Scaling Optimal Allocation of Cloud Resources Using Lagrange Relaxation

Luis de la Torre[1] and Mahantesh Halappanavar[2,1]

[1] Washington State University, Pullman WA, USA
[2] Pacific Northwest National Laboratory, Richland WA, USA
luis.delatorre@wsu.edu, hala@pnnl.gov

Abstract. The rapid growth of Cloud Computing (CC) has increased the variety of computing resources, storage, and communication services that pose significant new challenges for the efficient use of cloud resources. The cost-efficient allocation of cloud resources has become a decisive premise for the adoption of CC services. The cost-efficient selection and scheduling of these resources to meet the demands of a scientific workflow is a challenging problem that is exacerbated by the inclusion of multiple CC providers. In this paper, we present a novel strategy for the cost-efficient selection of CC resources using Lagrange relaxation. Our approach is based on preselection of resources and demand decomposition to create a series of smaller sub-problems, which allow the estimation of the best cost-structures and selection of CC service providers for a subset of the time period of the planning horizon. Decomposition of the demand is achieved through the boundary analysis of a continuous relaxation of the problem. Using the metrics defined in terms of the cost and time of completion, we demonstrate excellent performance in relation to optimal solutions. Our method reduced the computational time from hours to seconds for a representative 36-month problem and provided high-quality solutions ($< 0.05\%$ relative error). Given the importance of selecting resources and scheduling complex scientific workflows, we believe that this novel strategy will be beneficial for many researchers and users of cloud computing resources.

Keywords: Cloud Computing · Cost-efficient Selection · Cost Model

1 Introduction

The computational demands of modern applications have increased considerably [14], especially for data-intensive applications such as high-energy physics workflows that are conventionally executed on dedicated computing resources. However, these applications are increasingly being executed on geographically distributed opportunistic resources such as Cloud Computing Resources (CCR) [9]. Such special configurations pose a tremendous challenge for efficiently using computing resources. To reduce the total computing cost, it is necessary to implement energy-efficient power management and techniques to allocate tasks to appropriate computing resources optimally. Further, the operating overhead cost of

traditional dedicated resources is a significant fraction of the total computational cost [3]. Consequently, opportunistic resources provide a cost-efficient alternative with reduced infrastructure costs and lower energy consumption while providing compute and storage capacity on demand. While there are several CCR providers, we limit our study to two main providers as representative examples: Amazon AWS and Google Cloud. These providers have different cost models based on consumption. Example models include On-Demand, Savings Plans, Reserved Instances (No Upfront, Partial Upfront, Total Upfront), Sustained Usage, and Spot Instances. Further, the computing power for each resource is estimated by a specific metric, for example, Elastic Compute Unit (ECU) for Amazon AWS and Google Compute Engine Unit (GCEU) for Google Cloud.

Although there are many research papers on cloud resource allocation [10,15], there is a dearth of work on the analytical comparison of different cost models that exist for CCR. Further, existing studies do not address forecasted demands that are longer than one year in duration. In this paper, *we present a computationally efficient approximation algorithm using Lagrange relaxation for optimal resource allocation on the cloud.* Our study is motivated by the complex workflow from the Belle II experiments, a high-energy physics experiment designed to probe the interactions of the most fundamental constituents of our universe. Belle II is expected to generate about 25 petabytes (25×10^{15} bytes) of raw data per year with expected total storage to reach over 350 petabytes [1,7]. The data is generated from the physical experiment through the Belle II detector, from Monte Carlo simulations, and user analysis. This data is processed and reprocessed through a complex set of operations, which is followed by analysis in a collaborative manner. Users, data, storage, and compute resources are geographically distributed across the globe, offering a complex data-intensive workflow as a case study, and providing an excellent case study to test the efficacy of the proposed approach.

One of the key problems in these cloud computing scenarios is the selection of cloud resources that can meet the predicted demand in a cost-efficient manner for a given period of time. In order to solve this problem in an efficient manner, the analogy of the *unit commitment* problem in electric power grids (§3) has been used. This analogy was shown to be successful in formulating and solving the resource selection problem using a mixed-integer linear program (MILP) [5], and later using genetic algorithms [2]. Although successful, these methods do not scale for larger problem sizes. Therefore, our goal in this paper is to scale this approach using *Lagrange relaxation* (§4). We employ Lagrange relaxation to explore the solution space and create a list of subintervals of the planning horizon that can describe the entire space and produce a series of small subproblems with a reduced number of resources. We detail the mathematical formulation and the boundary analysis in §4. Boundary condition analysis enables the computation of *normalized average cost value*, which is a good estimator for the specific average cost for each resource in terms of the period of its utilization. We provide an illustrative case study based on a carefully designed synthetic workflow inspired by the Belle II experiment (§6). We demonstrate the proximity and quality of

the proposed solution by comparing with optimal solutions that are computed using integer linear programming. This method, on average, is almost negligibly higher (in cost) than the optimal solutions, and the completion time significantly reduces from hours to seconds to compute a solution for an allocation problem spanning a 36-month period.

Contributions:

We make the following key contributions:

- Building on the unit commitment framework of Halappanavar *et al.* [6], we present a novel Integer Linear Program formulation for efficient allocation of cloud resources (§3);
- Present a Lagrange relaxation-based approach to scale computation (§4), and a novel strategy and approximation algorithms based on a low boundary approach to decompose the forecast demand into a set of smaller sub-demand problems (§5and §6); and
- Using a representative problem, we demonstrate performance speedups up to **9**× for 16-machine configuration and **1412**× for 162-machine configuration, relative to the optimal solution – from hours to seconds. We show that the relative error is negligible, less than 0.05% for all cases (§7).

To the best of our knowledge, this is the first effort to analyze the Lagrange relaxation to scale cloud resource allocation problem in complex scientific workflows formulated as a unit commitment problem.

