RISC: Robust Infrastructure over Shared Computing Resources Through Dynamic Pricing and Incentivization

Tridib Mukherjee*, Partha Dutta*, Vinay G. Hegde† and Sujit Gujar‡
*Xerox Research Center India (XRCI), Bangalore, India
†International Institute of Information Technology (IIIT), Bangalore, India
‡Artificial Intelligence Laboratory, EPFL, Lausanne, Switzerland

***Abstract*—This paper presents a framework for Robust Infrastructure over Shared Computing resource (RISC), which can offer Organizations with Small-scale Computing infrastructures (OSCs) a way to share their unused resources in an ad-hoc manner for suitable monetary incentives. Such a framework provides dual benefits to an OSC: it enables sharing of unused resource during periods of low computing load while allowing execution of any long-term computation on public or anonymized data at a very low cost during periods of high load. The ad-hoc and heterogeneous nature of the shared infrastructure make the resource management problem in RISC non-trivial—a resource manager needs to: (i) maximize profit while determining incentives for resource owners and prices for resource users in an integrated manner; and (ii) emulate large-scale cloud-like robustness and capabilities out of unreliable, small-scale and intermittently available resources at a low cost. This leads to a constrained market situation where offered prices and incentives should lead to a desired level of SLA and reliability for the consumers.**

Existing approaches of incentive based scheduling for market-like grids assume an open market, based only on demand response; and thus are inapplicable for the constrained market situation in shared resources infrastructure. Specifically, RISC framework has two main components: (i) a *first-of-a-kind* Dynamic Pricing and Incentivization (DPI) strategy that computes the incentives and the prices while maximizing profit for RISC, using an epoch-by-epoch pricing feedback loop; and (ii) a DPI dependent Reliability, Cost and SLA-aware (RCS) scheduler that takes the resource reservation requests as input and assigns replicas of these requests to one or more shared resources for guaranteeing performance SLAs and reliability, while minimizing the cost of resource reservations. Moreover, to handle the communication overhead of computing over geographically distributed resources, the scheduler strives to reduce the network cost of resource allocation. Results from extensive trace-driven experimentation show that our approach can indeed provide appropriate incentives for resource providers, and robust cost-efficient infrastructure solution for resource users.

***Keywords*-Pricing, Reliability, Incentivization**

I. Introduction

Shared resource management has been an active area of research over the years in grid and volunteer computing. Organizations with Small-scale Computing infrastructure (OSCs), e.g., Small and Medium Businesses (SMBs), can benefit from a marketplace of shared resources by getting access to a low-price large-scale infrastructure. They can further share unused/surplus resources in the marketplace in return of monetary incentives (thus, additionally offsetting the prices).

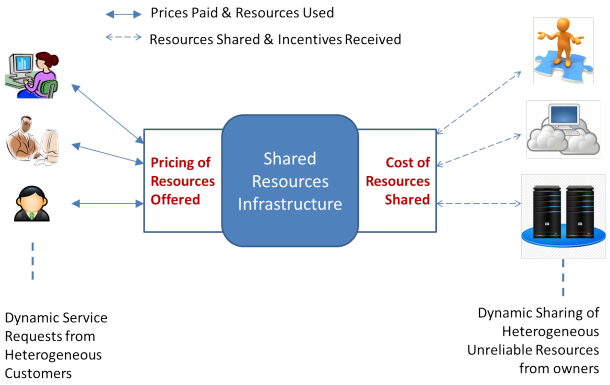


Figure 1. Vision of shared resource infrastructure

Typically, in a shared resource infrastructure (Fig. 1), a consumer/requester would want access to certain computation power to execute their application—usually these applications are computations on public or anonymized data, thus not requiring privacy guarantees. The infrastructure operates as a resource market that offers certain prices for the requests (similar to infrastructure-as-a-service offerings over public cloud) while guaranteeing SLAs and reliability for the requests. However, unlike in public cloud environments, the underlying infrastructure hosting the requests are composed of shared *unreliable* resources. Further, as opposed to dedicated cloud infrastructures, the cost of hosting the requests depend on the incentives offered to the resource owners/providers—leading to a dynamic marketplace of resources and requests. Enabling such a shared resources infrastructure needs to address the following challenges:

- *Constrained Marketplace.* It is imperative to offer prices to consumers/requesters and incentives to providers/owners based on dynamic supply and demand in a way that maximizes profit as well as caters to constraints on: (i) offered prices, which should be less than any comparable cloud offerings; and (ii) robustness, which should address inherent unreliability of resources while minimizing cost. Existing incentive based scheduling for market-like grids [3], [4] assumes an open market, based only on demand response; and thus are inapplicable for shared resources infrastructure.
- *Unreliable Ad-hoc Resources.* The resources can be shared periodically with temporal constraints, e.g. every

night (10 pm–midnight). The periodicity may vary for different OSCs (e.g., based on geographical distribution and time-zones), or even within an OSC for different resources. Further, the resources shared for a particular duration can be retracted by the respective owners. Finally, the accessibility of the resources can be highly unpredictable because of intermittent connection as well as system failures. Awareness of such unreliable and ad-hoc behaviour while determining prices and incentives in a controlled marketplace is unexplored in the literature to the best of our knowledge.

- *Trade-off between monetary and network cost.* The resource cost can be monetary (through incentivization) as well as networking (e.g., bandwidth and latency). There is an underlying trade-off between these costs. Allocating a request to geographically distributed resources with the lowest monetary cost (or incentive), so that the offered prices to the resource users can be lowered, may select geographically distant resources causing significant networking overhead, which in turn can degrade the performance of any job over that allocation.

Our Contributions. This paper proposes RISC, a framework for resource management for Robust Infrastructure over Shared Computing resources, to address all these issues together. In this regard, a Dynamic Pricing and Incentivizing (DPI) algorithm is introduced that determines the incentives of the resources and prices for the requests in an integrated manner while maximizing profit. DPI employs an *epoch-by-epoch* strategy, where, in every epoch, the prices and incentives are fine tuned based on outcomes of previous epoch. Further, in each epoch, based on the monetary cost (i.e. incentives offered) of resources, a Reliability, Cost, and SLA-aware (RCS) scheduler efficiently assigns resource reservation requests (also referred as tasks) on unreliable resources with time-varying capacity, while addressing the trade-off between monetary and network cost of such an assignment through joint optimization. The core of the scheduler is a subtask scheduling algorithm, which provides provably near-optimal performance guarantee for the subtask scheduling problem.

DPI and RSC are both inter-dependent with each other. DPI determines the prices and incentives based on the overall demand/usage of resources as per the allocations of requests by RCS. On the other hand, RCS determines the allocation based on the monetary costs (incentives) determined by DPI. This inherent inter-dependency is addressed by DPI as follows. In each epoch, DPI performs optimizations on the prices and incentives by calling the RCS scheduler and checking the impact of prices and incentives on the overall profit. Accordingly, in the subsequent epoch the prices and incentives are updated. DPI further chooses the prices and incentives from a finite range to ensure the dynamics remain within the market constraints. For example, the maximum price is assumed to be less than the public cloud prices of comparable requests (e.g., small or large instances in Amazon EC2). This ensures that the prices offered to requesters, albeit dynamic based on demand and supply, are always less than the market alternatives for OSCs.

