Decision Model for Cloud Computing under SLA Constraints

Artur Andrzejak, Derrick Kondo, Sangho Yi

Decision Model for Cloud Computing under SLA Constraints

Artur Andrzejak
Zuse Institute Berlin (ZIB), Germany
Email: andrzejak@zib.de

Derrick Kondo
INRIA, France
Email: derrick.kondo@inria.fr

Sangho Yi
INRIA, France
Email: sangho.yi@inria.fr

***Abstract*—With the recent introduction of Spot Instances in the Amazon Elastic Compute Cloud (EC2), users can bid for resources and thus control the balance of reliability versus monetary costs. A critical challenge is to determine bid prices that minimize monetary costs for a user while meeting Service Level Agreement (SLA) constraints (for example, sufficient resource availability to complete a computation within a desired deadline). We propose a probabilistic model for the optimization of monetary costs, performance, and reliability, given user and application requirements and dynamic conditions. Using real instance price traces and workload models, we evaluate our model and demonstrate how users should bid optimally on Spot Instances to reach different objectives with desired levels of confidence.**

***Index Terms*—Cloud Computing, SLA's, Optimization**

I. Introduction

With the recent surge of Cloud Computing systems, the trade-offs between performance, reliability, and costs are more fluid and dynamic than ever. For instance, in December 2009, Amazon Inc. released the notion of Spot Instances in the Amazon Elastic Compute Cloud (EC2). These Spot Instances are essentially idle resources in Amazon's data centers. To allocate them, a user must first bid a price for the instance. Whenever the current instance price falls at or below the bid price, the Spot Instance is made available to the user. Likewise, if the current price goes above the bid price, the user's Spot Instances are terminated without warning. Recent reports indicate that Google Inc. has developed a prototype that uses a similar market-based approach for resource allocation [1]. Many argue [2] that market economies will be increasingly prevalent in order to achieve high utilization in often-idle data centers.

Given these market-like resource allocation systems, the user is presented with the critical question of how to bid for resources. Clearly, the bid directly affects the reliability of the Spot Instances, the computational time, and the total cost of the user's job. The problem is challenging as often the user has Service Level Agreement (SLA) constraints in mind, for instance, a lower bound on resource availability, or an upper bound on job completion time.

Our main contribution is a probabilistic model that can be used to answer the question of how to bid given SLA constraints. A broker can easily apply this model to present automatically to the user a bid (or set of bids) that will meet reliability and performance requirements. This model is particularly suited for Cloud Computing as it is tailored for environments where resource pricing and reliability vary significantly and dynamically, and where the number of resources allocated initially is flexible and is almost unbounded. We demonstrate the utility of this model with simulation experiments driven by real price traces of Amazon Spot Instances, and workloads based on real applications.

This paper is organized as follows. In Section II, we detail the market system of Amazon's Spot Instances, and describe our system model for the application and its execution. In Section III, we present our method for optimizing a user's bid and simulation approach. In Section IV, we show the utility of our model through simulation results. In Section V, we compare and contrast our approach with previous work, and in Section VI, we summarize our contributions and describe avenues for future work.

II. System Model

A. Characteristics of the Spot Instances

Amazon users can bid on unused EC2 capacity provided as (42+6)[1] different types of *Spot Instances* [3], [4]. The price of each type (called the *spot price*) changes based on supply and demand (see Figure 1). If a customer's bid price meets or exceeds the current spot price, the requested resources are granted. Conversely, EC2 revokes the resources immediately without any notice when a user's bid is less than or equal to the current spot price. We call this an *out-of-bid* event or a *failure*, see Figure 2. The requested instances with the same bid price are completely allocated or deallocated as a whole (i.e. synchronously). We assume that the bid and number of instances requested does not have a "feedback loop" nor influences future pricing.

The following rules characterize some minor aspects of this schema. Amazon does *not* charge the latest partial hour when it stops an instance, but the latest partial hour is charged (as a whole hour) if the termination is due to the user. Each hour is charged by the current spot price, which could be lower than the user's bid. The price of Amazon's storage service is negligible - at most 0.15 USD for 1 GB-month, which is much lower than the price of computation [5].

[1]In the beginning of Dec. 2009, EC2 started to provide 42 types of instances, and 6 more types are added in the end of Feb. 2010.