2 Background: Cloud Computing Model

In the context of large-scale scientific workflows, the demand for computation and storage varies significantly over time. The demand needs to be satisfied from a heterogeneous set of resources that comprise dedicated and shared HPC resources, and cloud services from several commercial and non-commercial vendors. Effective solutions to identify the cost-efficient subset of resources to meet a forecasted demand should consider the operating costs emerging from computation, data movement, and storage. However, current solutions, such as DIRAC in the case of Belle II [5], only consider computational aspects of available resources. The demand for resources can be roughly estimated through historical data and performance modeling approaches. For example, several Monte Carlo campaigns are conducted in Belle II with the predicted number of events to be simulated over a given period of time (P_d). The forecasted demand can be satisfied through a combination of dedicated HPC resources and cloud computing resources.

Two broad categories of cost models for cloud computing are: (i) on-demand subscription, and (ii) reservation-based subscription. We briefly describe these two models in the following discussion.

On-demand: The *"pay-as-you-go"* model is the simplest pricing model. This

approach is designed to pay just for the machines that are in use. In other words, the customer pays only for the resources used during a specific period of time and is supported by many providers including Amazon AWS, Google Cloud, and Windows Azure. The model is better suited for situations with short bursts of infrequent and unpredictable use. As an exception, Google Cloud offers special discounts for *sustained usage* with up to 30% discounts for workloads that run for a significant portion of a billing month. Further, *"spot pricing"* is a particular on-demand model, where the price varies according to the demand and supply for resources at a given point in time.

Reservation: Another common pricing model is "subscription" or "committed use discounts." In this model, a customer pays or commits to use specific resources in advance for a predefined period of time, independent of the actual use of the resources. Amazon AWS is one of the leading companies using this model. A few variants of the subscription model are as follows:

- *No Upfront* model is designed to reserve resources for a fixed period of time without a subscription cost but full payment for the amount of resources used. This model is only available for Amazon accounts with more than a year of good performance.
- *Partial Upfront* model enables reservation in advance for a fixed period of time with an initial cost and a discounted payment for the amount of resources used.
- *Total Upfront* model charges the customer a one-time fixed payment for using specific resources, usually in time units measured by years. This model provides the most cost-effective means for sustained yearlong use of specific resources.
- *Committed-Use Discounts* model is offered by Google Cloud services, where the user agrees to purchase a "committed-use contract" for a usage term of 1 year or 3 years. In this model, the user receives up to 57% discount on normal prices.

As an illustration, we present a sample of different cost plans as advertised by Amazon EC2 and Google Cloud in Table 1. The table lists only a small set of possible cost plans for three machine types: T2, C4, and n1-standard. The third column, **CPU**, provides an estimated value for the processing power of a given machine instance. The CPU values provided for the first four machines (marked by an asterisk) are estimated by a standard Linux benchmark "sysbench" running on 128 threads to find 50,000 prime numbers. The results are compared with regular Amazon ECU values using the following equation: ECU=$1925/t$, where t is the benchmark makespan time. The fourth column (**Tm**) represents the subscription time for the machine under a specific subscription plan. The fifth column (**s**) represents the fixed subscription cost (in dollars) for the period specified in the fourth column. The sixth column (**c**) represents the monthly usage cost (in dollars per month).

3 Problem Formulation

A general problem in cloud computing that arises for many users can be expressed as: *Given an estimated (or forecasted) demand for computing and storage resources for a given period of time, what is the specific subset (configuration) of cloud resources that can meet the predicted demand at a minimum cost.* We consider the problem in a batched context, where we bundle the demand for a given time period to make the decisions. However, the time period can be relaxed accordingly. In order to solve this problem in an efficient manner, we use the analogy of the *unit commitment problem* in electric power grids, proposed by Halappanavar *et al* [6], where a set of power generators are determined to meet the predicted demand for a certain period of time. At present, cloud computing platforms do not provide any tools for optimal selection of resources to satisfy the given computational demand over a certain period of time. However, such a tool becomes necessary for heterogeneous high-performance computing and cloud-based computational resources with varying degrees of usage costs. Consequently, the question that needs to be answered is: *Given an estimated (forecasted) demand, what is the most cost-efficient allocation of resources?* This question becomes especially important when cloud computing services are used as the primary source of resources. In some cases, this question also enables decisions for choosing a particular subset of cloud computing services.

3.1 Unit Commitment Problem

The unit commitment problem in the context of resource allocation in cloud computing can be stated as: Given a set of computing resources with different subscriptions and usage costs and forecasted demand, the objective of the solution is to determine the *schedule* to identify the set of computing resources that must be reserved and the amount of each resource needed to meet the forecasted demand at a minimum cost. In order to address this problem, we use a Lagrange relaxation based approach (described in §4) inspired from the unit commitment problem [13]. A key difference in our approach is the use of comparative analysis of different computing cost models to solve the problem.

Building on the notation of Brittany [17], the unit commitment problem in electric power grids can be formally expressed as follows:

$$\min_{P,u,x} F = \sum_{t=1}^{T} \sum_{j=1}^{N} [C_j(P_j(t)) + S_j(x_j(t), u_j(t))] \tag{1}$$

$$s.t. \sum_{j=1}^{N} P_j(t) = P_d(t) \qquad \forall t = 1, ..., T \tag{2}$$

$$\sum_{j=1}^{N} r_j(x_j(t), P_j(t)) \geq P_r(t) \ \forall t = 1, ..., T \tag{3}$$

where F is the total system cost for N power generators over T time units with operating (fuel) cost function C_j and start-up cost function S_j to generate P_j units of power. The variable x_j represents the number of time units (for example, hours) that a given generator is on (positive) or off (negative). Similarly, the variable u_j represents the state of a generator at a given time unit $t+1$. It is positive $(+1)$ if the state is up and negative (-1) if the state is down. The constraints enforce that the demand, $P_d(t)$ at time t, is satisfied. Further, the system is also required to meet a certain additional (reserve) unanticipated demand P_r, and r_j is the reserve function available from generator j for the time unit t. The total time period under consideration is T. The complete set of restrictions is presented in [17].

The Cloud Computing problem (from the user perspective) has several similarities with this problem. However, it is necessary to carefully define the meaning of the functions $C_j(P_j(t))$ and $S_j(x_j(t), u_j(t))$ in the context of cloud computing, which we present in the following discussion.