We perform a detailed trace-driven evaluation under a highly dynamic task arrival and resource availability. Results show that DPI leads to profit maximization under different supply-demand combinations. Interestingly, because of the constrained market scenario of shared resources infrastructure, the profit is higher for low demand and large supply compared to high demand and low supply. In our experiments, DPI further ensures that for specific supply and demands, the optimum prices and incentives can be decided in at most two epochs—a significantly fast convergence considering the aforementioned interdependencies. RCS further provides a good balance between the monetary and network cost of resource allocation, and leads to high profit for RISC while meeting the task's SLA and reliability requirement. To summarize, RISC enables a large-scale robust infrastructure over unreliable shared resources in a way that both consumers and providers are interested in the shared resource infrastructure.

The rest of the paper is organized as follows. Section II presents the related work followed by the system model in Section III. Sections IV and V describe the DPI and RCS algorithms, respectively. The experimental evaluation is presented in Section VI. Finally, Section VII concludes.

II. Related Work

Dynamic Pricing and Incentivization. In online marketplace such as Ebay, pricing and payments decided by sellers/buyers and not by the owner of the marketplace. This however is not focused on profit maximization as required in a shared resource infrastructure model. Dynamic pricing in a cloud recommendation and marketplace have been explored in [7], [8]. The dynamism is on the prices but the costs (or incentives) are static since the back end infrastructure is dedicated and static. Literature in this area in general focuses on dynamic demands and assumes some static cost model of the elements [9]. In shared resource infrastructure, the cost can be dynamic since it is important to provide appropriate incentives for sharing resources. Literature in crowdsourcing and crowdsensing [10], [11], on the other extreme, focuses on dynamic supplies to determine dynamic incentives assuming some static budget. We focus on both dynamic demand as well as supply at the same time to determine dynamic prices and costs in an integrated manner.

Volunteer Computing, Grid Computing, and Reliable Scheduling. Volunteer and grid computing paradigms focus on resource sharing [12], [13]. End users sharing network bandwidth and network data plans have been considered in CrowdMAC [14]. Scheduling tasks over unreliable resources have also been considered in Map-reduce framework [15]. [16] has studied fault tolerant scheduling for shared resources, and [17] proposes scheduling of computing tasks on resources based on an estimation of network transfer time

of the task to the resource. Benjamin et. al. [18] propose proactive handling of resource failures in computing grid using replication without considering monetary cost of the same. A controlled market-oriented grid computing resource management that guarantee QoS has been addressed in [1]. However, resource management based on incentives and task prices in an integrated manner while maximizing profit and ensuring task reliability is unexplored.

Pricing and/or Incentive based Scheduling. [2]–[4] focuses on incentive based scheduling in a decentralized peer-to-peer computing grid. However, as mentioned previously, these approaches do not apply to the shared resources infrastructure vision because of the constrained market scenario. Monetary cost of scheduling task over distributed computing resources have been further considered in [19]. Pricing based scheduling that addresses demand response has been addressed for smart grids in general [5], [6]. However, combined decision making on the pricing and incentives at the same time for controlled shared resource infrastructure has not been addressed.

This paper focuses on a *first-of-a-kind* resource management framework that determines both dynamic incentives and prices in an integrated manner for constrained market environment to enable a large-scale robust infrastructure (guaranteeing SLAs and reliability to consumers) over unreliable and dynamically shared resources. The framework focuses on maximizing the profit while keeping both consumers and providers interested in the infrastructure.

III. System Model

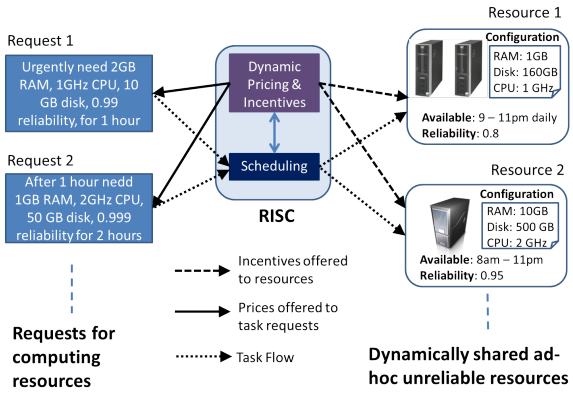


Figure 2. High-level architecture of *RISC*

In this section we give an overview and system model of the RISC framework. Fig. 2 shows a high-level architecture of RISC. Primarily, RISC takes reservation requests (which we also call *tasks*) as inputs and manages the dynamically shared ad-hoc resources (e.g., by the OSCs in return of some incentives) to host parts or whole of one or more requests. The reservation requests consist of the resource configuration required, when it is required, and for how long. The reservation requests are analogous to requesting a certain VM instance in modern day public clouds, e.g. Amazon, Azure, etc. RISC: (i) allocates the tasks over

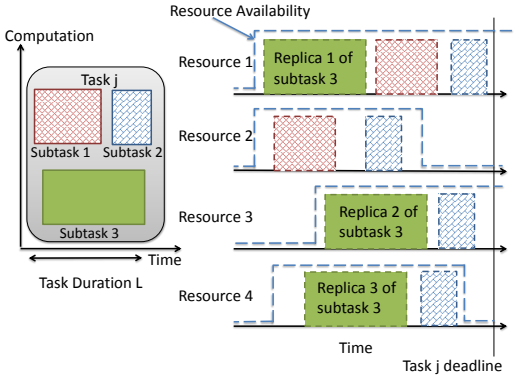


Figure 3. Example of a task and its resource allocation

heterogeneous resources, in presence of time-varying task arrival rate and resource capacity, and (ii) determines the prices and incentives offered for the requests and shared resources, respectively, in an integrated manner. We assume that time is divided into epochs of fixed duration, and the epochs are further subdivided into T time slots of fixed duration. At the beginning of each epoch, the Dynamic Pricing & Incentives module in RISC determines the prices and the incentives together based on the supply and the demand of resources. To determine the demand, the scheduler is invoked such that the cost is minimized. The pricing and incentivization ensures that the profit is maximized.

A. Tasks

Task or reservation request for computation resources, has a certain resource requirement for a certain duration. In particular, a task tk_j has four requirements: (i) the required resource capacity (or size), denoted by a size vector $size(j)$ that specifies requirement for compute, memory, and storage, (ii) the reservation length or duration $l(j)$, in terms of the number of slots, (iii) the deadline $d(j)$, which is the number of slots (from the arrival time slot) within which the reserved slots for the task should appear, and (iv) a desired (minimum) success probability $dsp(j)$ of the reservation in presence of resource failures or departures. A task tk_j from a user maybe composed of one or more smaller reservation request which can be allocated on different resources (see Fig. 3). We call these smaller reservation requests, subtasks, and denote them by $tk_{j,k}$. For a subtask $tk_{j,k}$, its *weight* $w(j,k)$ is defined as the product of its computation requirement and its duration. Depending on the total weight of each task, i.e. the sum of the weights of a task's subtasks, there are k distinct types of task requests $tt_1, \ldots, tt_k$. The demand for a task type tt_i is denoted as $|tt_i|$. All tasks of the same type can be offered the same price in a given time slot. The price vector pertaining to the distinct request types is denoted as $\vec{p} = \{p_1, \ldots, p_k\}$. For a time slot t, this is referred as $\vec{p}(t)$.

