Optimal Resource Allocation in Clouds

Fangzhe Chang
Bell Labs, Alcatel-Lucent
fangzhe@bell-labs.com

Jennifer Ren
Bell Labs, Alcatel-Lucent
reny@bell-labs.com

Ramesh Viswanathan
Bell Labs, Alcatel-Lucent
rv@bell-labs.com

***Abstract*—Cloud platforms enable enterprises to lease computing power in the form of virtual machines. An important problem for such enterprise users is to understand *how many and what kinds of virtual machines will be needed from clouds*. We formulate demand for computing power and other resources as a resource allocation problem with multiplicity, where computations that have to be performed concurrently are represented as tasks and a later task can reuse resources released by an earlier task. We show that finding a minimized allocation is NP-complete. This paper presents an approximation algorithm with a proof of its approximation bound that can yield close to optimum solutions in polynomial time. Enterprise users can exploit the solution to reduce the leasing cost and amortize the administration overhead (e.g., setting up VPNs or configuring a cluster). Cloud providers may utilize the solution to share their resources among a larger number of users.**

I. Introduction

With the support of virtualization technology [1], [2], computing power can now be packaged in the form of virtual machines (VM) on cloud platforms and leased to individual or enterprise users, e.g., as in Amazon Elastic Compute Cloud (EC2). Users can even lease several virtual machines from a cloud and create a secure highly scalable common platform for all their computations. For instance, in a hybrid cloud model (see Fig. 1), an enterprise can seamlessly extend its private network to securely include remote servers in a public cloud. The enterprise can save equipment cost by provisioning its private network for average demand of computing power, and by extending to leased machines in public clouds for peak demand. An important problem for an enterprise user is to understand *how many and what kinds of virtual machines and other resources it will need*.

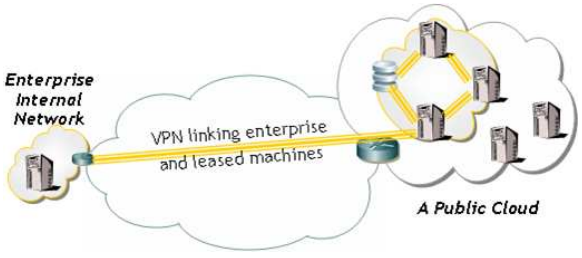

Fig. 1. Virtual subnet in public clouds

This paper studies the scenarios where an enterprise would like to manage cloud resource reservations together for all its computational needs. This makes it easier for the enterprise to perform administrative tasks, e.g., establishing VPN connections or provisioning a clustering platform. The administration overhead can then be amortized over whole group of computations.

Even though dependencies exists among computation tasks in a group, for simplicity we assume that tasks can be listed into a sequence and that a later task in the sequence can reuse resources released by an earlier task. Such a sequence generates a sequential batch of resource requests. This paper aims at minimizing the number or the cost of resources that need to be reserved for a batch of requests.

Resources required by tasks can be heterogeneous. Heterogeneity appears in different resource types (e.g., VMs, block storage, message queues, and databases), or in different configurations of a same type (e.g., VMs configured to have different number of CPUs or memory capacity). When describing resource requirements, a computation task can either refer to resources at configuration name level (e.g., c1.xlarge in Amazon EC2) or at detailed attribute level (e.g., 20 Compute Unit, 1690 GB storage). In this paper, for simplicity resources are referred to by a single configuration name. One can easily convert attribute-based requirements to configuration name based requirements. For instance, using Table I, we can easily

TABLE I
EC2 resource configurations

Configuration	Memory (GB)	Compute Unit	Disk (GB)	Platform (bit)	Cost ($/h)
m1.small	1.7	1	160	32	0.1
m1.large	7.5	4	850	64	0.4
m1.xlarge	15	8	1690	64	0.8
c1.medium	1.7	5	350	32	0.2
c1.xlarge	7	20	1690	64	0.8

translate a resource requirement 'Memory $>$ 2 GB and Compute Unit $>$ 3' into and the equivalent requirement 'm1.large, or m1.xlarge, or c1.xlarge' with configuration names.

For a batch of resource requirements, a naive approach for figuring out what resources need to be reserved is to simply add the requests together (i.e., ignoring reusability among different computations). For instance, in a scenario where an enterprise wants run two computations one after another, with the resource requirements depicted in Table II, an naive approach might reserve fifteen resources: six c1.xlarge machines for the first task and eight m1.large machines plus one 100GB database for the second task, whereas in fact a reservation of six m1.xlarge machines, two m1.large machines and an 100GB

978-0-7695-4130-3/10 $26.00 © 2010 IEEE
DOI 10.1109/CLOUD.2010.38

TABLE II
AN EXAMPLE RESOURCE REQUIREMENTS

Task	Computing Node	Database
1	6× c1.xlarge or m1.xlarge	
2	8× m1.large	1× 100GB

TABLE III
REQUIREMENTS WITH SUBSUMPTION

Task	Computing Node (1st-slot)	Database (2nd-slot)
1	({c1.xlarge, m1.xlarge}, 6)	
2	({m1.large, **m1.xlarge**}, 8)	({100GB}, 1)

database would be sufficient, since six m1.xlarge machines are powerful enough to be used by both task 1 and task 2.

We formulate the batch resource allocation problem precisely below. An instance of the batch resource allocation problem is given by

- a resource universe $\mathbb{B}$ consisting of multiple copies of various resource configurations to be allocated, and
- a batch of k number of tasks $R = \{R_1, \cdots, R_k\}$. Each task R_i has a sequence of slots $x_{i1}, \cdots, x_{im_i}$ (with $m_i >= 1$) representing the collection of m_i kinds of resources required by task R_i, written as $R_i = [x_{i1}, x_{i2}, ..., x_{im_i}]$.

The resource universe $\mathbb{B}$ is a a multiset (i.e., a bag) since it may contain many copies of a same resource. We use notation B to represent the base set of $\mathbb{B}$, and multiplicity function $m_{\mathbb{B}}(b)$ to represent the number of copies a resource $b \in \mathbb{B}$ (i.e., $B = \{b \in \mathbb{B} | m_{\mathbb{B}}(b) > 0\}$). Each slot $x_{ij} \in 2^B \times \mathbf{N}$ (where 2^B denotes the powerset of B and **N** the set of natural numbers) describes what types of resource configurations are requested together with the number of copies to satisfy the requirement from j-th slot of task R_i.

