as the CORA scheduler.

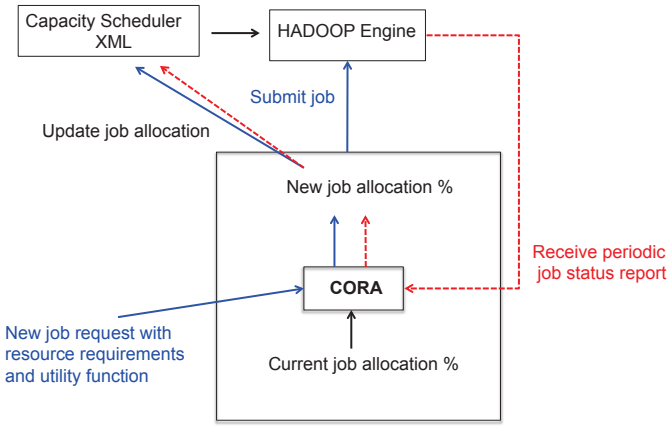

Fig. 2. CORA-Hadoop and its signaling cycle.

III. CORA Scheduler for Hadoop: Implementation and Evaluation

Shifting from theory to systems, we now describe the implementation of our CORA scheduler as a new job scheduler for Hadoop. Further, we compare the CORA scheduler with other commonly used schedulers through extensive experiments performed with a heterogeneous mix of MapReduce jobs using real-world data sets. Results confirm that the CORA scheduler not only optimizes the max-min utility across the jobs in Hadoop but also achieves strong results in terms of sum utility and in terms of adherence to target completion times.

A. Design and Implementation

Hadoop is the de facto standard for large scale data processing applications modeled as MapReduce jobs. Each job is split into Map and Reduce tasks, where the Map tasks perform the actual computation on ⟨key, value⟩ pairs of data and the Reduce tasks aggregate the results according to the keys. An important problem, referred to as scheduling in Hadoop, concerns the division of cluster resources among the jobs. We implement our CORA scheduler as a utility-aware scheduler in Hadoop 2.4.0, where cluster resources are partitioned into multiple units called containers, where each container, is assigned enough CPU, memory and I/O bandwidth to process a Map or Reduce task. Through our scheduler, we control the number of containers assigned to each Hadoop job. Figure 2 illustrates the functioning of the CORA scheduler with Hadoop. The CORA scheduler periodically checks the status of running jobs in the cluster and maintains the exact number of Map and Reduce tasks remaining for each job. Our solver for the FINAL problem is triggered on precisely three events: the addition of a new job, the completion of a Map or Reduce task in the system and the completion of an existing job. At every scheduling event, CORA generates the exact percentage of cluster containers to be allocated to each active job and specifies this to the Hadoop capacity scheduler [21]. This scheduler in turn ensures that Hadoop allocates the specified amount of containers to each job.

A significant challenge in this implementation is in generating the input parameters to the FINAL problem at each scheduling event. Since Hadoop only allows control over the number of containers, in this framework we can only consider a single resource type that effectively bundles multiple resources. The cluster capacity value C (we drop the r from C^r and R_i^r since there is just one resource), is the total number of containers in the cluster. This is a standard parameter configured by the Hadoop administrator, chosen such that simultaneous Map and Reduce tasks do not compete for resources. In Hadoop 2.4.0 this is calculated as the ratio between the amount of total memory available in the cluster and the desired memory size of each container.