B. Resource

The system is composed of R computing resources or machines $r_1, \ldots, r_R$, whose capacity (a vector of compute, storage and memory capacities) may vary with time. A

Table I
SYMBOL TABLE

symbol	definition
T	number of time slots with fixed duration
R	total number of computing resources shared
n	total number of shared resource *types*
r_i	i-th resource, $1 \leq i \leq R$
rt_i	resource type i, $1 \leq i \leq n$
λ_i	mean of lifespan of r_i
pf	penalty factor, i.e. severity on incentive
tk_j	j-th task (reservation request)
k	total number of request types
tt_i	task type i, $1 \leq i \leq k$
$size(j)$	required capacity vector <cpu, mem, disk> of tk_j
$l(j)$	reservation length of tk_j in terms of no. of time slots
$dsp(j)$	desired success probability of task tk_j
p_i	price of request type i
c_j	incentive for resource type j
$\vec{p}$	price vector $\{p_i\}_{i\in\{1,\ldots,k\}}$
$\vec{c}$	incentive vector $\{c_j\}_{i\in\{1,\ldots,n\}}$
$D_i(\vec{p}\|\vec{r})$	0–1 indicator variable denoting acceptance of i^{th} request type for prices in $\vec{p}$
$D_{ij}(\vec{p},\vec{c}\|\vec{r})$	demand for j^{th} resource type for i^{th} request type when prices in $\vec{p}$ and incentives in $\vec{c}$ are offered
$ts_{j,k}$	k-th subtask of task tk_j
$RS(j,k)$	set of resources where replicas of $ts_{j,k}$ are placed
$FR(j,k)$	set of resources where replica allocation of $ts_{j,k}$ is feasible
$asp(j,k)$	allocation success probability of $ts_{j,k}$ over $RS(j,k)$
$dsp(j,k)$	desired success probability of $ts_{j,k}$
$idfp(j,k)$	inverse desired failure probability of $ts_{j,k}$, i.e. $idfp(j,k,i) = \frac{1}{1-dsp(j,k)}$
$MC(j,k,i)$	monetary cost for executing $ts_{j,k}$ on r_i
$NC(j,k,i)$	network cost for executing $ts_{j,k}$ on r_i
$\rho(j,k,i)$	efficiency of allocation of $ts_{j,k}$ on r_i
$ra(j,k,i)$	earliest feasible allocation of a replica of $ts_{j,k}$ on r_i
$p(j,k,i)$	probability that $ra(j,k,i)$ is successfully completed
$iafp(j,k,i)$	inverse allocation failure probability of an allocation $ra(j,k,i)$, i.e. $iafp(j,k,i) = \frac{1}{1-p(j,k,i)}$
$x(j,k,i)$	0–1 indicator variable denoting whether allocation of $ts_{j,k}$ over r_i is selected

resource can be unavailable in a given time slot, which is modelled by setting the resource capacity to $< 0, 0, 0 >$ in the corresponding slot. A resource can also fail or depart the system either permanently or for an extended period of time. Following the experimental results in [20], we assume that the lifespan of a resource r_i is exponentially distributed with a mean λ_i, i.e., the probability $p(j,k,i)$ that a task allocation $ra(j,k,i)$ is successfully completed is $e^{\frac{-d(j,k)}{\lambda_i}}$. Depending on the distinct capacity vectors, the resources are categorized into n types: $rt_1, \ldots, rt_n$. In a given time slot, same incentives needs to be offered to all the resources of same type. The cost vector pertaining to the distinct resource types is denoted as $\vec{c} = \{c_1, \ldots, c_n\}$ and for a time slot t, this is referred as $\vec{c}(t)$. $MC(j,k,i)$ is the monetary cost for executing subtask $tk_{j,k}$ on resource r_i, per unit computation per unit time. There is also a network (transfer) cost $NC(j,k,i)$ for transferring a subtask from the origin of the task tk_j to the resources r_i executing the task, per unit

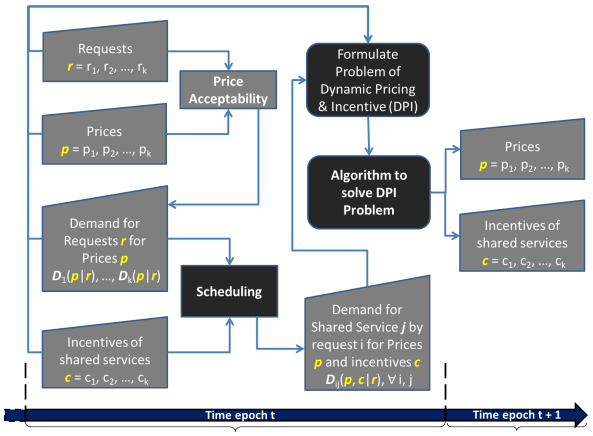


Figure 4. Process to determine dynamic prices offered for requests and incentives for shared resources

of computation and per time slot. $NC(j,k,i)$ may depend on the distance, number of network hops, or the bandwidth, between the origin of the task and the resource.

IV. PRICING AND INCENTIVIZATION IN RISC

This section describes the Dynamic Pricing and Incentivization (DPI) strategy. Figure 4 shows the process employed by DPI. The DPI problem is formulated based on price and incentive vectors from the last epoch along with the probability distributions of acceptance of prices for requests (determined by D_i). As shown in Figure 4, the actual demands of resources, captured by D_{ij}, can be determined through the scheduler that determines the task allocations. For RISC, the task allocation should try to minimize the cost to ensure the profit maximization. D_{ij} can also be weighted by a probability distribution of acceptance of incentives by the resource owners. The distributions can be learned online depending on the acceptance of prices as well as sharing of resources. The DPI problem can be formulated as follows:

$$\text{Max}_{\vec{p},\vec{c}} \sum_{r_i \in FR(j,k)} p_i D_i(\vec{p}|\vec{r}) - \sum_j c_j \sum_i D_{ij}(\vec{p},\vec{c}|\vec{r}) \quad (1)$$

where: $p_i \in \{l_i, l_i + \delta, l_i + 2\delta, \ldots, h_i\}$ and $c_j \in \{l_j, l_j + \delta, l_j + 2\delta, \ldots, h_j\}$. If $M = \max(\max_i \frac{h_i - l_i}{\delta}, \max_j \frac{h_j - l_j}{\delta})$, then a brute force algorithm that checks all possibilities of $\vec{p}, \vec{c}$ will have running time $O(M^{nk})$.