For instance, the resource requirement of Task 1 in Table II has exactly one slot $(\{\text{m1.xlarge}, \text{m1.large}\}, 6)$ requesting six copies of either c1.xlarge, or c1.xlarge machines, or their combinations.

Subsumption relationship among resources exists in real world. For instance, a m1.large machine is subsumed by a m1.xlarge machine (but not by c1.xlarge due to memory attribute). In other words, a request for two m1.large machines can be satisfied by either two m1.large machines, or two m1.xlarge machines, or one m1.large and one m1.xlarge machine. A natural way to model such subsumption relations is as a partial order. Our problem formulation is general enough to encode such subsumption relations specifiable as a partial order. Specifically, the set of acceptable resources specified in each slot x_{ij} can be taken to be the closure with respect to any subsumption relations so that it includes any resource kind that can subsume a resource kind included in the set. (In the terminology of partial orders, this would be an upper closed set.) For example, the fact that two ml.large large machines can also be satisfied by an allocation of two ml.xlarge machines would be encoded as the slot $(\{\text{m1.large}, \text{m1.xlarge}\}, 2)$. With subsumption relations, the requirements of the above example is specified in Table III.

Resources are allocated to meet the demands from all slots of all tasks. A resource allocation $\mathbb{A}$ is a function with $\mathbb{A}(i,j) \subseteq \mathbb{B}$ identifying which resources and how many copies are allocated for the j-th slot of task R_i (note that $\mathbb{A}(i,j)$ is also a multiset which allows allocation of multiple copies of a resource to a slot). We use $\mathbb{A}(i)$ to denote all resources allocated to task R_i by allocation $\mathbb{A}$, i.e., $\mathbb{A}(i) = \{(b, \sum_{1 \le j \le m_i} m_{\mathbb{A}(i,j)}(b))\}$.

A resource allocation $\mathbb{A}$ is *valid* (or satisfying) iff

1) $|\mathbb{A}(i,j)| = x_{ij}$.snd and for all b, $b \in \mathbb{A}_{ij}$ implies $b \in x_{ij}$.fst (where $|s|$ denotes the size of the set s, fst and snd denotes the first and the second element of a pair respectively), for all $1 \le i \le k$ and $1 \le j \le m_i$. In other words, the resource requests from every slot have to be exactly honored, using a combination of resources allowed in that slot; and
2) $\mathbb{A}(i) \subseteq \mathbb{B}$, for all $1 \le i \le k$. In other words, the total number of allocated resource copies to any task has to be no more than the number of available instances in the resource universe $\mathbb{B}$.

The multiset of resources allocated by an allocation $\mathbb{A}$, denoted $\mathbb{B}(\mathbb{A})$, is given by

$$\mathbb{B}(\mathbb{A}) = \{(b, \max_{1 \le i \le k} m_{\mathbb{A}(i)}(b)) | b \in \mathbb{B}\}$$

where (b, n) denotes n copies of b in $\mathbb{B}(\mathbb{A})$. In other words, $\mathbb{B}(\mathbb{A})$ consists of maximum number of copies of resource b allocated to each individual task (i.e., $\max_{1 \le i \le k} m_{\mathbb{A}(i)}(b)$), for $b \in B$. Note importantly, that in computing the resources allocated to any task $A(i)$, the resources allocated to each of the slots x_{ij} were *added* corresponding to their concurrent execution while across different tasks taking only the maximum corresponds to the fact that resources can be shared/resources among the sequentially executing tasks. The *batch allocation problem* is to find a valid resource allocation $\mathbb{A}$ such that

$$|\mathbb{B}(\mathbb{A})| = \sum_{b \in \mathbb{B}(\mathbb{A})} m_{\mathbb{B}(\mathbb{A})}(b)$$

is minimum. In other words, the total number of copies of all resources allocated by $\mathbb{A}$ is minimized.

In the above definition, note that specifying the sequential components of the task $\{R_1, \ldots, R_k\}$ enables the opportunity to minimize the allocated resources by reusing them across the tasks that run sequentially. A task consisting purely of parallel components is a special instance of our problem — namely, a batch consisting of one task ($k = 1$). For this special instance of a singleton batch, the algorithmic problem of finding a minimum allocation admits a particularly simple solution (given by Theorem 1 in Section II).

We can also associate a monetary cost $c(b)$ for each resource $b \in B$ (i.e., a flat rate nonnegative real number cost for each resource configurations by function $c : B \to \mathbf{R}^+$) and try to minimize the total cost $|\mathbb{B}(\mathbb{A})|_c$, given by

$$|\mathbb{B}(\mathbb{A})|_c = \sum_{b \in \mathbb{B}(\mathbb{A})} c(b) \times m_{\mathbb{B}(\mathbb{A})}(b).$$

We denoted it as *batch resource allocation with cost problem* $(\mathbb{B}, R, c)$. Obviously minimizing the allocation size is a special instance of minimizing the total cost where every resource is assigned the same unit cost of 1. In this paper, we will discuss both of them together.

We studied a simpler problem in the context of automated system testing in [3] where each resource request only allows one instance and no monetary cost is considered. The general problem is better suited for reserving resources in cloud computing platforms, e.g., allowing request for multiple copies of m1.large machines and for reducing monetary cost. The contribution of this paper is to investigate a more general problem which allows resources to be associated with cost and allows a task to request several instances of a same resource, resulting in new allocation algorithms. We have developed a new approximation algorithm with its approximation analysis.

This paper is organized as follows. Section II studies complexity of the batch allocation problem. Section III describes an approximation algorithm whose bound is proved in Section IV. Section V evaluates the proposed algorithm using simulation. Section VI discusses related work. Section VII summarizes the paper.

II. Problem Complexity

We first consider the problem piece on allocating resources to a single task and show that it is solvable in polynomial time. With optimization requirement over a batch of tasks the problem turns out to be NP-Complete.