3.2 Cloud Computing Model

In the context of large-scale scientific workflows, the demand for computation and storage varies significantly over time. Effective solutions to identify the cost-efficient subset of resources to meet the forecasted demand should consider the operating costs emerging from computation (for example, floating-point operations), data movement (for example, the number of bytes transferred on the network), and storage (for example, the number of bytes stored on disk and tape). However, current solutions, such as DIRAC in the case of Belle II [5], consider only the computational aspects of available resources (through a pull model). The demand for resources can be estimated through planned experiments. For example, several Monte Carlo campaigns are conducted in Belle II with the predicted number of events to be simulated over a given period of time (P_d). The two broad cost models for cloud computing are: (i) on-demand subscription, and (ii) reservation-based subscription. We briefly describe these two models in the following discussion

On-demand: The "pay-as-you-go" is the simplest pricing model. This method is designed to pay just for the machines that are in use. This model is supported by many providers including Amazon AWS, Google Cloud, and Windows Azure. While popular, on average, this cost model is expensive and in many cases is better suited only for situations with short bursts of use. Further, *Spot Price* is a particular on-demand model, where the price varies according to the demand and supply for resources.

Reservation: Another common pricing model is "subscription" or "committed use discounts." In this model, a customer pays or commits to use specific resources in advance for a predefined period. Amazon AWS is one of the leading companies using this model.Variants of the subscription model are as follows:

- *No Upfront* model is designed to reserve resources for a fixed period of time without a subscription cost but full payment for the amount of resources used. accounts with more than a year of good performance.
- *Partial Upfront* model enables reservation in advance for a fixed period of time with an initial cost and a discounted payment for the amount of resources used.
- *Total Upfront* model charges the customer a one-time fixed payment for using specific resources, usually in time units measured by years. This model provides the most cost-effective means for sustained yearlong use of specific resources.
- *Committed-Use Discounts* model is offered by Google Cloud services, where the user agrees to purchase a "committed-use contract" for a usage term of 1 year or 3 years. In this model the user receives up to 57% discount no normal prices.

Cost plan	**Machine type**	**CPU**	**Tm** (m)	**s** ($)	**c** ($/month)
Lower-Price	n1 standard 1	3.3*	1	0	24.27
Lower-Price	n1 standard 2	6.6*	1	0	48.54
Lower-Price	n1 highcpu 2	6.7*	1	0	36.20
Lower-Price	n1 highcpu 4	13.4*	1	0	72.41
No-Upfront	T2 Nano	0.2	12	0	2.63
No-Upfront	T2 Micro	0.4	12	0	5.26
No-Upfront	T2 Small	0.8	12	0	13.14
P-U 3 year	C4 Large	8	36	539	15.33
T-U 3 year	C4 Large	8	36	1013	0
P-U 3 year	C4 Xlarge	16	36	1091	30.66
T-U 3 year	C4 Xlarge	16	36	2050	0
P-U 3 year	C4 2XLarge	31	36	2168	59.86

Table 1: Representative subscription costs for Amazon EC2 and Google Cloud resources. This information was accessed on 5/10/2022. *Values estimated based on "sysbench" benchmark and the CPU burst minutes, the price includes the monthly use discount.

As an illustration, we present a sample of different cost plans as advertised by Amazon EC2 and Google Cloud in Table 1. The table lists only a small set of possible cost plans for three machines type: T2, C4, and n1-standard. The third column, **CPU**, provides an estimated value for the processing power of a given machine instance. The **CPU** values provided for the first four machines (marked by an asterisk) were estimated by a standard Linux benchmark "sysbench" running on 128 threads to find 50,000 prime numbers. The results are compared with regular Amazon ECU values using the following equation: ECU=$1925/t$, where t is the benchmark makespan time. The fourth column **Tm** represents the subscription time for the machine under a specific subscription plan. The fifth column, **s**, represents the fixed subscription cost (in dollars) for the period specified in the fourth column. The sixth column, **c**, represents the monthly usage cost (in dollars per month).

3.3 Integer Linear Program Formulation

We now present an Integer Linear Program (ILP) formulation of the cloud computing resource (CCR) selection problem. The objective is to compute a minimum-cost allocation of resources to satisfy the forecasted demand using available resources. We define the following variables and parameters. Let F denote the total operating costs attributed to the use of N resources for a total of T time units, where T represents the total time period under consideration. Let $P_d(t)$ represent the predicted demand in CPU units and $P_j(t)$ represent the number of resources used of type j at time unit t. The function $S_j(u)$ represents the subscription cost for the use of resource u of type j, and CPU_j represents the number of standard computational units for a specific resource type j.

The variable $u_j(t)$ represents the number of subscriptions for resource type j at a given time unit t. Now, we define $C_j(x)$ and $S_j(u)$ as follows: let $C_j(p) = c_j p$ be the cost of using p units of machine type j, and $S_j(u) = s_j u$ be the cost to subscribe u number of machines of type j, where s_j is payed once for a continuous interval of time, Tm_j. In our formulation, the variable $u_j(t)$ is used as a non-negative integer variable representing the number of resources of a specific type that are subscribed to complete a specific demand, and P_j^{max} represents the maximum number of machines available for every type. Specific numbers can be computed similarly to the work of Singer *et al.* [15]. Our objective is to minimize the total subscription cost as well as the operating costs while satisfying the demand over the planning horizon. Notations used in this paper are summarized in Table 2. Adapting the unit commitment for cloud computing, we reformulate the power grid-related equations (1– 3) as follows.

$$\min_{u,P} F = \sum_{t=1}^{T} \sum_{j=1}^{N} c_j P_j(t) + s_j u_j(t) \tag{4}$$

$$s.t. \sum_{j=1}^{N} CPU_j P_j(t) \geq P_d(t) \qquad \forall t = 1, ..., T \tag{5}$$

$$\sum_{i=max\{1,t-Tm_j\}}^{t} u_j(i) \geq P_j(t), \quad \forall j = 1, ..., N, \ \forall t = 1, ..., T \tag{6}$$

$$0 \leq P_j(t) \leq P_j^{max} \qquad \forall j = 1, ..., N, \ \forall t = 1, ..., T \tag{7}$$

$$u_j(t) \geq 0 \qquad \forall j = 1, ..., N, \ \forall t = 1, ..., T \tag{8}$$

A potential approach to solve the cloud unit commitment problem is to assign integer values to variable $P_j(t)$, which results in its formulation as a *mixed-integer linear programming problem* (MILP). We develop strategies to relax the problem to enable the computation of lower-bound approximate solutions quickly. We then use the approximate solution to compute a feasible close-to-the-optimal integer solution in pseudo-polynomial time (§4).