Table I
USER PARAMETERS AND CONSTRAINTS

Notation	Description
n_{inst}	number of instances that process the work in parallel
n_{max}	upper bound on n_{inst}
W	total amount of work in the user's job
W_{inst}	workload per instance (W/n_{inst})
T	task length, time to process W_{inst} on a specific instance
B	budget per instance
c_B	user's desired confidence in meeting budget B
t_{dead}	deadline on the user's job
c_{dead}	desired confidence in meeting job's deadline
u_b	user's bid on a Spot Instance type
I_{type}	EC2 instance type

Note that there is a potential exploitation method to reduce the cost of the last partial hour of work called "Delayed Termination" [6]. In this scenario, a user waits after finished computation almost to the next hour-boundary for a possible termination due to an out-of-bid situation. This potentially prevents a payment for the computation in the last partial hour.

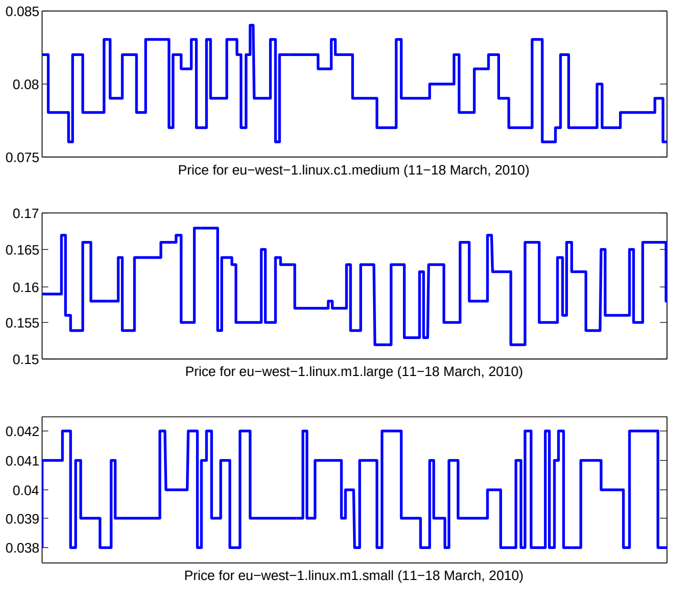


Figure 1. Price history for some Spot Instance types (in USD per hour; geographic zone *eu-west*; operating system Linux/UNIX)

B. Workloads and SLA Constraints

We assume a user is submitting a compute-intensive, embarrassingly parallel job that is divisible. Divisible workloads, such video encoding and biological sequence search (BLAST, for example), are an important class of application prevalent in high-performance parallel computing [7]. We believe this is a common type of application that could be submitted on EC2 and amenable to failure-prone Spot Instances.

The job consists of a *total amount of work* W to be executed by n_{inst} many instances (of the same type) in parallel, which yields $W_{inst} = W/n_{inst}$, the *workload per instance*. Note that n_{inst} can be usually varied up to a certain limit given by the number n_{max} of "atomic" tasks in the job; thus, $n_{inst} \leq n_{max}$. We measure W and W_{inst} in hours of computation needed on a single EC2 instance with processing capacity of one EC2 Compute Unit (or simply *unit*), i.e. equivalent CPU capacity of a $1 \ldots 1.2$ GHz 2007 Opteron or 2007 Xeon processor [4]. We refer to amount of work done in one hour on a machine with one EC2 Compute Unit as *unit-hour*. We call the time needed for processing W_{inst} on a specific instance type I_{type} the *task length* $T = T(I_{type})$. Simplifying slightly, we assume the perfect relationship $T(I_{type}) = W_{inst}/(\text{processing capacity of } I_{type})$. Note that T is the "net" computation time excluding any overheads due to resource unavailability, checkpointing and recovery. This is different than the actual clock time needed to process W_{inst}(called *execution time*, see Section III-A), which is at least T.

Table II
PARAMETERS OF THE EXEMPLARY WORKLOADS

Workload	I_{type}	n_{max}	W_{inst}	T	t_{dead}	c_{dead}
W1	2.5GHz	$20,000$	11.5	4.6h	9d	0.9
W2	2.5GHz	50	6.83	2.7h	17.9h	0.8

Further constraints which might be specified as part of user's input are:

- *budget* B, upper bound on the total monetary cost per instance
- c_B: user's desired confidence in meeting this budget
- *deadline* t_{dead}, upper bound on the execution time (clock time needed to process W_{inst})
- c_{dead}: the desired confidence in meeting t_{dead}.

Table I lists the introduced symbols.

C. Optimization Objectives

We assume that a user is primarily interested in answering the following questions:

Q1. Can the job be executed under specified budget and deadline constraints?

Q2. What is the bid price and instance type that minimizes the total monetary costs?

Q3. What is the distribution of the monetary cost and the execution time for a specific instance type and bid price?