Given a single task $R_i = [x_{i1}, \ldots, x_{im_i}]$ $(1 \le i \le k)$, a resource multiset $\mathbb{S} \subseteq \mathbb{B}$, the *feasibility problem* asks whether the resource multiset has enough resource copies to satisfy the task's requirement. The feasibility problem can be reduced to the following instance $G_i(\mathbb{S})$ of the max flow problem with respect to R_i and $\mathbb{S}$, as in Fig. 2.

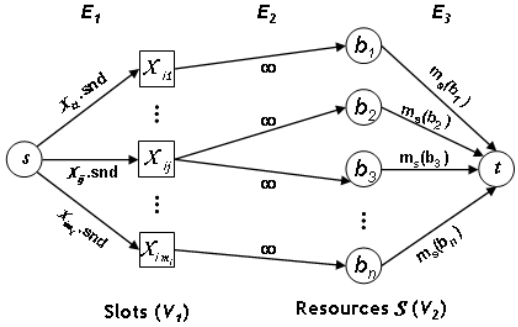

Fig. 2. Feasibility as a max flow problem

- *Flow graph* $G_i(\mathbb{S})$: Define the flow graph $G_i(\mathbb{S}) = (V, E, s, t, c)$, with $V = \{s, t\} \cup V_1 \cup V_2$, where $V_1 = \{v_1, ..., v_{m_i}\}$ (i.e., one vertex for each slot position of task R_i) and $V_2 = S$ (where S is the base set of the multiset $\mathbb{S}$) with the additional vertices s and t denoting the source and sink respectively. The edge set E is defined by $E = E_1 \cup E_2 \cup E_3$ with $E_1 = \{s\} \times V_1$ (*i.e.*, an edge from the source to each slot), $E_2 = \{(v_j, r) \mid j = 1, \ldots, m_i, r \in S \cap x_{ij}.\text{fst}\}$ (*i.e.*, edges between slot j and the resources from S that can be placed in slot j), and $E_3 = S \times \{t\}$ (*i.e.*, an edge from each resource in S to the sink).
- *Capacity function*: The capacity function p is defined as follows. For any $e = (s, v_j) \in E_1$, we define $p(e) = x_{ij}.\text{snd}$ (*i.e.*, the number of resources that need to filled in slot j); for any $e \in E_2$, define $p(e) = \infty$ and for any $e = (r, t) \in E_3$, define $p(e) = m_{\mathbb{S}}(r)$ (*i.e.*, the number of copies of resource r in multiset $\mathbb{S}$). Denote the maximum flow of this graph by $M(G_i(\mathbb{S}))$.

Theorem 1. *A task $R_i = [x_{i1}, \ldots, x_{im_i}]$ is satisfiable by resources $\mathbb{S}$ iff $M(G_i(\mathbb{S})) = \Sigma_{j=1,\ldots,m_i} x_{ij}.\text{snd}$.*

Proof: There is a one-to-one correspondence between allocations A and flows f defined by taking $A(i, j)$ to be the multiset $\{(b, f(v_j, b)) \mid f(v_j, b) > 0\}$, *i.e.*, flows on edges in E_2. Flow conservation at the nodes $b \in V_2$ ensures that no more than the available copies in $\mathbb{S}$ are used in the allocation A. The choice of edges in E_2 ensures that any resource b allocated is in x_{ij}.fst. Finally, flow conservation at the nodes $v_j \in V_1$ implies that $f(s, v_j) = |A(i, j)|$. It therefore follows that the allocation A defined by the flow meets the constraint of being a valid allocation iff $f(s, v_j) = x_{ij}.\text{snd}$ iff the total flow $|f| = \Sigma_j f(s, v_j) = \Sigma_j x_{ij}.\text{snd}$. ■

When monetary cost is considered, the feasibility problem can be extended with cost, as follows: given a resource universe $\mathbb{B}$, a single computation task R_i, and a cost threshold c', whether the resource universe has enough resource copies in $\mathbb{B}$ such that the total cost is no more than c' to satisfy the Task R_i's requirements. Correspondingly edges in Fig. 2 are associated with cost for unit flow: Every edge $e = (r, t) \in E_3$ (from a resource r to the sink node t) is additionally labeled with with a cost of $c(b)$; all other edges are given a cost of 0. A feasibility problem with cost can be reduced to the minimum-cost maximum flow problem [4] using graphs constructed this way.

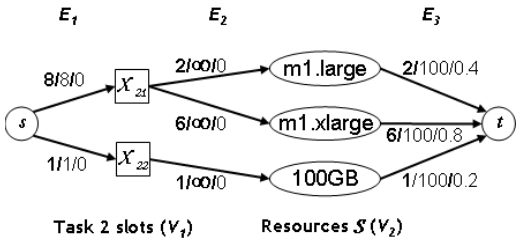

Fig. 3. A max flow for Task 2 in Table III

Fig. 3 shows a max flow for Task 2 and resources $\mathbb{S} = \{(\text{m1.large}, 100), (\text{m1.xlarge}, 100), (\text{100GB}, 100)\}$, where each edge is labeled with flow/capacity/cost. The shown max flow allocates 2 m1.large machine, 6 m1.xlarge machine, and 1 100GB database, with a total cost of $2 \times 0.4 + 6 \times 0.8 + 1 \times 0.2 = 5.8$. Apparently if considering Task 2

alone, this allocation is not the least-cost one. Though, it turns out to be the minimum-cost allocation when taking into account resource requirements from both Task 1 and Task 2 of Table III, since because Task 1 and Task 2 can then share the cost of the six m1.xlarge machines. The sharing among tasks make locally sub-optimal allocations better from global point of view.

Theorem 2. *The batch resource allocation problem* $(\mathbb{B}, R, c)$ *is NP-complete.*

Proof: For the decision version of the problem, given cost c', we guess a subset of $\mathbb{B}$ of cost c' and compute a max flow on constructed graphs as in Fig. 2 repeatedly for each task in the batch R to determine whether the guessed subset admits a valid allocation. Since computing max flow is known to be polynomial, this way we have a nondeterministic polynomial (NP) algorithm for the batch resource allocation problem. On the other hand, the hitting set problem [5], is a special instance of the batch resource allocation problem when each task has exactly one slot and each resource has a fixed cost of one. Since the hitting set problem is known to be NP-complete, the batch resource allocation problem must also be NP-Complete. ■

The NP-Completeness result means that it is unlikely to find an efficient algorithm that gives the optimal solution for batch resource allocation problem. One of the non-optimal heuristic repeatedly computes a least cost allocation for every task of a batch using min-cost max flow and then takes the multiset union of computed resource allocation for each task (note that multiset union $\mathbb{A} \cup \mathbb{B} = \{(x,i) | i = \max\{m_{\mathbb{A}}(x), m_{\mathbb{B}}(x)\}\}$). We denote it maximum feasibility algorithm (*MaxF*) and use it as the baseline to see how much improvement can be made.