Table 2: A summary of key notations

Notation	Description
	Sets
J	Set of machines, $j \in J$
	Parameters
N	No. of machines
P_d^t	Demand at period t
s_j	Subscription cost for machine j over Tm_j periods
c_j	Operating cost of machine j for a single period
T	Total planning horizon
$\lambda(t)$	Lagrange multipliers for every time unit t
CPU_j	Computing power of machine j
P_j^{max}	Maximum machines j available.
	Variables
$u_j(t)$	No. of machine j subscribed in the period t
$P_j(t)$	No. of machine of type j used at period t

4 Lagrange Relaxation

In this section, we present an approximate solution to the cloud unit commitment problem (§3) using Lagrange relaxation. Lagrange relaxation enables us to explore the solution space efficiently and create a series of smaller sub-problems that can be solved quickly. We start the discussion in this section on the boundary analysis and then proceed to discuss the normalized average cost analysis of the proposed solution.

4.1 Boundary Analysis

We formulate the relaxed variants of the resource selection problem where the demand constraint (Equation 5) is included in the objective function (Equation 4) using Lagrange multipliers $\lambda(t)$ for every time unit t. A new formulation of the objective function can be expressed as follows (Equation 9):

$$L(u,p,\lambda) = \sum_{t=1}^{T} \Big[\sum_{j=1}^{N} (c_j P_j(t) + s_j u_j(t)) + \lambda(t) \Big(P_d(t) - \sum_{j=1}^{N} CPU_j P_j(t) \Big) \Big]. \quad (9)$$

In this reformulation, the primal problem is converted into a new optimization problem where the penalty terms $\lambda(t)$ become a set of additional variables. The constraint 6 is not included in the new formulation since it is only a restriction on the number of machines used.

With the reformulation, the problem now becomes a *max-min* problem where the variables u and p are for the minimization and variables $\lambda(t)$ are for the maximization of the objective function. The problem can then be reformulated as:

$$\max_{\lambda \geq 0} \min_{u,p} L(u,p,\lambda) \quad (10)$$

subject to the remaining restrictions. In order to develop a lower boundary for the set of solutions, it is necessary to estimate particular values of the Lagrange multipliers $\lambda(t)$.

Analytical solution with fixed subscriptions.

We develop an analytical solution for the problem when the subscription $\{u_j(t)\}$ is given. The relaxed problem can be decomposed into one sub-problem for each time unit t. The objective function for the smallest problem is given by Equation 11:

$$L_t(\lambda, P) = \sum_{i=1}^{N} \big(c_j - \lambda CPU_j\big) P_j(t) + \sum_{i=1}^{N} s_j u_j(t) + \lambda P_d(t). \tag{11}$$

If enough resources are available to meet the demand, then the minimal values are achieved when the best machines are selected at full capacity and the gap to complete the demand is supplied by the remaining machines with the best value. In other words, machines are selected at the limits imposed by the subscription restrictions in Equation 6, or the maximum number of machines for a given type, P_j^{max} (cloud providers define the maximum values). Under this consideration, one of the remaining machines is selected partially in order to satisfy the remaining demand $P_d(t)$.

Based on this consideration, an optimal solution will be one where the machines selected must be such that the fraction $\frac{c_j}{CPU_j}$ is less than λ values, which means $(c_j - \lambda CPU_j) \leq 0$. Similarly, the value for λ must be large enough to include the machines with $(c_j - \lambda CPU_j) \leq 0$ in a solution set J such that $\sum_{j \in J} CPU_j P_j^* \leq P_d(t)$, where the value is given by

$$P_j^* = \min\{P_j^{max}, \sum_{i=\max\{1, t-Tm_j\}}^{t} u_j(i)\}.$$

Now, let k represent the machines with $\frac{c_k}{CPU_k}$ close to λ and,

$$P_k^* = \frac{P_d(t) - \sum_{j \in J} CPU_j P_j^*}{CPU_k}.$$

Under this consideration the optimal values for $L_t(\lambda, P)$ are given by the following equation:

$$L_t(\lambda, P) = \sum_{j \in J} \Big(\frac{c_j CPU_k + c_k CPU_j}{CPU_k} \Big) P_j^* + \sum_{i=1}^{N} s_j u_j(t). \tag{12}$$

In this solution, the set of machines J represents the first $k-1$ available ("subscribed" or "on demand") resources in non-decreasing order of $\frac{c_j}{CPU_j}$ values such that $\sum_{j \in J} CPU_j P_j^* \leq P_d(t)$. Variable k is an index to the leading machines when the demand is oversupplied, which means the machines with the best possible values for λ is $\lambda^* = \frac{c_k}{CPU_k}$.

Analytical boundary with non-predetermine subscriptions

In order to determine the best strategy for the problems when subscription u_j is not fixed, we decompose the problem as a set of N low-level minimization problems, one for each resource. If the Lagrangian multipliers $\lambda(t)$ are fixed, the objective function for every lower level problem can be expressed as:

$$L_j(P,u) = \sum_{t=1}^{T} \big(c_j - \lambda(t) CPU_j\big) P_j(t) + \sum_{t=1}^{T} s_j u_j(t). \tag{13}$$

The minimization sub-problems are also subject to the same constraints as maximization problems (Equation 6).

For every sub-problem, if $(c_j - \lambda(t) \times CPU_j) \leq 0$, then the value of $P_j(t) > 0$. Consequently, the optimal values for $L_j(P,u)$ are expected to be a negative values. Therefore, it is necessary to select the best available machines, i.e., machines with lower $L_j(P,u)$ values. By using the same λ^* values for all time units, $\lambda(t) = \lambda^*$, it is possible to create an upper boundary condition for the minimization problem. This is given by Equation 14. Let $L_j^*(P)$ be the new value when $\lambda(t) = \lambda^*$ then,

$$\max_{\lambda \geq 0} \min_{u,p} L \leq \sum_{j=1}^{N} L_j^*(P) + \sum_{t=1}^{T} \lambda^* P_d(t). \tag{14}$$

The upper bound is useful to estimate the critical values to select a single machine to meet the demand. The critical limit happens when $L_j^*(P) \to 0$. This means that the minimum possible values for any $\lambda(t)$ in order to select one resource j is represented in Equation 15 as follows:

$$\lambda_j^* = \frac{\sum_{t \in M} \big[s_j u_j(t) + c_j P_j(t)\big]}{CPU_j \sum_{t \in M} P_j(t)}, \tag{15}$$

where M is the set that contains all the time intervals where machine j is used, $c_j - \lambda(t) CPU_j \leq 0$. Equation 15 can then be interpreted as the *normalized average* CPU cost per unit of time.