We propose a sequential optimization algorithm for DPI. First, the requests types are re-ordered according to decreasing demand, i.e., $|tt_1(t)| \geq |tt_2(t)| \geq \ldots \geq |tt_k(t)|$. Also, the resources are considered in the order in which we have their supply, i.e., type 1 resource is most abundant resource and type n is in lowest supply. After this ordering, each p_i and c_j is optimized sequentially. During the optimization of p_i, the prices p_x ($\forall x \neq i$) of all other types of requests as well as the incentives of all types of resources are assumed to be constant. For the first request type, the price p_1 is optimized assuming the prices of the other resources types from the previous epoch. For any request type tt_i, the prices till tt_{i-1} have already been optimized but for the prices for

1: At the beginning of each epoch
2: fix prices as $p_2 \leftarrow p_2(t), \ldots, p_k \leftarrow p_k(t)$
3: fix incentives as $\vec{c} \leftarrow \{c_1(t), \ldots, c_n(t)\}$
4: $p_1(t+1) \leftarrow \arg\max_{p_1} \sum_i p_i D_i(\vec{p}|\vec{r}) - \sum_j c_j \sum_i D_{ij}(\vec{p}, \vec{c}|\vec{r})$
5: **for** $2 \leq i \leq k$ **do** {*over all request type*}
6: $p_1 \leftarrow p_1(t+1), \ldots, p_{i-1} \leftarrow p_{i-1}(t+1)$
7: $p_{i+1} \leftarrow p_{i+1}(t), \ldots, p_k \leftarrow p_k(t)$
8: $p_i(t+1) \leftarrow \arg\max_{p_i} \sum_i p_i D_i(\vec{p}|\vec{r}) - \sum_j c_j \sum_i D_{ij}(\vec{p}, \vec{c}|\vec{r})$
9: fix prices as $p_2 \leftarrow p_2(t+1), \ldots, p_k \leftarrow p_k(t+1)$
10: $c_1(t+1) \leftarrow \arg\max_{c_1} \sum_i p_i D_i(\vec{p}|\vec{r}) - \sum_j c_j \sum_i D_{ij}(\vec{p}, \vec{c}|\vec{r})$
11: **for** $2 \leq j \leq n$ **do** {*over all resource type*}
12: $c_1 \leftarrow c_1(t+1), \ldots, c_{j-1} \leftarrow c_{j-1}(t+1)$
13: $c_{j+1} \leftarrow c_{j+1}(t), \ldots, c_n \leftarrow c_n(t)$
14: call scheduler to determine $D_{ij}(\vec{p}, \vec{c}|\vec{r})$
15: $c_j(t+1) \leftarrow \arg\max_{c_j} \sum_i p_i D_i(\vec{p}|\vec{r}) - \sum_j c_j \sum_i D_{ij}(\vec{p}, \vec{c}|\vec{r})$

Figure 5. The DPI algorithm

request types tt_{i+1} onwards are assumed from the previous epoch. The process continues iteratively until the prices for all the request types are optimized. The incentives for price optimization for all the request types are assumed to be from the previous epoch.

After the price optimizations are completed the incentive optimization is performed for all the resource types in a similar way. Figure 5 presents the algorithm. Since the resource and request types are ordered in decreasing supply and demand, respectively, the most wanted request types are optimized first in terms of price. From a cost perspective, the most available resource types are optimized. Intuitively, if the prices and incentives in the previous epoch were near optimal, and there are small perturbations in supply and demand from last epoch, then this sequential approach will lead to near optimal solution for the current epoch. The running time of the algorithm is $O(kM + nM *$ run time of scheduler$)$. It should be noted that with every change in the cost variables, the scheduler may have to be invoked to determine the actual demand D_{ij}.

V. Scheduling in RISC

The objective of RISC scheduler is to minimize both the monetary cost (MC) and network cost (NC) of executing each task while meeting the task's performance requirements. In particular, for a given task minimize $\alpha.MC + (1-\alpha).NC$, where *cost parameter* α is provided by the system operator. (We later discuss about how to choose α.)

Consider a task tk_j and one of its subtask $tk_{j,k}$. A task is successful if all its subtasks are successful. Therefore, for ensuring a desired success probability for the task, its subtasks are replicated over multiple resources. For a subtask $ts_{j,k}$, let $dsp(j,k)$ be its desired success probability. Let a *replica allocation* $ra(j,k,i)$ for a replica of the subtask $ts_{j,k}$ be a reservation of $size(j,k,i)$ over a resource r_i from a starting time slot $st(j,k,i)$ to a finishing time slot $st(j,k,i) + l(j,k) - 1$. Here, $size(j,k,i)$ is a capacity or size vector of compute, memory and storage of the task, and $l(j,k)$ is the duration of the subtask. At a given point in the execution of the scheduling algorithm, an allocation $ra(j,k,i)$ for subtask $tk_{j,k}$ is *feasible* provided that: (1) before the allocation is made, the residual capacity of resource r_i is greater than or equal to the size $size(j,k)$ for the subtask (for each component, compute, memory and storage) from the starting to the finishing time slot of the allocation, and (2) the finishing time slot of the allocation is same or lower than the deadline slot of the parent subtask. For ease of presentation, we denote by $ra(j,k,i)$, the earliest feasible allocation of $tk_{j,k}$ on resource r_i.

A subtask succeeds if any of its replicas succeed. Assume that the replicas of a subtask $tk_{j,k}$ are placed on different resources with uncorrelated lifespan, where $RS(j,k)$ denotes the set of resources on which the replicas are placed. Then the allocated success probability $asp(j,k)$ of the subtask, is $1 - \Pi_{i \in RS(j,k)}(1 - p(j,k,i))$.

To maintain the success probability requirement of the subtask, the scheduler needs to ensure that the allocated success probability of the subtask is greater than or equal to the desired success probability of the subtask, i.e., $asp(j,k) \geq dsp(j,k)$. Considering the reciprocal of the corresponding failure probabilities in the constraint, and then taking logarithm on both sides we have the transformed constraint: $\sum_{i \in RS(j,k)} log\big(iafp(j,k,i)\big) \geq log\big(idfp(j,k)\big)$, where $iafp(j,k,i) = \frac{1}{1-p(j,k,i)}$ and $idfp(j,k) = \frac{1}{1-dsp(j,k)}$.

We now formally describe the core problem at the heart of RISC. Let $FR(j,k)$ be the set of resource over which replica allocation for subtask $ts_{j,k}$ is feasible. (Note that some resources may not have feasible allocations.) The problem for a subtask is to select a subset of $FR(j,k)$ so as to minimize the following cost objective (where the $0-1$ indicator variable $x(j,k,i)$ denotes whether the allocation over resource r_i is selected):

$$\text{Min} \sum_{r_i \in FR(j,k)} \big(\alpha.MC(j,k,i) + (1-\alpha).NC(j,k,i)\big) .w(j,k).x(j,k,i) \quad (2)$$

such that the following constraint holds:

$$\sum_{r_i \in FR(j,k)} log\big(iafp(j,k,i)\big).x(j,k,i) \geq log\big(idfp(j,k)\big) \quad (3)$$

A. Subtask Scheduling

Before describing our scheduling algorithm over all tasks, we discuss the algorithm for a single subtask scheduling problem, as described in equations 2 and 3. We note that even the problem of scheduling a subtask is NP-Hard (which can be showed by a reduction from the knapsack problem, but we omit the proof due to lack of space). In RISC, the number of entities (resources, tasks, and subtasks) that are considered by the scheduling algorithm can be quite large. Therefore, we need a fast algorithm in practice, for which we propose a greedy algorithm next.