III. An Approximation Algorithm

We start with defining a function f_R measuring how much resource requirements of a batch $R = \{R_1, \ldots, R_k\}$ can be satisfied by a resource multiset $\mathbb{S}$. Define the function $f_R : \mathcal{P}(\mathbb{B}) \rightarrow \mathbf{N}$ (where we use $\mathcal{P}(\mathbb{B})$ to denote the powerset consisting of all multisets that are subsets of $\mathbb{B}$) to be: $f_R(\mathbb{S}) = \Sigma_{i=1,\ldots,k} M(G_i(\mathbb{S}))$. That is, $f_R(\mathbb{S})$ is equal to the sum over all tasks of the maximum flow of $\mathbb{S}$ with respect to each task in R. The value of this function is compared with a constant F_R reflecting total requirements of R, defined as $F_R = \Sigma_{i=1,\ldots,k} \Sigma_{j=1,\ldots,m_i}\ x_{ij}$.snd (where $R_i = [x_{i1}, \ldots, x_{im_i}]$, $k = |R|$).

Theorem 3. *A batch R is satisfiable by a resource multiset $\mathbb{S}$ iff $f_R(\mathbb{S}) = F_R$.*

Proof: For each $R_i \in R$, the total capacity of the edges originating from the source s (i.e., edges in E_1) of its flow graph $G_i(\mathbb{S})$ (see Fig. 2) is $\Sigma_{j=1,\ldots,m_i}\ x_{ij}$.snd, which is the upper bound for its max flow. That is, $0 \leq M(G_i(\mathbb{S})) \leq \Sigma_{j=1,\ldots,m_i}\ x_{ij}$.snd. Therefore, if $f_R(\mathbb{S}) = F_R$, we have $M(G_i(\mathbb{S})) = \Sigma_{j=1,\ldots,m_i}\ x_{ij}$.snd for all i (since otherwise $f_R(\mathbb{S}) < F_R$). From Theorem 1, all tasks are satisfiable (sufficient condition). Necessary condition obviously holds from Theorem 1 and definition of $f_R(\mathbb{S})$ and F_R. ■

Besides checking satisfiability, the function f_R also has the property of being submodular (see Sec. IV). Define the factor $\alpha = \max_{b \in \mathbb{B}} |\{i \mid b \in \cup_{j=1,\ldots,m_i} x_{ij}.\text{fst}\}|$. Intuitively, α is the maximum number of tasks in R which any given resource is mentioned. It is proved in Sec. IV that the solution produced by the algorithm below is worse than the optimal solution by a factor of at most $\ln \alpha$. Clearly, α is at most the number of tasks k but could be much less if resource requirements of the tasks are sufficiently different.

Algorithm 1 An Approximation Algorithm for Resource Allocation

input Batch allocation problem instance $(\mathbb{B}, R, c)$
output Resource allocation multiset $\mathbb{S}$
1: **procedure** APPROX-RESALLOC($\mathbb{B}$, R)
2: $F_R \leftarrow \Sigma_{i=1,\ldots,k} \Sigma_{j=1,\ldots,m_i}\ a_{ij}$
▷ where $k = |R|$, $R_i = [x_{i1}, \ldots, x_{im_i}]$, $a_{ij} = x_{ij}$.snd
3: $\mathbb{S} \leftarrow \emptyset$
4: **while** $(\mathbb{B} - \mathbb{S} \neq \emptyset)$ **do**
5: $\mathrm{q} \leftarrow \arg\min_{b \in \mathbb{B} - \mathbb{S}} c(b)/(f_R(\mathbb{S} \uplus \{b\}) - f_R(\mathbb{S}))$
6: $\mathbb{S} \leftarrow \mathbb{S} \uplus \{\mathrm{q}\}$
7: **if** $(f_R(\mathbb{S}) \geq F_R)$ **then return** $\mathbb{S}$ **end if**
8: **end while**
9: **return** $\emptyset$ ▷ Resource requirements cannot be satisfied
10: **end procedure**

The algorithm is detailed in Algorithm 1. The algorithm starts with allocation multiset $\mathbb{S} = \emptyset$ (the empty set). At each step, it picks an element $q \in \mathbb{B} - \mathbb{S}$ for which the value of $c(q)/(f_R(\mathbb{S} \uplus \{q\}) - f_R(\mathbb{S}))$ is the minimum among all elements in $\mathbb{B} - \mathbb{S}$, and add q into $\mathbb{S}$ (i.e., multiset sum $\mathbb{S} \uplus \{q\}$). The algorithm terminates and returns $\mathbb{S}$ when $f_R(\mathbb{S}) = F_R$.

This algorithm can be used by both enterprises and cloud providers to reduce cost or to allow a higher degree of resource sharing. An enterprise can simply substitute $\mathbb{B}$ with a hypothetical big-enough resource multiset and apply the algorithms with its internal resource requirements R to figure out how much resources it needs to reserve. At the cloud platform side, at certain situations (e.g., when resource allocations are about to reach the platform's full capacity), the cloud provider can use the algorithm to reduce reservation size of individual users, thus allowing more users to share the platform concurrently.

IV. Approximation Bound Analysis

In this section, we prove a worst case approximation bound on the cost of the solutions produced by our resource allocation algorithm Approx-ResAlloc (Algorithm 1). While the design of our algorithm was strongly guided by techniques for approximating the submodular set cover (SSC) problem, the existing results on SSC do not directly yield the algorithm or its approximation analysis. In the presence of multiplicities (in either the resource universe or resource requirements), the solution to the resource allocation problem is a multiset. On the other hand, the SSC problem is defined over sets

and its solutions are sets; it is therefore not immediately applicable to the resource allocation problem with multiplicities considered in this paper. Our approximation analysis is structured as follows. In Section IV-A, we define an extension of the submodular set cover problem to multisets that we call the submodular multiset cover problem (SMC). We present an algorithm for the general SMC problem and proof of its approximation analysis. In Section IV-B, we apply the results of Section IV-A to derive a worst case bound on the approximation ratio achieved by our resource allocation algorithm.