The job resource requirement R_i for each job i, is the total number of Map and Reduce tasks it requires to complete. This is readily available from the Hadoop system since these numbers are predetermined, based on the total amount of data submitted with the job and the Hadoop data splitting scheme. However, the Map and Reduce tasks of different job types may take different times to execute. In the technical report [18], we describe how we scale the resource requirement accordingly. The next parameter required for the FINAL problem is the number of discrete time slots, $|\mathcal{T}|$. We choose the number of slots to be large enough for our FINAL problem to generate a feasible solution. For example, if a cluster has $C = 40$ containers and the total resource requirement across all the current jobs is 160 tasks (both Map and reduce), then we choose $|\mathcal{T}| = 4$. Finally, the job utility function $U_i(t)$ is specified by each job following the sigmoid function, which contains both a convex and a concave part:

$$U_i(t) = \frac{p_i}{1 + e^{\mu_i(t-d_i)}} \tag{11}$$

In the above utility function, d_i and p_i denote the target completion-time and the priority of job i respectively. The constant value μ_i, referred to as job i's **decay factor**, controls how quickly the jobs' utility values decay, thereby quantifying its sensitivity to completion times. Note that the client specifies d_i in terms of absolute time. In the technical report, we describe how we convert this to the slotted time model. We choose the sigmoid utility function in our experiments since it models a wide range of job utilities. However, our scheduler is versatile enough to handle a large family of non-increasing job utility functions in similar fashion (we also present results for linear utility functions).

B. Other Schedulers for Comparison

We compare our scheduler with four mainstream schedulers: (i) **FIFO** scheduling, in which jobs are scheduled according to the order of their arrival time. This is the default scheduler of Hadoop. (ii) **Proportional** scheduling, in which each job is allocated the share of cluster resources proportional to its priority. This reflects a comparison with the **fair scheduler** [10], one of the most popular Hadoop schedulers. (iii) **Earliest-deadline-first** (EDF) scheduling, in which jobs are scheduled according to the order of their target completion-times. (iv) **Risk-reward-heuristic** (RRH) scheduling [14], in

which, scheduling decisions are made based on the future utility gain and opportunity cost of reallocating resources. Specifically, the opportunity cost is calculated using the decay rate (equivalent to the slope of the job utility function at the target completion time) of the job utility function.

The comparisons among these four schemes serve to illustrate that a scheduling strategy oblivious to the jobs' sensitivity to completion-times or resource constraints is prone to making wrong scheduling decisions, which lead to low utility value. Further, even a scheme such as RRH, that specifically considers heterogeneous time-dependent job utilities, cannot achieve optimal performance because it only takes into account the tangent to the job utility function locally at the target completion time, rather than the shape of the entire utility function.

C. Experimental Set-up

The experiments are carried out on a Hadoop cluster that consists of 5 HP Proliant DL360 G6 servers with a total of 40 CPU cores and 100 GB of RAM. The linear programming problem FINAL is solved using the simplex method implemented in the Mosek optimization library [20]. To emulate a real-world Hadoop cluster as closely as possible, we choose an equal mix of eight heterogeneous Hadoop job templates (Movie Classification, Histogram of Movies, Histogram of Ratings, InvertedIndex, SelfJoin, SequenceCount, WordCount and Terabyte Data Sorting) and multiple real-world data sets from the PUMA benchmark suite [15]. Each job processes at least 10 GB of data and each experimental run process up to 1 TB of data. To broadly capture jobs' sensitivity to completion-time, we consider three job classes based on the decay factor μ_i:

- **completion-time-critical (CT-critical)** jobs with decay factor between 4 and 6, in which the utility drops rapidly if the jobs' actual completion-time exceeds the target completion-time.
- **completion-time-sensitive (CT-sensitive)** jobs with decay factor between 0.01 and 1, in which the utility decreases smoothly over a period of time.
- **completion-time-insensitive (CT-insensitive)** jobs with decay factor of 0.

These jobs represent a wide range of MapReduce applications with different characteristics. The generated Hadoop jobs are submitted to the cluster dynamically according to a Poisson arrival process with a mean inter-arrival time of 130 seconds.

Cluster capacity C is configured to have a value of 40 task slots. The jobs in our system required up to 100 Map tasks and 20 reduce tasks. We chose job utilities based on the functions described in (11). Priority p_i is generated randomly according to a uniform distribution with a range of $[1, 5]$. For each job template, we benchmark the total running time of the jobs. The running time is then used to generate the target completion times, d_i, by multiplying it with a random factor generated from a Gaussian distribution with mean 1.25 and a standard deviation of 0.1. Note that such target completion times are possible only if each job can consume all the resources in the cluster. Because of the resource competition among multiple active jobs, without scaling up the d_i values at runtime, no scheduler can finish the jobs before their target completion times. Therefore, at the job's submission time, d_i is multiplied by the number of active jobs in the cluster.