4.2 Normalized Average Cost Analysis

The formula presented in Equation 15, is a good estimator for the specific average cost for each resource in terms of how long the resource has been used. In Figure 1, we present a graphical representation of the cumulative cost $\sum_{t \in M} \big[s_j u_j(t) + c_j P_j(t)\big]$ for different combinations of s_j and c_j (different cost models), and the average cost per CPU, λ_j^* (computed using Equation 15). We present the information representing the number of time intervals included in set M. For this example, we assume that the machines are used for an entire month. The solid lines in Figure 1 represent the values for the minimum cost by aggregated months in different computing configurations: Amazon AWS initial account, and a combination of Amazon AWS and Google Cloud. On the other

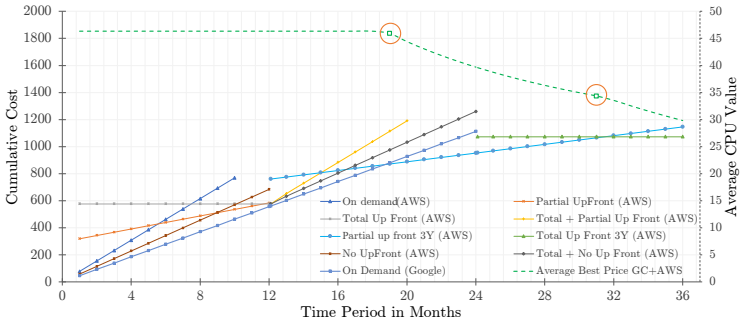

Fig. 1: Cumulative and normalized average cost by month.

hand, the dashed line represents the normalized average cost over the accumulated months for the best possible configurations computed using Equation 15. It is possible to compute the best possible combinations by splitting the demand curve into several portions as represented by the red circles in Figure 1.

Table 3: Interception points of accumulated costs (from Figure 1) for possible machine type configurations.

Types	**On Demand**	**No Upfront**	**PU 1 year**	**TU 1 year**	**AU+ On-De**	**PU 3 years**	**TU 3 years**
T2	6.9	11	11.51	12	16.59	31	36
M3	4.97	9.8	11.51	12	16.88	31	36
M4	4.97	9.7	11.51	12	16.88	31	36
C3	4.97	9.7	11.51	12	16.50	31	36
C4	4.97	9.8	11.51	12	16.70	31	36
GCloud	–	–	–	–	–	19	24

Table 4: Optimal interval for decomposition of the demand curve

TIntervals	**1-6**	**7-10**	**11**	**12**	**13-17**	**19**	**20-31**	**32-36**
EC2								
EC2 GH								
EC2 GH+GC								

In order to decompose the problem into smaller chunks, it is possible to create a list of subsets of resources for the problem under consideration. This approach is presented in Table 3. The table lists the most representative intersection points for the entire set of machine types for Amazon AWS and Google Cloud. The intersection points are measured in time units of months. Table 4 demonstrates the time intervals necessary for the classification of the forecasted demand into the smallest subset of bases.

5 Decomposition of Forecasted Demand

We presented a Lagrange relaxation based method in Section 4 as a means to scale the resource selection problem. In this section, we present a heuristic to first decompose the demand curve into partitions (sub-problems), and then present an algorithm to efficiently solve the sub-problems using an integer linear program solver. We present the algorithm for demand decomposition in Algorithm 1, and the ILP-based solution in Algorithm 2.

Algorithm 1 ***"Decomp"***, Algorithm to decompose of the demand curve

Input: $DCurve$[months], $TISet$
Output: $DPart$
1: $SD \leftarrow$ SortDemand($DCurve$)
2: $DSets \leftarrow$ initialSet($TISet$)
3: **for** D in SD **do**
4: $subD \leftarrow$ partialDemand(D,DCurve)
5: $i \leftarrow$ SelectIIndex($subD, TISet$)
6: $DPartition[i]$.aggrDemand($subD$)
7: **end for**
8: **return** $DPartition$

Algorithm 2 Decomposition Based Approximation Algorithm

Input:$R, DCurve[months], TISet$
Output: TotalCost,P,u
1: $DPart \leftarrow$ Decomp($DCurve, TISet$)
2: $S^{'} \leftarrow \emptyset$
3: **for** d in $DPartition$ **do**
4: **if** $S^{'} \neq \emptyset$ **then**
5: d.updateDemand($S^{'}$)
6: **end if**
7: **if** $d.types$!= "OD" **then**
8: $S^{'} \leftarrow$ solveILP(R, d)
9: **end if**
10: $S[d.I] \leftarrow S^{'}$
11: **end for**
12: $demand \leftarrow DPartition$["OD"]
13: **for** $t \in T$ **do**
14: $S[d.I][t] \leftarrow$ solveILP(R, d.getM(t))
15: **end for**
16: $Cost \leftarrow$ computeCost(R, S)
17: **return** {$Cost$, $S.P$, $S.u$}

The result from Table 4 induces a partition in the forecasted demand. This partitioning is created by grouping the demands with the number of active months in the same time interval of forecasted computational needs (CPU hours). For example, there are three intervals for EC2 GH+GC: 1-19, 20-31, and 32-36. Let us denote the intervals with variable *TISet*. We present our method to decompose a given demand curve in Algorithm 1.