A. Submodular Multiset Cover

We begin by extending the notion of monotone, submodular functions to multisets.

Definition 1. *Let $\mathbb{B}$ be a multiset and $f : \mathcal{P}(\mathbb{B}) \to \boldsymbol{R}$ be a real-valued function defined on all multisets that are subsets of $\mathbb{B}$.*

1) *The function f is* monotone *if for all multisets $\mathbb{X}, \mathbb{Y} \subseteq \mathbb{B}$, we have that if $\mathbb{X} \subseteq \mathbb{Y}$ then $f(\mathbb{X}) \leq f(\mathbb{Y})$.*
2) *The function f is* submodular *if for all multisets $\mathbb{X}, \mathbb{Y} \subseteq \mathbb{B}$, we have that*

$$f(\mathbb{X}) + f(\mathbb{Y}) \geq f(\mathbb{X} \cup \mathbb{Y}) + f(\mathbb{X} \cap \mathbb{Y})$$

We define the submodular multiset cover problem for monotone, submodular functions equipped with a cost function on the underlying set.

Definition 2 (Submodular Multiset Cover)**.** *An* instance *of the* Submodular Multiset Cover (SMC) *problem is given by $(\mathbb{B}, f, c)$, where $\mathbb{B}$ is a multiset, $f : \mathcal{P}(\mathbb{B}) \to \boldsymbol{R}$ is monotone and submodular, and $c : B \to \boldsymbol{R}$ is a cost function on the underlying set of $\mathbb{B}$. With respect to such an instance $(\mathbb{B}, f, c)$ of the SMC problem, we define the following:*

- *A* feasible solution *is a multiset $\mathbb{S} \subseteq \mathbb{B}$ such that $f(\mathbb{S}) = f(\mathbb{B})$.*
- *The* cost of a feasible solution $\mathbb{S} \subseteq \mathbb{B}$ *is given by $c(\mathbb{S}) = \sum_{b \in B} m_{\mathbb{S}}(b)c(b)$.*
- *The* optimization problem *is to find the minimum value of $c(\mathbb{S})$ among all feasible solutions $\mathbb{S}$.*

Intuitively, the function f defines the coverage of any solution and a feasible solution is one that has the same coverage as the universe $\mathbb{B}$, *i.e.*, the most complete coverage possible. The SMC problem is then to find the smallest cost solution with the most coverage.

For any function $f : \mathcal{P}(\mathbb{B}) \to \mathbf{R}$ and multiset $\mathbb{S} \subseteq \mathbb{B}$, we define the *marginal value* function of f with respect to $\mathbb{S}$ as follows. Define the function $f_{\mathbb{S}} : \mathcal{P}((\mathbb{B} \setminus \mathbb{S})) \to \mathbf{R}$ by $f_{\mathbb{S}}(\mathbb{X}) = f(\mathbb{X} \uplus \mathbb{S}) - f(\mathbb{S})$. It is straightforward to show that if f is monotone and submodular on $\mathbb{B}$ then so is $f_{\mathbb{S}}$ on the multiset $\mathbb{B} \setminus \mathbb{S}$.

An approximation algorithm for SMC, presented in Algorithm 2, proceeds as follows. It starts with an empty $\mathbb{F}$ and repeatedly adds elements to it until it becomes a feasible solution. The element b picked in each iteration is the "greedy" one in the sense that b maximizes the coverage increase per cost among those not yet selected, *i.e.*, the one minimizing $c_j/(f(\mathbb{F} \cup \{b\}) - f(\mathbb{F})) = c_j/f_{\mathbb{F}}(\{b\})$.

Algorithm 2 An Approximation Algorithm for Submodular Multiset Cover

input An SMC instance $(\mathbb{B}, f, c)$
output A multiset $\mathbb{F}$ that is a feasible solution
1: **procedure** APPROX-SMC($\mathbb{B}$, f, c)
2: $\quad\mathbb{F} \leftarrow \emptyset$
3: $\quad$**while** $f(\mathbb{F}) < f(\mathbb{B})$, *i.e.*, $\mathbb{F}$ not feasible **do**
4: $\quad\quad$Add $\arg\min_{b \in \mathbb{B} \setminus \mathbb{F}} c_j/f_{\mathbb{F}}(\{b\})$ to $\mathbb{F}$.
5: $\quad$**end while**
6: **end procedure**

Our approximation analysis of the algorithm approx-smc is based on formulating a suitable integer linear program IP-SMC such that the cost of the solution produced by procedure approx-smc in Algorithm 2 can be bounded with respect to the objective function of the dual program to IP-SMC. To this end, the following theorem establishes the requisite integer programming formulation of SMC.

Theorem 4. *For any SMC instance $(\mathbb{B}, f, c)$, define the integer linear program IP-SMC as follows:*

$$\textit{Minimize } \textstyle\sum_{b \in B} c(b) \cdot x_b \quad \textit{subject to}$$

$$\textstyle\sum_{b \in \mathbb{B} \setminus \mathbb{S}} f_{\mathbb{S}}(\{b\}) \cdot x_b \geq f_{\mathbb{S}}(\mathbb{B} \setminus \mathbb{S}) \qquad \forall \mathbb{S} \subseteq \mathbb{B} \quad (1)$$

$$x_b \in \boldsymbol{N} \qquad \forall b \in B \quad (2)$$

Then for any feasible solution $\mathbb{T}$ to the SMC instance, we have that the assignment $x_b = m_{\mathbb{T}}(b)$ is a feasible solution for IP-SMC. Therefore, the optimum value for the SMC instance is lower bounded by that of IP-SMC.