D. Performance Results

1) Scheduling Accuracy: To construct a bridge between the theoretical model and the effectiveness of our CORA scheduler implementation, we first examine the optimality of the FINAL problem using real-world experiments. In the first part of the experiment, we dynamically submit 100 MapReduce jobs to the cluster over a period of 4 hours. Among these jobs, 20% are CT-critical, 60% are CT-sensitive and 20% are CT-insensitive. Resources are allocated to jobs according to the CORA scheduler. During the entire life span of the experiment, there are 1133 scheduling events in which CORA scheduler solves the FINAL problem. In each scheduling event, the utility value of the FINAL problem achieved by CORA scheduler is notably higher than the utility value achieved by the other schedulers. In Figure 3, we plot the distribution of such utility value improvements recorded for all 1133 scheduling events. It shows that the implemented CORA scheduler is able to accurately construct and solve the FINAL problem.

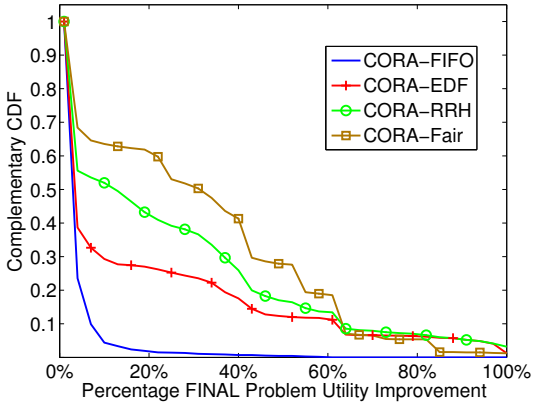

Fig. 3. CCDF of utility improvement achieved by CORA over other schedulers. We calculate utility improvement as the difference between the utility achieved in the FINAL problem and the minimum utility achieved by every other scheduler. For example, the first marker on the CORA-FAIR curve indicates that CORA obtains at least 18% utility improvement over FAIR for 0.65 fraction of all scheduling events.

Next, we focus on the actual utility value of each job at the end of the experiment. We run the same experiment separately for each of the five schedulers and compare their end-to-end performance. Table I shows the minimum value and the sum value of the actual job utility achieved for CT-critical and CT-sensitive jobs (the utility of CT-insensitive jobs does not vary across schedulers). If a single CT-critical job does not finish in time, it causes the minimum utility to become zero. CORA protects such jobs and hence achieves a utility greater than

TABLE I
MIN UTILITY, SUM UTILITY AND SOLVER RUNNING TIME FOR 100 JOBS

	Min	Sum	Running time (ms)
CORA	0.008367	**241.8256**	Mean 347.47 STD 726.29
FIFO	0	164.0450	< 1
Fair	0	120.6410	< 1
RRH	0	167.1594	< 1
EDF	0	147.6562	< 1

zero unlike the others. CORA performs much better in terms of the sum utility since it not only tries to maximize the worst-case utility but also tries to improve the subsequent worst-case utilities. Even though the CORA simplex solver has a higher running time than the other schedulers, on average, it executes in less than 500 milliseconds and is hence efficient.

To compare the performance of various schedulers with respect to completion-times, we define the Δ **value** of a job to be the difference between its actual and target completion-time.