When a resource r_i is selected for subtask $tk_{j,k}$, we pay a cost of $\big(\alpha.MC(j,k,i) + (1-\alpha).NC(j,k,i)\big).w(j,k)$ in

1: At the beginning of each epoch
2: **while** task queue is not empty **do**
3: $tk_j \leftarrow$ a task dequeued from the task queue
4: $st_j \leftarrow$ list of subtasks of tk_j in decreasing order of their weight
5: **for** $1 \leq k \leq$ length(st_j) **do** {*over all subtask of* tk_j}
6: $FR(j,k) \leftarrow$ list of all resources over which allocation of subtask $tk_{j,k}$ is feasible
7: sort resources r_i in list $FR(j,k)$ in decreasing order $\frac{log\big(iafp(j,k,i)\big)}{\big(\alpha.MC(j,k,i)+(1-\alpha).NC(j,k,i)\big).w(j,k)}$
8: $SR(j,k) \leftarrow \emptyset$; $h \leftarrow 1$
9: **while** $\sum_{r_i \in SR(j,k)} log\big(iafp(j,k,i)\big) < log\big(idfp(j,k)\big)$ **and** $h \leq |SR(j,k)|$ **do**
10: add h^{th} element r_h of $FR(j,k)$ to $SR(j,k)$
11: reserve capacity for replica of $tk_{j,k}$ over resource r_h
12: $h \leftarrow h+1$;
13: **if** $\sum_{r_i \in SR(j,k)} log\big(iafp(j,k,i)\big) \geq log\big(idfp(j,k)\big)$ **then** accept subtask $tk_{j,k}$ **else** reject subtask $tk_{j,k}$
14: **if** all subtasks of tk_j are accepted **then** accept task tk_j **else** reject task tk_j and free all associated reservations

Figure 6. The task scheduling algorithm

the problem objective in equation 2, and get a profit (or benefit) of $log\big(iafp(j,k,i)\big)$ in terms of the progress made towards satisfying the constraint in equation 3. Thus, for the subtask, we select the resources for replica allocation from $FR(j,k)$ in the decreasing order of the their ratio of profit to cost, until the constraint in equation 3 is satisfied. We show in the appendix that this greedy algorithm provides provable performance guarantee, namely, its solution is within a small additive factor of the optimal.

B. RCS Scheduler

We now describe our scheduling algorithm over all tasks, which we call *Reliability, Cost, and SLA-aware scheduler* (*RCS-scheduler*). A pseudocode of the algorithm is given in Figure 6. All tasks or reservation requests in the system are enqueued in a task priority queue in the order of their deadlines. At the beginning of an epoch, the scheduling algorithm dequeues tasks one at a time, and allocates resources for each task. For each task tk_j, the scheduling method replicates every subtask $tk_{j,k}$ over one or more resources to achieve the desired success probability for the task. The subtasks of a task are considered in the decreasing order of the subtask weight (i.e., product of the compute requirement and duration). At the end of considering all subtask of a task, the task is accepted if all subtasks are allocated with the desired success probability. Next we describe how the replica allocation is selected for each subtask.

Consider a subtask $tk_{j,k}$. To allocate resources for subtask, we need to compute the desired success probability $dsp(j,k)$ of the subtask from the corresponding success probability $dsp(j)$ of the parent task. Note that, a task is successful if all its subtasks are successful. Assuming the success of the subtasks are independent events, we have $dsp(j,k) = \big(dsp(j)\big)^{\frac{1}{K}}$, where K is the number of subtasks of the current task.

Next, for the given subtask, we consider each resource r_i for feasible replica allocation, i.e., we check if the resource has enough residual capacity to execute a replica of the subtask within the task deadline. If an allocation is feasible, the algorithm finds the earliest such feasible replica allocation on the resource, and adds the corresponding r_i to the set $FR(j,k)$. Next, the resources in $FR(j,k)$ are selected in the decreasing order of:

$$\frac{log\big(iafp(j,k,i)\big)}{\big(\alpha.MC(j,k,i)+(1-\alpha).NC(j,k,i)\big).w(j,k)} \quad (4)$$

until sum of $log\big(iafp(j,k,i)\big)$ over the selected resources is greater than or equal to $log\big(idfp(j,k)\big)$ or all resources in $FR(j,k)$ have been considered. Let the set of selected resources be $SR(j,k)$. When adding a resource to $SR(j,k)$, the algorithm reserves capacity for a replica of subtask $tk_{j,k}$ as per the earliest feasible allocation over the resource. The subtask is accepted at the end of the allocation if the desired success probability is achieved, and is rejected, otherwise.

C. Resource Management using RCS scheduler

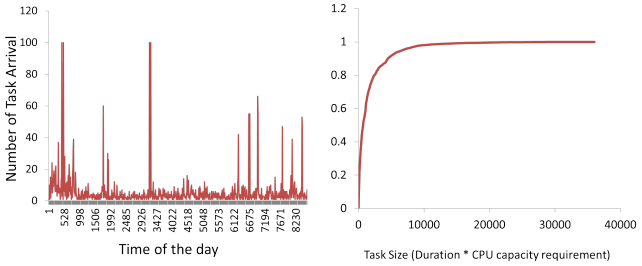
We now present some important aspects regarding usage and complexity of RCS scheduler in managing resources as part of the RISC framework.

Choice of α. The behavior of the scheduling algorithm is heavily dependent on the choice of α. A high value of α (i.e., closer to 1) gives more importance to monetary cost, and a low value gives more importance to network cost. The choice of α may be based on the overall condition of the system. For instance, $\alpha > 0.5$ should be chosen if the task monetary costs are considered to be too high. On the other hand, if network congestion or latency is a dominant problem, then $\alpha < 0.5$ should be chosen. If neither of the above (or both of the above) is a pressing concern for the system operator, α may be set to 0.5. For this paper, we choose $\text{RCS}_{\alpha=0.5}$ as the proposed scheduler.

Time-complexity. It is easy to see that , if the maximum duration and maximum deadline of a subtask is $D1$ and $D2$, respectively, the feasibility of a single resource can be checked in $O(D1.D2)$ in the worst-case. Thus scheduling of a single subtask can be done in $O(R.D1.D2)$, where R is the total number of resources. Assuming that the number of subtasks per task is bounded by a constant, the time-complexity of scheduling a task is also $O(R.D1.D2)$.

VI. Experimental Study

We perform trace-driven simulation to study the efficacy of RISC. For the dynamic pricing and incentive component of RISC, we use the proposed DPI algorithm, which is a *first-of-a-kind* method to determine prices and incentives at the same time. For the scheduler component of RISC, we consider RCS algorithm having an α of 0.5 ($\mathbf{RCS}_{\alpha=0.5}$), i.e. giving equal importance to network and monetary costs. The efficacy of the algorithm is shown by comparing with the following 4 scheduling algorithms: (ii) $\mathbf{RCS}_{\alpha=1}$: resources are selected based only on MC (i.e., greedily minimize



(a) Task arrival (b) CDF of task Size

Figure 7. Task distribution

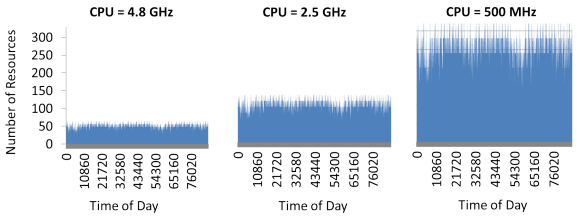


Figure 8. Availability of resources of different types

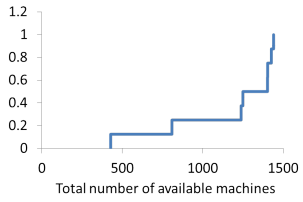


Figure 9. CDF of total number of available machines

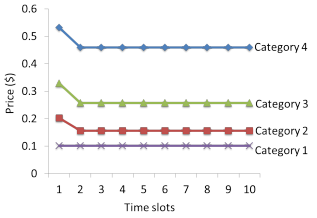


Figure 10. Convergence of offered prices

Table II
RELIABILITY LEVELS

Reliability Level	Task success probability
low	0.999–0.9999
moderate	0.9999–0.99995
high	0.99995–0.99999
very high	0.99999–0.999999