Proof: Let $\mathbb{T}$ be any feasible solution. Then for x_b as defined above, we have that it satisfies Constraint (1) of IP-SMC as follows, for any $\mathbb{S} \subseteq \mathbb{B}$:

$$\begin{aligned}
\sum_{b \in \mathbb{B} \setminus \mathbb{S}} f_{\mathbb{S}}(\{b\})x_b &\geq \sum_{b \in \mathbb{T} \setminus \mathbb{S}} f_{\mathbb{S}}(\{b\})x_b && (3)\\
&\geq \sum_{b \in \mathbb{T} \setminus \mathbb{S}} f_{\mathbb{S}}(\{b\})m_{(\mathbb{T} \setminus \mathbb{S})}(b) && (4)\\
&\geq f_{\mathbb{S}}(\mathbb{T} \setminus \mathbb{S}) && (5)\\
&= f(\mathbb{T}) - f(\mathbb{S})\\
&= f(\mathbb{B}) - f(\mathbb{S}) = f_{\mathbb{S}}(\mathbb{B} \setminus \mathbb{S}) && (6)
\end{aligned}$$

where inequality (3) is due to $\mathbb{T} \subseteq \mathbb{B}$, inequality (4) is because $x_b = m_{\mathbb{T}}(b) \geq m_{(\mathbb{T} \setminus \mathbb{S})}(b)$, inequality (5) is because $f_{\mathbb{S}}$ is submodular, and equality (6) is because $f(\mathbb{T}) = f(\mathbb{B})$ (since $\mathbb{T}$ is a feasible solution for the SMC instance). ■

Define the linear program relaxation of IP-SMC by replacing the integral constraint (2) by the inequality $x_b \geq 0$. The dual of this linear program yields the dual program for SMC

that we call DP-SMC and that is as follows:

$$\text{Maximize} \sum_{\mathbb{S}\subseteq\mathbb{B}} f_{\mathbb{S}}(\mathbb{B}\setminus\mathbb{S})\cdot y_{\mathbb{S}} \quad \text{subject to}$$

$$\begin{aligned} \sum_{\mathbb{S}: b\notin\mathbb{S}} f_{\mathbb{S}}(\{b\})\cdot y_{\mathbb{S}} &\le c(b) \quad \forall b\in B \\ y_{\mathbb{S}} &\ge 0 \quad \forall \mathbb{S}\subseteq\mathbb{B} \end{aligned}$$

For any natural number $n \geq 1$, recall that the n'th harmonic number $H(n)$ is defined as $H(n) = \sum_{1\leq i\leq n} 1/i$. The following theorem establishes an upper bound on the ratio of the cost of the solution produced by approx-smc with respect to the optimal solution. The proof proceeds by constructing a feasible solution to the dual program DP-SMC based on the result produced by approx-smc. By analyzing the cost of the objective function in DP-SMC for this feasible solution, we then obtain the approximation bound.

Theorem 5. *Let $(\mathbb{B}, f, c)$ be an instance of SMC and $\mathbb{F}$ be the solution returned by approx-smc. If $f(\emptyset) = 0$, we have that $c(\mathbb{F}) \leq OPT * H(\max_{b\in B} f(\{b\})$ where OPT is the cost of the optimal SMC solution for the instance $(\mathbb{B}, f, c)$.*

Proof: (Sketch) Having defined IP-SMC suitably, the proof is very similar to that for set cover [6]. Let $\mathbb{F}^i$ denote the value of $\mathbb{F}$ at the i'th iteration of the while loop in the procedure approx-smc and suppose that it terminates after t iterations so that $\mathbb{F}^0 = \emptyset$ and $\mathbb{F}^t$ is the solution returned. Define the value $\theta^i = \min_{b\in\mathbb{B}\setminus\mathbb{F}^{i-1}}(c(b)/f_{\mathbb{F}^{i-1}}(\{b\}))$. Then, we can show that, for any $b \in B$,

$$\begin{aligned} & \theta^1 f_{\mathbb{F}^0}(\{b\}) + \cdots + (\theta^t - \theta^{t-1}) f_{\mathbb{F}^{t-1}}(\{b\}) \\ \leq\ & (\max_{1\leq i\leq t} \theta^i f_{\mathbb{F}^{i-1}}(\{b\}))H(f(\{b\})) \\ \leq\ & c(b)H(f(\{b\})) \qquad (7) \\ \leq\ & c(b)H(\max_{b\in B} f(\{b\}) \end{aligned}$$

where the inequality (7) follows from $\theta^i f_{\mathbb{F}^{i-1}}(\{b\}) \leq c(b)$ (for all $b \in B$) due to the greedy choice made at each iteration. Define $y_{\mathbb{F}^0} = \theta^1/H(\max_{b\in B} f(\{b\})$, $y_{\mathbb{F}^i} = (\theta^{i+1} - \theta^i)/H(\max_{b\in B} f(\{b\})$ for $i = 1, \ldots, t-1$ and $y_{\mathbb{S}} = 0$ for all other multisets $\mathbb{S} \subseteq \mathbb{B}$. By the above inequality it follows that these assignments to y defines a feasible solution to DP-SMC. The cost of the solution $\mathbb{F}^t$ can then be rewritten in terms of this assignment to y as:

$$c(\mathbb{F}^t) = H(\max_{b\in B} f(\{b\}) * \sum_{\mathbb{S}\subseteq\mathbb{B}} f_{\mathbb{S}}(\mathbb{B}\setminus\mathbb{S}) y_{\mathbb{S}}$$

It therefore follows that the cost of the solution $\mathbb{F}^t$ is at most $H(\max_{b\in B} f(\{b\})$ times the objective value in DP-SMC for y. The statement of the theorem then follows from Theorem 4 and the weak duality theorem (that the objective value in DP-SMC of any feasible solution is a lower bound on the objective value of any feasible solution to IP-SMC). ■

B. Resource Allocation as Submodular Multiset Cover

We cast the resource allocation problem as an instance of the submodular multiset cover problem. To this end, we first restate a classical result on network flows in terms of multiset functions. For a network $G = (V, E)$, two edges $e_1, e_2 \in E$ are said to be *parallel* if every simple cycle containing both of them has to orient them in opposite directions and a set of arcs is *parallel* if it consists of pairwise parallel arcs. Let P be a parallel arc set. With respect to any fixed capacity assignments to the edges in $E \setminus P$, we define the following function F on multisets whose underlying set is P. For any multiset $\mathbb{S}$ whose underlying set is P, define the flow network $G_{\mathbb{S}}$ on G by assigning capacity $p(e) = m_{\mathbb{S}}(e)$ to any edge $e \in P$ (with capacities on other edges already fixed) and define $F(\mathbb{S}) = M(G_{\mathbb{S}})$ the max flow on $G_{\mathbb{S}}$. Then, the following proposition is a straightforward consequence of the results of [7].