Algorithm 1 begins by ordering the demand per month in a non-decreasing order of values to produce a set SD (Line 1)). We then create a group of empty sets $DSets$, where each set represents one of the time intervals $TISet$ as shown in Table 4. We then construct a vector $subD$ for every value of $D \in SD$. The vector $subD$ contains one value for each time unit (month) and represents the fraction of the demand D_t that remains to be satisfied from the previous time interval, but less than the current time interval. In other words, the value between two intervals in *TISet* for each time unit (month). This action is represented on Line 4 in Algorithm 1 by the function `partialDemand()`. Scalar variable i represents the index of the time interval based on the number of non-zero values in *subDemand*. Variable i will indicate one particular value in *TIntervalsSet*. Further, the time interval is also important for full and partial upfront subscription schemes. Therefore, the variable i will also depend on the range of active

months between non-zero values in *subD*. Variable i is computed by the function `SelectIIndex()`. Finally, we aggregate all the *subD* values in the same interval. Thus, at the end of the execution, we will have one vector of length equal to the number of time units (months) for each interval representing the cumulative demand that needs to be satisfied for a given month. Thus each *demandPartition* for a given interval becomes an independent sub-problem that needs to be solved using Algorithm 2.

The underlying intuition of the solution is to precompute the values of normalized average cost to decompose the problem into easily solvable sub-problems. We present a method to efficiently solve sub-problems in the following section.

5.1 Decomposition Based Approximation Algorithm

Our approximation algorithm uses the demand decomposition method to create a set of small problems, then we use an integer linear program (ILP) to solve each of the sub-problems.

Algorithm 2 illustrates the proposed method to compute the machine configurations that need to be purchased to meet the given demand. The algorithm begins by decomposing the given demand (Line 1) using the method described in Algorithm 1. We process the set $DPartition$ in an order based on the decreasing value of the number of months in the interval. For example, we process the interval $32 - 36$ before processing the interval $20 - 31$ in Table 4.

The variable S is a two-dimensional matrix consisting of N rows and T columns. Each row represents a machine and each column represents a time unit (month). A given value, $S[t][j] = P_j(t)$, will represent the number machines of type j used at time unit t. The variable $S^{'}$ is defined similarly, but holds the temporary solutions that get accumulated in S. Algorithm 2 proceeds by processing each demand in a loop (Lines $3 - 11$). If *d.types* != "OD", OD (On-Demand) then we solve the problem using an integer linear program that we will describe in the following paragraphs. The computed solution from the function `solveILP()` is stored in the temporary variable $S^{'}$. In the subsequent iterations of the for loop, $S^{'}$ will be used to update the next demand (Lines 4 and 5). The temporary values are updated in the solution matrix S (Line 10). Since "On-Demand" part of the composition is not processed in the for loop, we process it separately for each time unit (month) (Lines $13 - 15$). Finally, the total cost of the solution is computed on the function *computeCost* using Equation 1 and the information from the Cloud providers resources R listed on Table 1.

The `solveILP()` function solves an integer linear programming problem using a pseudo-polynomial time algorithm [12], which is known as the inverse Knapsack problem. The problem can be formulated as:

$$\begin{aligned} \underset{u,p}{\text{minimize}} \quad & \sum_{j \in d.J} c_j P_j(t) \\ \text{subject to} \quad & \sum_{j \in d.J} CPU_j P_j(t) \geq d.getM(t). \end{aligned} \tag{16}$$

Note that the problem is solved once for each sub-problem (Line 8), and for at most T times for decisions on "On-Demand" allocations (Line 14). The function $d.getM(t)$ represents the demand for every time step $t \in T$ when called for "On-Demand" allocations, and it represents the maximum demand otherwise. $d.J$ represents the set of "On-Demand" preselect resources. We conclude this section with an observation that the solution computed by the approximation algorithm provides a set of cost-efficient resources that can meet the demand for any given period of time.

6 An Illustrative Case Study

As an illustrative case study, we provide a small example inspired by the Belle-II experiments and using representative data obtained from Amazon and Google Cloud Services. The key information is summarized in Table 1. We use a 36-month demand curve presented in Figure 2 for this example. The synthetic demand represents a time period T of three years, and the total committed resource for each time period (one month) is the constraint and needs to be satisfied for that period. The horizontal axis in Figure 2 represents the time periods in months, and the vertical axis represents the demand in CPUs units. For example, the eleventh month has a total demand of approximately 137 CPU units.

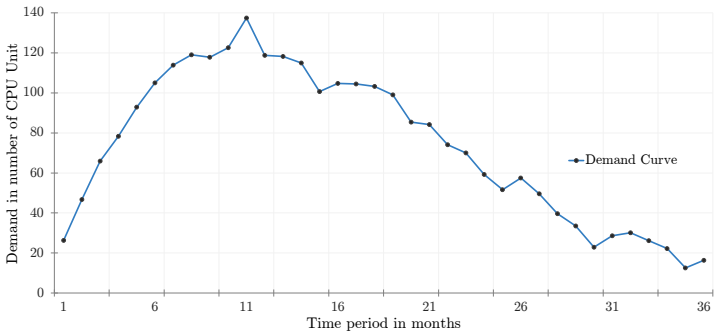

Fig. 2: The Base-forecasted demand curve for an example with a 36-month period.

The optimal solution for this problem is computed using the MILP formulation using $P_j(t)$ and $u_j(t)$ as integer values. The solution was computed using the complete set of available machines with the following price models: On-demand, No Upfront, Partial Upfront, and All Upfront.

We computed the optimal solution on an IBM NEOS Solver with CPLEX. We further note that the computation of solutions using this type of solvers is time-consuming, especially when the number of resources used is large. It took more than one hour, using ten threads (CPU cores) to find the optimal solution Table 6. A single optimal approximate solution is presented in Table 5

which illustrates that only a subset of available resources is sufficient to meet the demand for a given period of time under consideration. This side-by-side comparison shows some excellent similitude, the approximation solution put more effort into the C4 Large machines compared with the optimal solution. Also, the optimal solution is more efficient in finding the minimal set of machines to supply the demand, this is beneficial for implementation logistics. We present

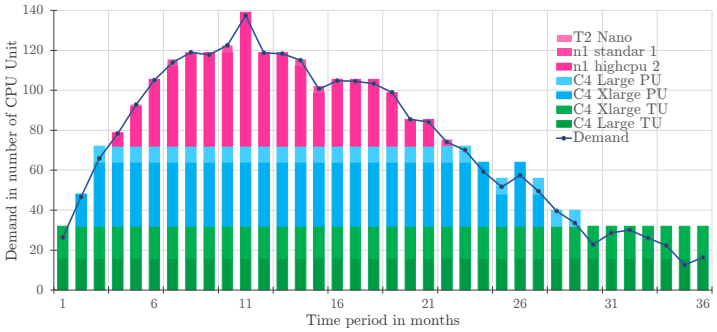

Fig. 3: Illustration of an optimal solution using the MILP-based solution.