Table III
DEMAND-SUPPLY LEVELS

Demand Level	Definitions
low	no. of resources required (x) $\geq$ no. of resources available (y)
moderate	$0.5 < \frac{x}{y} < 1$
high	$\frac{x}{y} \leq 0.5$

monetary cost); (ii) **RCS**$_{\alpha=0}$: resources are selected based only on NC (i.e., greedily minimize network cost); (iii) **Earliest Completion-time First (ECF)**: resources are selected in the decreasing order of finishing time of the subtask (i.e., choose the resource according to the time when the earliest feasible allocation on the machine finishes); and (iv) **Highest Residual-capacity First (HRF)**: resources are load-balanced and chosen in the decreasing order of the average residual capacity on the machine over the subtask allocation duration.

A. Experimental Setup

We now describe the experimental set-up outlining the reservation (task) and resource (machine) traces.

1) Task Traces: We consider Google cluster usage trace [21] for 10000 consecutive tasks as the reservation requests to our system. In particular, we use the task inter-arrival time, task duration and task compute requirements from the Google trace. Each task has upto 3 subtasks, generated by either splitting the task across duration, compute requirement or both. The arrival pattern for a day of the tasks (Fig. 7(a)) and the cumulative distribution function (CDF) of the task sizes (Fig. 7(b)) show that the demand pattern pertaining to the task requests is extremely dynamic with most of the tasks are small is size, typical of any shared resource framework for OSCs. Depending on the task sizes, the tasks are classified into 15 different categories.

2) Resource Traces: We gathered CPU utilization traces for 24 hours of dozens of machines in a large organization over multiple locations in India, Europe and USA. We consider a machine available if the CPU usage is below 5%. Based on this, we create 8 distinct commonly occurring time-varying availability patterns. Additionally, for a larger set of machines from the same organization, we considered the machine configuration (CPU, memory and storage) and network hop distances between them. We select 5 distinct network hop distances (namely, 1, 2, 6, 7, and 13) that we observed, for the simulation. We also extract a set of 23 distinct resource configurations. Thus, there were 8*23 = 184 combinations or resource types in terms of configurations and availability. In our simulation, we consider a system with around 1500 machines, uniformly distributed across the 184 resource types. Resources having high capacity is less available compared to resources with low capacity (Fig. 8). Also, a minimum of 500 resources are available at any time as shown in the CDF in Fig. 9. This indicates a highly dynamic environment resource availability with a reasonable number of resource always available—as envisioned for the shared resource infrastructure being addressed by RISC.

3) Experiments performed: In the simulation, depending on the individual experiment, the task success probability is varied from 0.999 (three nines) to 0.999999 (six nines). In particular, we study four cases as shown in Table II. Further, the task deadline is varied as percentage of duration from 300% to 1700%.

Each machine is further allocated uniformly at random one of the five mean lifespan: 1, 5, 10, 30, or 60 days (which is used to compute the success probability of an allocation on the machine)[1]. Finally, for each machine, the prices per GHz per hour is chosen from the range of \$0.05 to \$0.5[2]. The acceptability of the prices by the task requesters depend on whether the prices are less than the corresponding prices in public cloud instances. Thus assuming the aforementioned price range within which prices are offered, the acceptance

[1]In [20], the authors have observed that the mean lifespan of resources in a particular volunteer computing project is of the order of few months.

[2]This range is in the order of modern IaaS cloud offerings

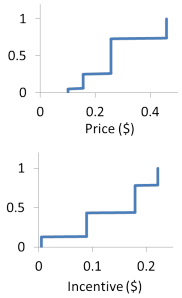


Figure 11. CDF of incentive offered and average price paid for a resource

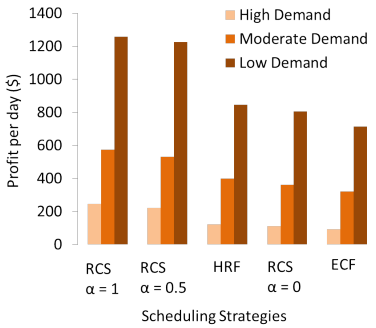


Figure 12. CDF of incentive offered and average price paid for a resource

probability of the prices would be 1. For experiments pertaining to the impact of supply and demand on the prices and incentives, we segregated the 24 hours time into three different types of zones with supply-demand characteristics shown in Table III.

B. Results

1) Efficacy of DPI: Fig. 10 shows the convergence of the prices for different categories of task requests. It is observed that within two epochs the convergence is achieved (for categories 2, 3, and 4) assuming there is no change in the supply and demand within these epochs. For category 1, the prices converge at the first epoch. This is typically the case for the rest of the 11 task categories (and thus is not shown in the figure for clarity). The results indicate a fast convergence of DPI.

Fig. 11 shows the CDFs of the incentives for a particular resource type as well as the prices paid for the resources. It should be noted that there are mainly three steps in the prices and incentives, referring to three different supply-demand characteristics in the shared resource infrastructure. Also, the incentives are lower compared to the prices paid, reflecting the profits for hosting the shared resource in RISC.

Fig. 12 shows the profit achieved by DPI for RISC. Interestingly, for low demand the total profit for a day is significantly high. This is counter-intuitive from a open marketplace perspective where high demand leads to higher profits. This deviation from open market standard is caused by the constrained market setting in a shared resource infrastructure. Since the price is bounded by (i.e. less than) the prices of the cloud instances, even at high demand, the price can not be increased in order to ensure that the task requesters accept the offered prices. However, the incentives go high for the resources leading to low profit for high demand. Furthermore, there are high number of tasks with low demand causing a different multiplicative scale while computing the total profit.

It is also notable that for different scheduling algorithm the profit changes. This is attributed to the monetary cost minimization by the scheduling algorithms as described in the following sections and shown in Figs. 13(a) and 14(a).

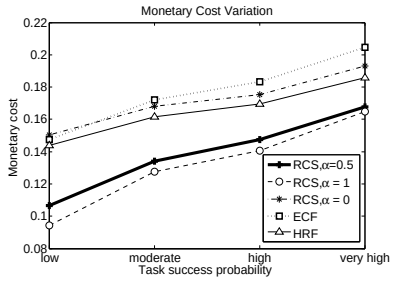
2) Task success probability and task scaling: Figure 13 presents the average monetary cost and network cost over all tasks, and total number of replicas over all accepted tasks, for the different requirements on success probability. Not surprisingly, as the success probability requirement is increased, monetary cost, network cost and number of replicas increase. However, we notice that our algorithm ($RCS_{\alpha=0.5}$) consistently performs second best both in terms of monetary and network cost, and its costs are close to the best performing algorithm ($RCS_{\alpha=1}$ for MC and $RCS_{\alpha=1}$ for NC). Thus, our algorithm provides a good balance between the two cost metrics. We also note that, for all algorithms other than ECF, the task rejection rate is 0 for the lower 3 success probability ranges, and is around 3% for the highest range[3]. ECF suffers from a much higher rejection rate and thus, requires a lower total number of replicas.