Proposition 1 ([7]). *Let $G = (V, E)$ and $P \subseteq E$ be any parallel arc set. Then, with respect to any fixed non-negative capacity assignment to the edges in $E\setminus P$, the function $F(\mathbb{S}) = M(G_{\mathbb{S}})$ is submodular.*

The following theorem establishes the necessary correspondence between the resource allocation problem and the multiset submodular cover problem, where f_R is the function definition used in the algorithm Approx-ResAlloc.

Theorem 6. *Let $\mathcal{A} = (\mathbb{B}, R, c)$ be any instance of the resource allocation problem. Then $\mathcal{C} = (\mathbb{B}, f_R, c)$ is an instance of the SMC problem. If the resource allocation instance $\mathcal{A}$ admits any feasible solutions, then a multiset $\mathbb{S} \subseteq \mathbb{B}$ is a feasible solution for $\mathcal{A}$ iff it is a feasible solution for $\mathcal{C}$ with identical costs as instances of both problems.*

Proof: To show that $\mathcal{C}$ is an instance of SMC, we prove that f_R is monotone and submodular. Recall that $f_R = \sum_{i=1,\ldots,k} f_i$ where $f_i(\mathbb{S}) = M(G_i(\mathbb{S}))$. Since the sum of monotone, submodular functions is also monotone and submodular it suffices to show the property for each of the functions f_i. Monotonicity is straightforward, and for submodularity, we can see that the arc set $E_3 = S \times \{t\}$ in the graph $G_i(\mathbb{S})$ is a parallel arc set. It therefore follows from Proposition 1 that each f_i is submodular. If $\mathcal{A}$ admits a feasible solution, we have that $f_R(\mathbb{B}) = F$ for $F = \sum_{i=1,\ldots,k}\sum_{j=1,\ldots,m_i} a_{ij}$. Then, for any $\mathbb{S} \subseteq \mathbb{B}$, $\mathbb{S}$ is a feasible solution for $\mathcal{A}$ iff $f_R(\mathbb{S}) = F = f_R(\mathbb{B})$ iff $\mathbb{S}$ is feasible for the SMC problem $\mathcal{C}$. ■

For any instance of the resource allocation problem, $(\mathbb{B}, \{R_1, \ldots, R_k\}, c)$ and any resource $b \in \mathbb{B}$, define the set $N_b = \{i \mid b \in \cup_{j=1,\ldots,m_i} x_{ij}.\text{fst}\}$ which includes the subset of tasks for which b be can be used in some slot. Define the factor $\alpha = \max_{b\in\mathbb{B}} |N_b|$. For the function f_R used in the SMC correspondence Theorem 6, it is easy to see that $\alpha = \max_{b\in B} f_R(\{b\})$. Since $H(k) \leq 1 + \ln k$ for any k, from Theorems 6 and 5, we can therefore establish that the solution returned by the algorithm Approx-ResAlloc is worse than the optimal solution by a factor of at most $\ln \alpha$.

Corollary 1. *Let $(\mathbb{B}, R, c)$ be an instance of the resource allocation problem and $\mathbb{S}$ be the solution returned by the algorithm Approx-ResAlloc. We have that $c(\mathbb{S}) \leq OPT*(1+\ln\alpha)$ where OPT is the cost of the optimal resource allocation.*

V. Empirical Evaluation

In this section, we evaluate the performance of the proposed algorithms and quality of the produced allocations using simulation. We first examine the algorithms in terms of how small the resulting allocation can be (i.e., having the cost fixed to 1 for all resources). Then we compare how much cost they can save by using a tiered cost model similar to Amazon EC2 (see Table I).

The simulation consists of many runs for each different setting. We vary the settings in terms of the number of tasks (ranging from 10 to 100) in a problem instance, number of slots in a task (ranging from 5 to 10), number of resource copies requested by a slot (ranging from 1 to 10). We use a resource universe containing 50 different resource configurations, with 1000 copies for each configuration. These variations are chosen to compare the algorithms relative to each other, rather than emulating situations in reality.

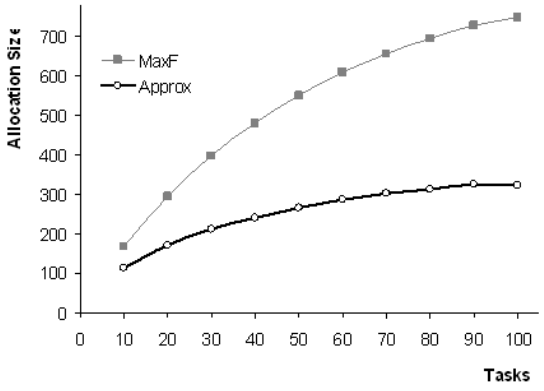

Fig. 4. Comparing size of allocation

Fig. 4 shows the size of the allocation yielded by the algorithms as the number of tasks in a batch (therefore the total amount of resource requests) increases. The base line algorithm MaxF allocates far larger number of resources whereas the approximation algorithm Approx reduces the allocation size by more than 50% of the base line.

Fig. 5 depicts the cost of the allocations produced by the algorithms. The general shapes of the curves are similar to Fig. 4. In this case, the approximation algorithm yields a reduction in cost around 66%.

VI. Related Work

Resource allocation problem has been an important topic in the fields of networking [8], [9], parallel [10] and distributed computing [11], [12]. Most research has focused on the utilization of multiple processors efficiently or minimization of completion time of tasks [13]. For instance, the problem of allocating distributed resources is studied in [14] with a goal of minimizing average response time. Intuitively it wants to use as many available resources and as efficiently as possible. This paper adopts a different perspective, and instead wants to

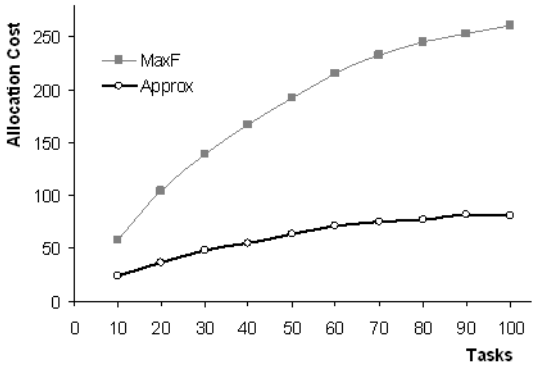

Fig. 5. Comparing cost of allocation

use as few resources as possible in order to reduce cost and administration overhead for setting up the resources. Though many researchers have investigated scheduling problems with explicit setup times [15], this paper only looks at the cost (or size) of a reservation without taking into consideration costs for setting up the resources after reservation is made.