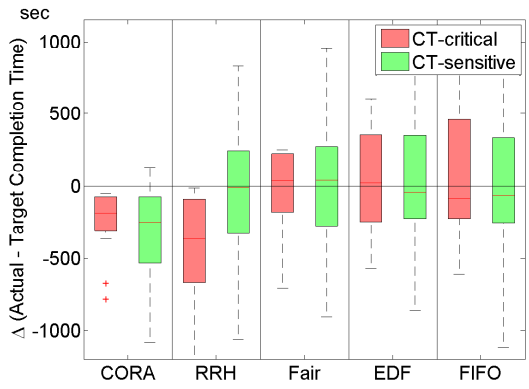

Fig. 4. Δ values for all schedulers across different job classes (100 Jobs). CORA outperforms Fair, EDF and FIFO for both CT-critical and CT-sensitive jobs. RRH overprotects CT-critical jobs at the cost of CT-sensitive jobs, leading to poor utility performance.

For the same 100 job experiment described above, Figure 4 shows the boxplot statistics of the jobs' Δ values for different CT-classes. The figure demonstrates that our proposed CORA scheduler is the only scheduler that completes all CT-critical jobs and the majority of the CT-sensitive jobs before their target completion times. The results clearly show that FIFO and Fair schedulers are oblivious to target completion times and incur high Δ values for CT-critical jobs. Since EDF only cares about the value of the target completion-time and ignores resource demands or completion-time-sensitivity, it leads to high Δ values as well. The RRH scheduler achieves low Δ values for CT-critical jobs, but in doing so, it sacrifices the performance of CT-sensitive jobs. CORA on the other hand achieves good balance of Δ values for both these classes, which eventually results in better utility performance. To understand this better, we investigate the effect of balancing CT-critical and CT-sensitive jobs, on the performance of CORA and RRH which consider the sensitivities of the jobs' completion-time.

2) Effect of CT-Critical/Sensitive Jobs: In this experiment, 20 heterogeneous jobs are submitted to the cluster dynamically. We repeat the experiment for five iterations. In each iteration we vary the proportion of CT-critical and CT-sensitive jobs while maintaining 25% of the jobs as CT-insensitive. Since EDF, FIFO and Fair schedulers do not account for sensitivity to job completion times, we exclude these three schedulers from the experiments. Figure 5 indicates that even as the number of CT-critical jobs increase and hence scheduling becomes harder, CORA still completes almost all the CT-critical and CT-sensitive jobs, by sacrificing performance of the CT-insensitive jobs (not shown in the figure to simplify presentation). The RRH scheduler performs well for CT-critical jobs but fails to protect the performance of the CT-sensitive jobs. This is because CT-critical jobs have a steeper decay slope which results in a larger reward value for RRH. Table II shows the minimum and sum of job utilities achieved by the CT-critical and CT-sensitive jobs across all the data points of the experiment. The superior performance of CORA indicates that a singular preference towards CT-critical jobs (at the cost of CT-sensitive jobs) is undesirable for sigmoid utility functions. For sigmoid utility functions, CORA attempts to finish jobs by their target completion times without caring about how early it finishes them. In the next section, we illustrate how the behavior of CORA can be changed as desired by changing the class of utility functions.

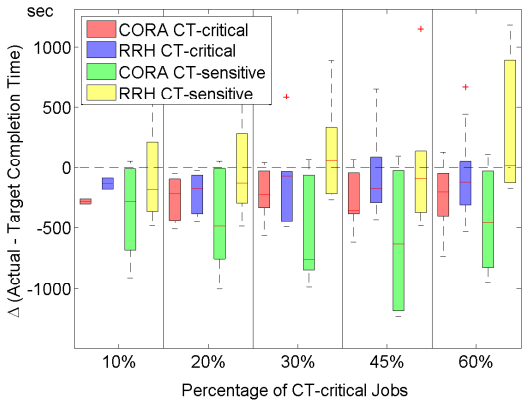

Fig. 5. Δ values for CORA/RRH varying % of critical/sensitive jobs (sigmoid utility). CORA achieves good performance for both CT-critical and CT-sensitive jobs, unlike RRH that overly focuses on CT-critical jobs.