To observe the performance of the algorithms with increase in system workload, we also conduct experiments by varying task compute requirement and varying task duration. We observe similar trends as the task success probability experiment: our algorithm performs second best (and is close to the best algorithm) in both monetary and network cost metrics, and thus provides a good trade-off between the two.

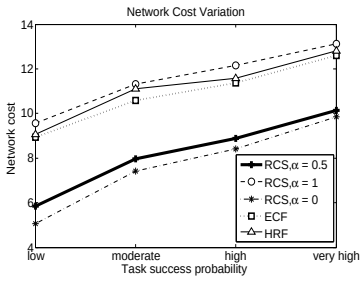
3) Task deadline: Figure 14(a) and 14(b) presents the average monetary and network costs with increase in deadline. We see that initially the monetary cost increases for all algorithms until the deadline increases to 700%. For subsequent increase in deadline, for two algorithms ($RCS_{\alpha=0.5}$ and $RCS_{\alpha=1}$) there is a decrease in monetary cost, but it remains almost the same for other algorithms. Similar trend is observed for network cost, where two algorithms ($RCS_{\alpha=0.5}$ and $RCS_{\alpha=0}$) perform significantly better than other 3 algorithms for higher values of deadline. This shows a clear benefit of using our algorithm, its cost is not only second-best (and close) to the cost of best algorithm for both the monetary and network cost, but also shows a decreasing trend as the deadlines are relaxed.

To explain the trend in the monetary cost, we divide the deadline variation range into two regions: when deadlines are strict (below 900%) and when deadlines are large (above 900%). In the former region, many large tasks (duration higher than 500 seconds) are rejected, and the number of accepted large tasks increases as the deadline is increased (as shown in Figure 15). Consequently, since the larger tasks have larger costs, the cost increases with increase in deadline in this region. In the second region, due to larger deadlines, all tasks are accepted (as shown in Figure 14(c)), and therefore, algorithms that consider monetary cost while resource selection ($RCS_{\alpha=0.5}$ and $RCS_{\alpha=1}$) improve their monetary cost by exploiting relaxation of deadline. However, other 3 algorithms that do not consider monetary cost while choosing resources, are unable to reduce the monetary cost as deadlines are relaxed. Similar explanation holds for network cost, where algorithm that consider network cost

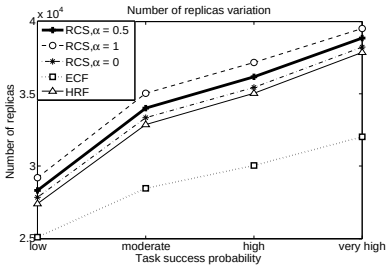
[3]The rejection rate plot is not shown due to lack of space.



(a) Monetary cost

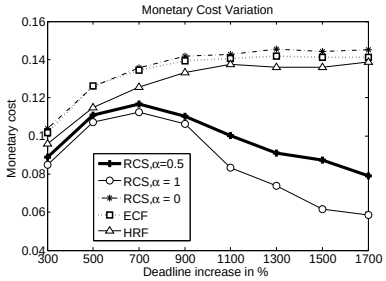


(b) Network cost

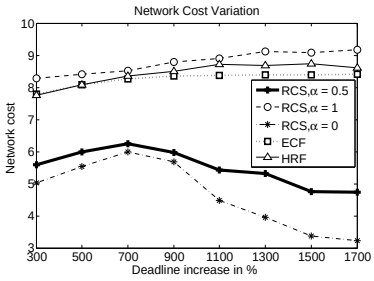


(c) Number of replicas

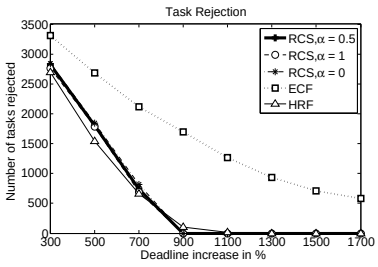
Figure 13. Varying task success probability.



(a) Monetary cost



(b) Network cost



(c) Task rejections

Figure 14. Varying task deadline

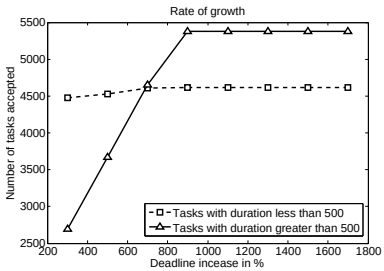


Figure 15. Varying task deadline: acceptance of large tasks for $\text{RCS}_{\alpha=0.5}$

($\text{RCS}_{\alpha=0.5}$ and $\text{RCS}_{\alpha=0}$), exploit the relaxation of deadline to reduce the network cost in second region.

VII. Conclusions

This paper introduced RISC, a novel framework for composing a robust computing infrastructure out of heterogeneous and unreliable resources that are dynamically shared in an ad-hoc manner by the owners in return of incentives. RISC addresses constrained resource marketplace situation, where: (i) requests and resources are dynamically priced and incentivized, respectively, based on supply demand while maximizing profit; and (ii) the prices offered are below than any alternatives (e.g. cloud offerings) for infrastructure consumers and prices & incentives are such that a required reliability guarantees can be ensured by the infrastructure. To this effect, RISC incorporates the DPI strategy for dynamic price and incentive determinations and a DPI dependent RCS scheduler that ensures the reliability and cost minimization from both monetary and network perspective while allocating requests to the shared resources. Results based on experiments over real request and resource traces show that DPI indeed leads to profit maximization. Interestingly, the profit reduces with increased demand because of the constrained market situation. Further, RCS can ensure reliability while balancing monetary and network cost towards the aforementioned profit maximization. Future work includes incorporation of online learning techniques for price and incentive acceptance in RISC.

References

[1] R. Buyya and K. Bubendorfer, *Market-Oriented Grid and Utility Computing*. Wiley, 2010.

[2] Y. Zhu, L. Xiao, L. Ni, and Z. Xu, "Incentive-based p2p scheduling in grid computing," ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004.

[3] L. Xiao, Y. Zhu, L. Ni, and Z. Xu, "Incentive-based scheduling for market-like computational grids," *IEEE Transactions on Parallel and Distributed Systems*, 2008.

[4] L. Xiao, Y. Zhu, L. M. Ni, and Z. Xu, "Gridis: An incentive-based grid scheduling," in *IEEE IPDPS05*, 2005.

[5] J. Yang, G. Zhang, and K. Ma, "Real-time pricing-based scheduling strategy in smart grids: A hierarchical game approach," *Journal of Applied Mathematics*, 2014.

[6] D. Li, S. K. Jayaweera, O. Lavrova, and R. Jordan, "Load management for price-based demand response scheduling—a block scheduling model," in *International Conference on Renewable Energy and Power Quality (ICREPQ)*, 2011.

[7] T. Mukherjee, K. Dasgupta, S. Gujar, G. Jung, and H. Lee, "An economic model for green cloud," in *ACM MGC*, 2012.

[8] G. Jung, T. Mukherjee, S. Kunde, H. Kim, N. Sharma, and F. Goetz, "Cloudadvisor: A recommendation-as-a-service platform for cloud configuration and pricing," in *IEEE SERVICES*, 2013.