Reserving many resources makes it easy for users to manage needed resources collectively. For instance, CloudNet [16] proposes to build virtual private networks/clouds from machines inside enterprises and allocated from public clouds. For high-demanding parallel computations users can reserve resources with certain characteristics (e.g., machines with more than 1GB memory) to build a cluster [17], [18]. This notion of clusters on demand [19], [20] has developed from using physical machines to using virtual machines. Nimbus [21] (part of the Globus [12] Alliance) and Shirako [22] are two such systems that allow users to build a virtual cluster on Amazon EC2 cloud platform.

Shirako [22] and Globus's leasing-based architecture Haizea [23] are close to our work in that both of them study resource reservations from cloud platforms. Shirako [22] enables applications to reserve groups of resources (e.g., physical machines and with specified processor and memory attributes) using leases and tickets across multiple autonomous sites. Haizea uses leases to support best-effort and advanced reservations for allocation of both hardware resources and software environments on cloud platforms. In both cases, leases bind a set of resources from a platform to a request from some time interval, corresponding to requirements from a single task in this paper. Both works focus on the system aspects of building an infrastructure that enables allocations to individual leases in clouds. In contrast, this paper concentrates on the algorithmic aspect of resource reservation, where the challenges lie in optimization across multiple tasks/leases.

VII. Summary

Cloud computing enable enterprises to reserve a group of resources and use them to establish common platforms (e.g., VPNs, clusters) to run computation tasks. This paper

investigate the issue of deciding how many and what kinds of resources need to be reserved for these computations after the computations have been linearized into the form of a sequence of tasks. We show that it is NP-Complete to minimize the cost/size of the reservation. An approximation algorithm with bounded ratio is proposed and its performance studied using simulation. This algorithm can be used by both enterprises and cloud providers to reduce cost or to allow a higher degree of resource sharing.

REFERENCES

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proc. of 19th ACM symposium on Operating systems principles*, 2003.

[2] J. Fisher-Ogden, *Hardware support for efficient virtualization*, http://cseweb.ucsd.edu/~jfisherogden/hardwareVirt.pdf, April 2006.

[3] F. Chang, J. Ren, and R. Viswanathan, "Optimal resource allocation for batch testing," in *ICST '09: Proc. of 2009 IEEE International Conference on Software Testing Verification and Validation*, 2009.

[4] A. V. Goldberg and R. E. Tarjan, "Finding minimum-cost circulations by canceling negative cycles," *J. ACM*, vol. 36, no. 4, pp. 873–886, 1989.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[6] L. A. Wolsey, "An analysis of the greedy algorithm for the submodular set covering problem," *Combinatorica*, vol. 2, no. 4, pp. 385–393, 1982.

[7] D. Gale and T. Politof, "Substitutes and complements in network flow problems," *Discrete Appl. Math.*, vol. 3, pp. 175–186, 1981.

[8] B. N. Chun, D. E. Culler, T. Roscoe, A. C. Bavier, L. L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

[9] C. Dumitrescu, I. Raicu, M. Ripeanu, and I. T. Foster, "Diperf: An automated distributed performance testing framework," in *GRID*, 2004.

[10] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *In Job Scheduling Strategies for Parallel Processing*. Springer Verlag, 1997.

[11] A. Baratloo, P. Dasgupta, and Z. M. Kedem, "Calypso: A novel software system for fault-tolerant parallel processing on distributed platforms," in *HPDC*, 1995, pp. 122–129.

[12] I. T. Foster, "The globus toolkit for grid computing," in *CCGRID*. IEEE Computer Society, 2001, p. 2.

[13] M.Drozdowski, "Scheduling multiprocessor tasks - an overview," *European Journal of Operational Research*, vol. 94, pp. 215–230, 1996.

[14] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir, "On chromatic sums and distributed resource allocation," *Information and Computation*, vol. 140, no. 2, pp. 183–202, 1998.

[15] A. Allahverdi, C. T. Ng, T. C. E. Cheng, and M. Y. Kovalyov, "A survey of scheduling problems with setup times or costs," *European Journal of Operational Research*, vol. 187, no. 3, pp. 985–1032, 2008.

[16] T. Wood, A. Gerber, K. Ramakrishnan, and J. van der Merwe, "The case for enterprise-ready virtual private clouds," in *Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2009.

[17] M. Aron, P. Druschel, and W. Zwaenepoel, "Cluster reserves: A mechanism for resource management in cluster-based network servers," in *Proceedings of the ACM Sigmetrics 2000*, June 2000.

[18] I. T. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy, "A distributed resource management architecture that supports advance reservations and co-allocation," in *IWQoS '99*, 1999, pp. 27–36.

[19] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. Sprenkle, "Dynamic virtual clusters in a grid site manager," in *HPDC*. IEEE Computer Society, 2003, pp. 90–103.

[20] E. Walker, J. Gardner, V. Litvin, and E. Turner, "Creating personal adaptive clusters for managing scientific jobs in a distributed computing environment," in *proceedings of Challenges of Large Applications in Distributed Environments (CLADE 2006)*, 2006, pp. 95–103.

[21] K. Keahey and T. Freeman, "Contextualization: Providing one-click virtual clusters," *eScience, IEEE International Conference on*, 2008.

[22] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G. Yocum, "Sharing networked resources with brokered leases," in *Proc. of the annual conference on USENIX '06 Annual Technical Conference*. USENIX Association, 2006.

[23] B. Sotomayor, K. Keahey, and I. Foster, "Combining batch execution and leasing using virtual machines," in *HPDC '08: Proc. of 17th international symposium on High performance distributed computing*, 2008.