3) Robustness to Change in Utility Curvature: In this experiment, we study the schedulers' performance for truncated linear utility functions where the utility decreases linearly with the job's execution time. When the job's total execution time exceeds its target completion-time, the utility value drops to zero. The slope of the utility function is randomly generated from three classes: **steep-slope** jobs with a slope in [-1, -0.7], **gradual-slope** jobs with a slope in [-0.4, -0.1] and no-slope

TABLE II
MIN AND SUM UTILITY FOR VARYING % OF CRITICAL/SENSITIVE JOBS

	CORA		RRH	
	Min	Agg.	Min	Agg.
10%	0.1729	46.5574	0	36.0844
20%	0.3610	45.9568	0	37.5157
30%	0	46.9446	0	32.3376
45%	0	45.6386	0	37.7590
60%	0	44.0081	0	35.5677

jobs. Linear utility functions encourage jobs to be completed as soon as possible. For these classes of utility functions, we perform experiments similar to those in the previous section and vary the % of steep-slope and gradual-slope jobs while keeping the % of no-slope jobs constant. Figure 6 shows the median, 25% and 75% quartiles of the jobs' Δ values achieved by CORA and RRH for the slope-sensitive jobs. On average CORA finishes jobs earlier than RRH and as indicated by the much lower 25% quartile, for many jobs it finishes considerably earlier. Table III shows that CORA outperforms RRH in terms of the sum utility value. The results of sections III-D2 and III-D3 demonstrate that when the utility function encourages jobs with steeper slops/higher decay values to finish earlier, CORA outperforms RRH. The results also show CORA's versatility in tacking a wide range of utility functions with different physical interpretations.

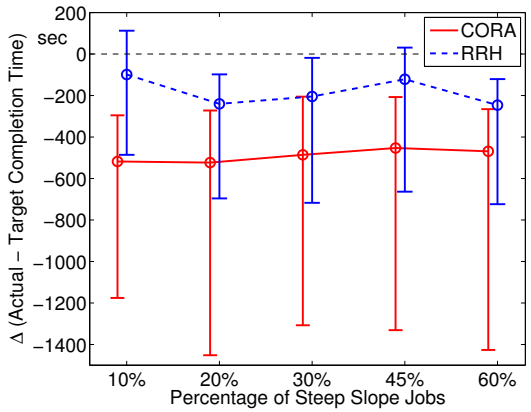

Fig. 6. Δ values for CORA/RRH for varying % of steep/gradual-slope jobs. CORA outperforms RRH and adapts to the fact that linear utility functions actively reward earlier completion times (reflected in the low 25% quartiles).

TABLE III
MIN AND SUM UTILITY FOR VARYING % OF STEEP/GRADUAL-SLOPE JOBS

	CORA		RRH	
	Min	Agg.	Min	Agg.
10%	0.5	1864.1	0.5	884.5
20%	0.5	2461.4	0.5	1411.0
30%	0.5	3181.6	0.5	1630.5
45%	0.5	3881.0	0.5	2291.4
60%	0.5	6380.1	0.5	3651.1

IV. RELATED WORK

There has been prior work in real-time systems, grid computing and high performance computing in the area of utility-driven resource-allocation, with the utility function directly dependent on job completion-times [14], [22]–[25]. In [22], the authors present heuristics to handle task scheduling across heterogeneous machines when the tasks have utility functions that depend on task priority and completion-times. In [1]–[3], similar techniques are applied to scheduling in data centers, where the focus is on hard-deadline jobs and soft-deadline jobs. While the high level goal of this paper is similar, note that we make no assumptions about the client utility functions apart from the fact it should be non-decreasing. Further, we focus on the optimality of our solution as opposed to the heuristics presented in these prior works.

There has been previous work on ensuring fairness in resource-allocation. The authors in [26] present a method to incorporate data locality to allocate an equal share of resources to all jobs in the cluster. Proportional fairness is achieved in [5] in the context that all jobs are allocated equal resources independent of target completion-times. However, all of these previous works consider fairness in terms of resource-allocation, rather than fairness in terms of utilities or completion-times. In many cases, it may be necessary to allocate a disproportionate share of resources to a job that is completion-time-critical.