[9] S. Lehmann and P. Buxmann, "Pricing strategies of software vendors," *Journal of Business & Information Systems Engineering*, 2009.

[10] D. Zhao, X.-Y. Li, and H. Ma, "How to crowdsource tasks truthfully without sacrificing utility: Online incentive mechanisms with budget constraint," in *IEEE INFOCOM*, 2014.

[11] I. Koutsopoulos, "Optimal incentive-driven design of participatory sensing systems," in *IEEE INFOCOM*, 2013.

[12] J. Zhang and C. Phillips, "Job-scheduling via resource availability prediction for volunteer computational grids," *Int. J. Grid Util. Comput.*, vol. 2, no. 1, pp. 25–32, May 2011.

[13] D. Anderson and K. Reed, "Celebrating diversity in volunteer computing," in *HICSS*, 2009.

[14] N. Do, C.-H. Hsu, and N. Venkatasubramanian, "Crowdmac: A crowdsourcing system for mobile access," in *ACM/Usenix/IFIP Middleware*, 2012.

[15] H. Lin, X. Ma, J. Archuleta, W.-c. Feng, M. Gardner, and Z. Zhang, "MOON: MapReduce On Opportunistic eNvironments," in *ACM HPDC*, 2010, pp. 95–106.

[16] S. Choi, M. Baik, C. Hwang, J. Gil, and H. Yu, "Volunteer availability based fault tolerant scheduling mechanism in desktop grid computing environment," in *IEEE NCA*, 2004.

[17] J. Jurkiewicz, K. Nowinski, and P. Baa, "Prediction of the jobs execution on the community grid with added network latency," in *Distributed and Parallel Systems*. Springer, 2008.

[18] B. B. Khoo and B. Veeravalli, "Pro-active failure handling mechanisms for scheduling in grid computing environments," *Journal of Parallel and Distributed Computing*, 2010.

[19] J. Broberg, S. Venugopal, and R. Buyya, "Market-oriented grids and utility computing: The state-of-the-art and future directions," *Journal of Grid Computing*, 2008.

[20] E. M. Heien, D. P. Anderson, and K. Hagihara, "Computing low latency batches with unreliable workers in volunteer computing environments," *J. Grid Comput.*, 2009.

[21] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format + schema," Google Inc., Tech. Rep., 2011, revised 2012.03.20. Posted at URL http://code.google.com/p/googleclusterdata/wiki/TraceVersion2.

[22] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack problems*. Springer, 2004.

APPENDIX

We show that the greedy subtask scheduling algorithm presented in Section V is within an additive factor of the optimal solution for the subtask. First, we note that the subtask scheduling problem can be modeled as Minimization Knapsack (MinKP) problem [22]: Min $\sum_{i=1}^{n} w_i x_i$ subject to $\sum_{i=1}^{n} p_i x_i \geq B$ and $x_i \in \{0, 1\}$.

Consider any subtask $ts_{j,k}$. The subtask problem defined in equations 2 and 3 is the MinKP problem with $w_i = \big(\alpha.MC(j,k,s) + (1-\alpha).NC(j,k,s)\big).w(j,k)$, $p_i = log\big(iafp(j,k,i)\big)$, $B = log\big(idfp(j,k)\big)$, and n as the number of resources in $FR(j,k)$. Since the task size and reliability values can be chosen to result in arbitrary positive integral values of w_i, p_i, and B, the subtask scheduling problem is as hard as MinKP, and therfore NP-Hard.

Next, for ease of presentation, assume that $\frac{p_1}{w_1} > \frac{p_2}{w_2} > \ldots > \frac{p_n}{w_n}$. Let, $\sum_{i=1}^{m} p_i = P_m$ and $\sum_{i=1}^{m} w_i = W_m$. Let s be an index such that $P_{s-1} < B \leq P_s$. Item s is called the split-item of the MinKP problem instance. Note that our greedy algorithm outputs the solution containing items $\{1, \ldots, s\}$.

For a parameter $Y > 0$, let MinKP(Y) denote the MinKP where B replaced by Y. For a problem P, let P* denote the value of an optimal solution (if it exist) to that problem.

Let MinFracKP(Y) be same as the MinKP except that the integrality constrain on x_i is replaced by $0 \leq x_i \leq 1$. Let FracKP(Y) be the standard fractional (maximization) knapsack problem as: Max $\sum_{i=1}^{n} w_i x_i$ subject to $\sum_{i=1}^{n} p_i x_i \leq Y$ and $0 \leq x_i \leq 1$.

Now we show that the solution returned by the greedy algorithm is within an additive factor of optimal.

Lemma 1: W_{s-1} = MinFracKP$^*(P_{s-1})$ = MinKP$^*(P_{s-1})$ $\leq$ MinKP$^*(B) \leq$ MinKP$^*(P_s)$ = MinFracKP$^*(P_{s-1}) = W_s$.

Proof: (Sketch.) We start with two straightforward observations. First, note that, as the value of Y increases, the optimal value of MinKP(Y) is non-decreasing. Therefore, from the definition of s, we have MinKP$^*(P_{s-1}) \leq$ MinKP$^*(B) \leq$ MinKP$^*(P_s)$. Second, it is easy to see that, $(y_1, \ldots, y_n)$ is an optimal solution for MinFracKP(Y) if and only if $(1-y_1, \ldots, 1-y_n)$ is an optimal solution for FracKP($P_n - Y$).

Using the greedy choice property of FracKP (from Theorem 2.2.1 of [22]), we know that $(x_1 = 0, \ldots, x_{s-1} = 0, x_s = 1, \ldots, x_n = 1)$ is an optimal solution of FracKP($P_n - P_{s-1}$), i.e., greedily choosing items (either in full or in fraction) in the ascending order of $\frac{p_i}{w_i}$ until the sum of p_i remains less than or equal to $P_n - P_{s-1} = \sum_{i=s}^{n} p_i$, gives an optimal solution to FracKP($P_n - P_{s-1}$). Then, $(x_1 = 1, \ldots, x_{s-1} = 1, x_s = 0, \ldots, x_n = 0)$ is an optimal solution for MinFracKP(P_{s-1}).

Note that, the values of x_i in the above optimal solution of MinFracKP(P_{s-1}) are all integral. Therefore, it is also an optimal solution for MinKP(P_{s-1}). Therefore, W_{s-1} = MinFracKP$^*(P_{s-1})$ = MinKP$^*(P_{s-1})$.

We can similarly show, MinKP$^*(P_s)$ = MinFracKP$^*(P_{s-1})$ = W_s. This observation completes the proof. ■

Lemma 2: Consider any subtask $ts_{j,k}$. The subtask scheduling algorithm proposed in this paper is within an additive factor of $w_s = \big(\alpha.MC(j,k,s) + (1-\alpha).NC(j,k,s)\big).w(j,k)$ of the optimal, where s is the split-item of the subtask scheduling problem instance.

Proof: Follows immediately from two simple observations: (i) in Lemma 1, $W_{s-1} = W_s - w_s \leq$ MinKP$^*(B)$ $\leq W_s$, and (ii) our greedy algorithm outputs a solution containing items $\{1, \ldots, s\}$ which has the objective value of W_s. ■