Table 5: Optimal and approximation solution machine selection sets.

		Optimal		**Aproximation**	
Machine Type	**Type**	**Machines**	**Months**	**Machines**	**Months**
C4 Xlarge	PU 3 year			2	49
C4 Large	TU 3 year	4	144	4	142
C4 Large	PU 3 year	5	123	1	25
T2 Micro	No-Upfront			2	3
T2 Nano	No-Upfront	4	9	1	3
n1 highcpu 8	Lower-Price			1	1
n1 highcpu 4	Lower-Price			3	7
n1 highcpu 2	Lower-Price	10	93	10	75
n1 standar 1	Lower-Price	1	5	1	5
	Total Cost		$12,144.21		$12,170.29

a possible graphical representation of an optimal solution in Figure 3. Each bar in the figure represents the computing power provided by the selected resources for each month. The black line, as before, represents the forecasted demand. Different colors in the bars (green, blue, and pink) represent specific resources selected to satisfy the demand by an optimal solution computed using MILP formulation. In Table 5, we present the resources used to supply the demand for the entire three-year period. This optimal solution uses a combination of "Total-Upfront 3 year C4 large" machines, "Partial-Upfront 3 year C4 large" machines,

"T2 Nano" from AWS, and "n1 highcpu 2" machines, and one "n1 standard 1" machine from Google Cloud. These machines are used month by month to supply the demand and cost a total of $12,144.21 for the optimal solution. The solution computed by the unit commitment-based formulation provides a set of cost-efficient resources that can meet the predicted demand for a given period of time. However, this solution does not specify how the tasks themselves are assigned to resources. While we do not consider the problem of task assignment in this paper, we refer you to our prior work using combinatorial optimization for energy-efficient assignment of tasks to resources [5]. We will now discuss the idea of demand decomposition for the efficient computation of solutions.

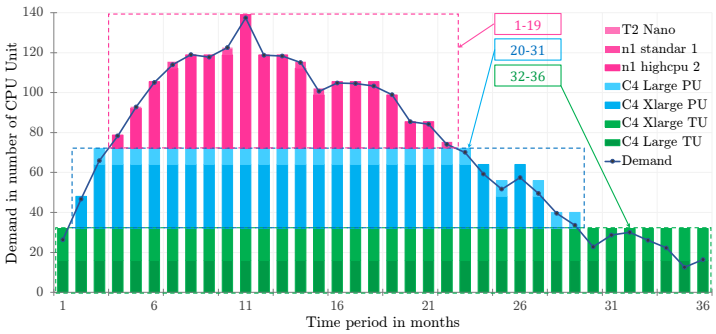

Fig. 4: Illustration of a possible solution and decomposition.

6.1 Decomposition of the Forecasted Demand

As described before the Algorithm 1 is used to construct the *DPartition*, which is represented by the red, blue, and green color on the Figure 4, its colors represent the subdemand (*DPartition*) for each interval 1-19, 20-31, and 32-36 respectively. This decomposition of demand is computed as follows:

- **(1-19)** is the top partition of the demand curve (red color in Figure 4). This sub-demand is obtained by selecting all parts of the demand with at most 19 months of total used in the entire 36 months span. The demand is satisfied using only the "On Demand Google Cloud" machines type (max 67 CPUs units).
- **(20-31)** is the second partition of the demand curve. This sub-demand is obtained by selecting the demand with at least 20 months of sustained use of the machines and maximum 31 months of sustained used. The optimal solution to satisfy this sub-demand is in blue color (max 40 CPUs units) using only the "Partial Upfront" subscription model.
- **(32-36)** is the third partition of the demand curve shown in green color. The optimal solution fills this part of the demand (up to a maximum of 32 CPUs unit) using only an "All Upfront" subscription model. This particular

part of the solution uses these machines for cases where it is required to use more than 32 months of sustained use.

Subsequently, the Algorithm 2 is used to determine the total number of resources used in each step, as described in Table 5. When we compare the cost of the approximate solution it is negligibly higher, which means that quick computation approximated solutions are a promising idea for the forecast problem. For an initial evaluation of the approximation algorithm, we propose a minimal set of experiments in the next section.

7 Experimental Results and Discussion

We evaluated the proposed approximation algorithm using a set of distinct scenarios created by adding random variations induced by a distribution function. We present the experimental results for three groups of simulations: (*i*) 1500 simulations using optimal (ILP) solver with a preselect 16-machine configuration, (*ii*) 1500 simulations using approximate solver (the number of machines is pre-selected for the algorithm), and (*iii*) 1500 simulations using optimal (ILP) solver with 162-machine configuration. We create a total of 1500 different random curves. The set of curves is distributed into several groups of 500-scenarios for each time period (12, 24, and 36 months). We analyzed the performance of the approximation algorithm by comparing it with the optimal solution, both in terms of the running time and the quality of solutions in terms total cost of the configurations. In table Table 6, we present the relative percentage error for

Period	**Relative Error**	Approx.(s)	16-MILP(s)	162-MILP(s)
12	0%	1.1	8.2	909.5
24	0.036%	2.7	12.4	1956.4
36	0.048%	2.8	26.5	3956.2

Table 6: *Relative Performance:* For Belle II (500) use case with three time periods (12, 24, and 36 months), the columns list relative percentage error for the decomposition algorithm with respect to the optimal solution, and the average completion time in seconds for Approximate, Optimal with 16-machines configuration, and Optimal with 162-machines configuration.

the decomposition algorithm with respect to the optimal solution. The results are presented for three time periods (12, 24, and 36 months). The experiment is designed to quantify the accuracy lost by the approximation solutions, which we observe to be negligible with respect to the optimal solutions, less than 0.05% for all cases.