Deadline-aware scheduling has been investigated for Hadoop-based data processing [4]–[7]. The authors in [5] present a Hadoop scheduler to solve the conflict between fair scheduling of jobs and the latency inherent in obtaining non-local data. So when a job cannot be launched to due to unavailability of data, is it delayed to accommodate other jobs. This approach speeds up job processing, but does not optimize resource-allocation to meet specific requirements in terms of completion-times. In [7], the authors present a technique for job scheduling in MapReduce clusters that incorporates two types of jobs: high priority production jobs and lower priority research jobs. Further they discuss job eviction policies to accommodate such a mixture of jobs. While they do consider different types of jobs and their target completion-times, they do not consider the completion-time-sensitivity of jobs to perform resource-allocation.

In [27] the authors propose a job execution model and design a scheduler that accounts for various parameters that affect job completion-times. However, this does not incorporate any optimization for resource-allocation and jobs are only scheduled if specified target completion-times can be met. In [28], the authors present a Nash Bargaining problem formulation for completion-time-aware resource-allocation. However, they define job utilities in terms of the resources allocated by the target completion-time. In this paper, we model utilities as a function of completion-time. Such utility functions is much more expressive and relevant in terms of client service requirements.

V. Conclusion and Future Work

Client specifications on completion-times is an important way to measure the utility clients obtain from the cloud. Due to the nature of heterogeneity in the cloud, it is a hard problem to allocate resources to optimize such utilities. Popular scheduling schemes such as fair and FIFO scheduling fail to achieve good performance when job utilities are measured based on completion times. Further, utility-based heuristics for resource allocation, which do not perform optimal allocation, are insufficient to address this problem. This is primarily because in a resource constrained set-up, even a few wrong decisions can lead to poor performance. In this paper, we take a significant step towards addressing this problem and present a resource allocation scheme (CORA) that optimizes specifically for completion-times. To enable widespread usage of our technique, we implement a utility-aware scheduler for the Hadoop data processing framework and using real-world data sets confirm the efficacy of our scheduler.

There are several avenues for future work. While we exploit job heterogeneity in this paper, we wish to consider metrics to quantify this heterogeneity and evaluate the utility achieved as a function of heterogeneity. Currently, we do not consider cluster-node-failures explicitly in our model and assume that the data processing framework handles it internally. An explicit focus on resource-allocation in the presence of failures is an important avenue of research. The approach introduced in this paper may be extended beyond resource-allocation in the cloud. We wish to apply the proposed problem formulation to other areas such as packet/flow scheduling and wireless networking with a specific focus on completion-times.

References

[1] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better Never Than Late: Meeting Deadlines in Datacenter Networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, (New York, NY, USA), pp. 50–61, ACM, 2011.

[2] M. N. Bennani and D. Menascé, "Resource Allocation for Autonomic Data Centers using Analytic Performance Models," in *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, (Washington, DC, USA), pp. 229–240, IEEE Computer Society, 2005.

[3] A. Chandra, W. Gong, and P. Shenoy, "Dynamic Resource Allocation for Shared Data Centers Using Online Measurements," in *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '03, (New York, NY, USA), pp. 300–301, ACM, 2003.

[4] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, (Berkeley, CA, USA), pp. 29–42, USENIX Association, 2008.

[5] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proceedings of the 5th European conference on Computer systems*, pp. 265–278, ACM, 2010.

[6] B. Rao and D. L.S.S.Reddy, "Article: Survey on improved scheduling in hadoop mapreduce in cloud environments," *International Journal of Computer Applications*, vol. 34, pp. 29–33, November 2011. Published by Foundation of Computer Science, New York, USA.

[7] B. Cho, M. Rahman, T. Chajed, I. Gupta, C. Abad, N. Roberts, and P. Lin, "Natjam: Design and evaluation of eviction policies for supporting priorities and deadlines in mapreduce clusters," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, (New York, NY, USA), pp. 6:1–6:17, ACM, 2013.

[8] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st ed., 2009.

[9] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, (New York, NY, USA), pp. 59–72, ACM, 2007.

[10] "Hadoop Fair Scheduler." https://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html.

[11] S. Shakkottai and R. Srikant, "Scheduling real-time traffic with deadlines over a wireless channel," in *Proceedings of the 2nd ACM international workshop on Wireless mobile multimedia*, WOWMOM '99, (New York, NY, USA), pp. 35–42, ACM, 1999.

[12] K. Johnsson and D. Cox, "An adaptive cross-layer scheduler for improved qos support of multiclass data services on wireless systems," *Selected Areas in Communications, IEEE Journal on*, vol. 23, no. 2, pp. 334–343, 2005.

[13] "CORA Hadoop Scheduler." http://cloudfleet.ece.ust.hk/~ecefelix/CORA.

[14] D. E. Irwin, L. E. Grit, and J. S. Chase, "Balancing risk and reward in a market-based task service," in *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, HPDC '04, (Washington, DC, USA), pp. 160–169, IEEE Computer Society, 2004.

[15] "Puma: Purdue Mapreduce benchmarks suite." https://sites.google.com/site/farazahmad/pumabenchmarks.

[16] R. Meyer, "A class of nonlinear integer programs solvable by a single linear program," *SIAM Journal on Control and Optimization*, vol. 15, no. 6, pp. 935–946, 1977.

[17] A. Ghouila-Houri, "Caractérisation des matrices totalement unimodulaires," *CR Acad. Sci. Paris*, vol. 254, pp. 1192–1194, 1962.

[18] Z. Huang, B. Balasubramanian, M. Wang, T. Lan, M. Chiang, and D. H. K. Tsang, "Need for speed: Cora scheduler for optimizing completion-times in the cloud," technical report, 2014. Available as http://cloudfleet.ece.ust.hk/~ecefelix/CORA/techReport.pdf.

[19] L. Schrage, *Linear, integer, and quadratic programming with LINDO: user's manual*. Scientific Press, 1986.

[20] E. Andersen and K. Andersen, "The mosek interior point optimizer for linear programming: An implementation of the homogeneous algorithm," in *High Performance Optimization* (H. Frenk, K. Roos, T. Terlaky, and S. Zhang, eds.), vol. 33 of *Applied Optimization*, pp. 197–232, Springer US, 2000.

[21] "Hadoop Capacity Scheduler." https://hadoop.apache.org/docs/r1.2.1/capacity_scheduler.html.

[22] L. D. Briceno, B. Khemka, H. J. Siegel, A. A. Maciejewski, C. Groer, G. Koenig, G. Okonski, and S. Poole, "Time utility functions for modeling and evaluating resource allocations in a heterogeneous computing system," *2013 IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum*, vol. 0, pp. 7–19, 2011.

[23] B. Ravindran, E. D. Jensen, and P. Li, "On recent advances in time/utility function real-time scheduling and resource management," *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, vol. 0, pp. 55–60, 2005.

[24] F. I. Popovici and J. Wilkes, "Profitable services in an uncertain world," in *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, SC '05, (Washington, DC, USA), pp. 36–, 2005.

[25] A. Auyoung, L. Grit, J. Wiener, and J. Wilkes, "Service contracts and aggregate utility functions," in *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing*, pp. 119–131, 2006.

[26] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 261–276, ACM, 2009.

[27] S. Wagner, E. V. D. Berg, J. Giacopelli, A. Ghetie, J. Burns, M. Tauil, S. Sen, M. Wang, M. Chiang, T. Lan, *et al.*, "Autonomous, collaborative control for resilient cyber defense (accord)," in *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2012 IEEE Sixth International Conference on*, pp. 39–46, IEEE, 2012.

[28] Y. Xiang, B. Balasubramanian, M. Wang, T. Lan, and C. Mung, "Self-adaptive, deadline-aware resource control in cloud computing," technical report, 2013. Available as http://www.seas.gwu.edu/~tlan/papers/deadlineaware.pdf.