Table 6 also summarizes the average completion time for three approaches: Approximate, Optimal with 16-machine configuration, and Optimal with 162-machine configuration. We observe that the time to compute the Approximate solution, ranging from 1.4 to 2.8 seconds, is orders of magnitude smaller. The average improvement in performance relative to the optimal solution is 9× for

16 machine-configuration, and 1412× for 162-machine configurations. The approximate method also scales across the forecasted time periods (12, 24, 36). Thus, demonstrating superior performance in quality vs compute time.

8 Related Work

Efficient scheduling of computational tasks is a classical problem that has been studied extensively. The minimum makespan scheduling problem on independent machines has also been studied extensively [4]. Given a set of tasks and a set of resources, the objective is to assign tasks to resources such that the load on each resource is balanced. In particular, the maximum load of any resource is minimized. A relaxed variant of this problem has been solved using the semi-matching formulation by several researchers [8]. Scheduling of tasks on heterogeneous resources with different energy consumption patterns is an emerging topic of research [16], that employs mathematical programming for computing efficient solutions. The novelty of our scheduling work comes from employing scalable approaches to solve computationally challenging optimization problems. Preselection of resources in a cost-efficient manner allows us to formulate and solve the complex problem at multiple levels. The unit commitment analogy enables us to build from a large body of work on efficient optimization approaches [11]. However, to the best of our knowledge, we believe this is the first work that employs an analysis of the Lagrange relaxation for the unit commitment-based formulation of cloud resource allocation for complex scientific workflows.

9 Summary and Future Work

We presented a computationally efficient approximation algorithm using Lagrange relaxation for the optimal allocation of cloud resources using demand decomposition as a heuristic. We presented a detailed description of mathematical formulation and intuition for employing the Lagrange relaxation-based technique for solving the integer programming problem. We supported the efficiency of the proposed approach using a representative case study and a small set of experiments inspired by the Belle-II experiment and representative cloud computing resources.

In the near future, we plan to conduct an extensive empirical evaluation of the proposed approach using a variety of demand scenarios. We also plan to integrate the solution for Belle-II workflows that can exploit dedicated HPC resources as well as cloud computing resources. We also plan to develop a continuous optimization formulation of the problem that can be used in a continuous setting for resource allocation. We believe that a fast solution to this problem can be beneficial for future applications that are intended to test multiple variations of the forecast demand almost in real-time.

References

1. Asner, D.M., Dart, E., Hara, T.: Belle II: Experiment network and computing. Tech. Rep. arXiv:1308.0672. PNNL-SA-97204 (Aug 2013), contributed to CSS2013 (Snowmass)
2. Friese, R.D., Halappanavar, M., Sathanur, A.V., Schram, M., Kerbyson, D.J., de la Torre, L.: Towards efficient resource allocation for distributed workflows under demand uncertainties. Proceedings of the 21st Workshop on Job Scheduling Strategies for Parallel Processing (2015)
3. Gao, P.X., Curtis, A.R., Wong, B., Keshav, S.: It's not easy being green. ACM SIGCOMM Computer Communication Review **42**(4), 211–222 (2012)
4. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of discrete mathematics **5**(2), 287–326 (1979)
5. Halappanavar, M., Schram, M., de la Torre, L., Barker, K., Tallent, N.R., Kerbyson, D.J.: Towards efficient scheduling of data intensive high energy physics workflows. In: Proceedings of the 10th Workshop on Workflows in Support of Large-Scale Science. pp. 3:1–3:9. WORKS '15, ACM, New York, NY, USA (2015)
6. Halappanavar, M., Schram, M., de la Torre, L., Barker, K., Tallent, N.R., Kerbyson, D.J.: Towards efficient scheduling of data intensive high energy physics workflows. In: Proceedings of the 10th Workshop on Workflows in Support of Large-Scale Science. WORKS '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2822332.2822335
7. Hara, T.: Belle II: Computing and network requirements. In: Proc. of the Asia-Pacific Advanced Network. pp. 115–122 (2014)
8. Harvey, N.J.A., Ladner, R.E., Lovász, L., Tamir, T.: Semi-matchings for bipartite graphs and load balancing. J. Algorithms **59**(1), 53–78 (Apr 2006)
9. Juve, G., Deelman, E.: Scientific workflows and clouds. XRDS **16**(3), 14–18 (mar 2010). https://doi.org/10.1145/1734160.1734166
10. Kamiński, B., Szufel, P.: On optimization of simulation execution on amazon ec2 spot market. Simulation Modelling Practice and Theory **58**, 172–187 (2015)
11. Mallipeddi, R., Suganthan, P.N.: Unit commitment – a survey and comparison of conventional and nature inspired algorithms. International Journal of Bio-Inspired Computation **6**(2), 71–90 (2014). https://doi.org/10.1504/IJBIC.2014.060609
12. Roland, J., Figueira, J.R., De Smet, Y.: The inverse 0,1-knapsack problem: Theory, algorithms and computational experiments. Discrete Optimization **10**(2), 181–192 (2013). https://doi.org/https://doi.org/10.1016/j.disopt.2013.03.001, `https://www.sciencedirect.com/science/article/pii/S1572528613000066`
13. Saravanan, B., Das, S., Sikri, S., Kothari, D.P.: A solution to the unit commitment problem—a review. Frontiers in Energy **7**(2), 223–236 (2013)
14. Schwartz, R., Dodge, J., Smith, N.A., Etzioni, O.: Green AI. Commun. ACM **63**(12), 54–63 (nov 2020). https://doi.org/10.1145/3381831
15. Singer, G., Livenson, I., Dumas, M., Srirama, S.N., Norbisrath, U.: Towards a model for cloud computing cost estimation with reserved instances. Proc. of 2nd Int. ICST Conf. on Cloud Computing, CloudComp 2010 (2010)
16. Tarplee, K., Friese, R., Maciejewski, A., Siegel, H.: Efficient and scalable pareto front generation for energy and makespan in heterogeneous computing systems. In: Recent Advances in Computational Optimization, Studies in Computational Intelligence, vol. 580, pp. 161–180. Springer International Publishing (2015)
17. Wright, B.: A review of unit commitment. ELENE4511, May **28** (2013)