To simplify our approach, we assume that the instance type and the bid price are fixed, and focus on answering Q3. In order to answer Q1 and Q2, one needs just to evaluate a small number of relevant combinations of instance types and bid prices (see Section IV-E). The user can also apply this approach dynamically, i.e. periodically re-optimize the bid and instance type selection during the computation, depending on the execution progress and changes in spot prices.

D. Exemplary Workloads

To emulate real applications, we base the input workload on that observed in real Grids and Desktop Grids. The majority of workloads in traditional Grids consist of pleasantly

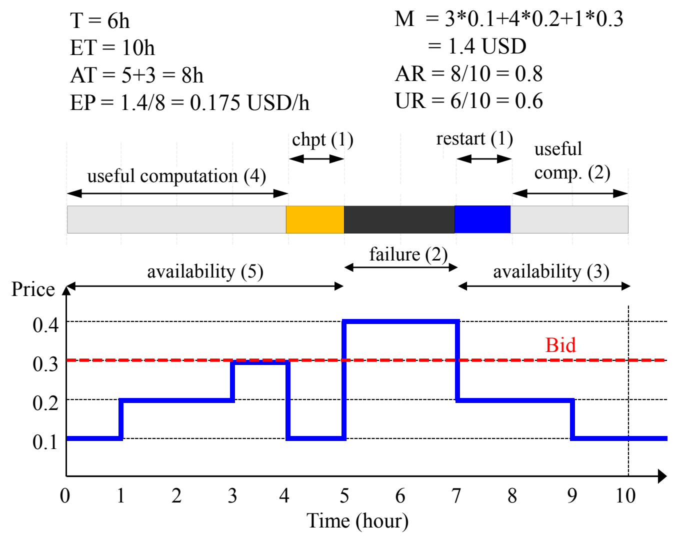


Figure 2. Illustration of the execution model and computation of the random variables (RVs)

parallel and independent bags of tasks. By far, the majority of workloads in Desktop Grids consist of compute-intensive and independent tasks. Desktop Grids resemble Spot Instances in the sense that a desktop user can reclaim his/her machine at any time, preempting the executing application. As such, we believe these workloads are representative of applications amenable or actually deployed across Spot Instances.

The Grid and Desktop Grid workload parameters correspond to relatively small and relatively large jobs respectively. This is partly because Grids have on the order of hundreds to thousands or resources, and Desktop Grids have on the order of tens to hundreds of thousands of resources. So the workload size is reflective of the platform size.

The specific Grid and Desktop Grid workload parameters that we use are based the BOINC Catalog [8] (Workload W1), and Grid Workload Archive [9], [10] (Workload W2), respectively (Table II).

Workload W1. In the BOINC Catalog [8], we find that the median job deadline t_{dead} is 9 days, and the mean task length T is 4.6 hours (276 minutes) on a 2.5GHz core. This translates to a mean per-instance workload W_{inst} of 11.5 unit-hours. We will assume in the following an instance type with 2.5 EC2 Compute Units (e.g. a single core of the High-CPU medium instance type) [3] so that the task lengths remain around the original values. We also learned that a typical value for n_{max} is 20,000 tasks. Thus, we center the range of W, t_{dead}, W_{inst} (or equivalently, of T) around these values. See Table II for these and additional parameters.

Workload W2. From the Grid Workloads Archive [9], [10], we find that the mean job deadline t_{dead} is 1074 minutes (17.9 hours), and the mean task length T is 164 minutes (2.7 hours) on a 2.5GHz core. This gives us an average per-instance workload W_{inst} of 6.75 unit-hours. n_{max} is 50 tasks, the highest average reported in [10]. We will again assume in the following an instance type with 2.5 EC2 Compute Units for a single core. This let us center our study around the values of T and t_{dead} as reported in the third row of Table II.

Table III
RANDOM VARIABLES (RVS) USED FOR MODELING

Notation	Description
ET	execution time of the job (clock time)
AT	availability time (total time in-bid)
EP	expected price, i.e. (cost per instance)/AT
M	monetary cost $AT \cdot EP$ per instance
AR	availability ratio AT/ET
UR	utilization ratio T/ET

III. Modeling and Optimization Approach

A. Execution Scenario

Figure 2 illustrates an exemplary execution scenario. A user submits a job with a total amount of work W of 12 unit-hours with $n_{inst} = 2$ which translates to a $W_{inst} = 6$ unit-hours and the task time (per instance) T of 6 hours (assuming EC2's "small instance" server). User's bid price u_b is 0.30 USD, and during the course of the job's computation, the job encounters a *failure* (i.e. an out-of-bid situation) between time 5 and 7. The total availability time was 8 hours, from which the job has $(4+2) = 6$ hours of useful computation, and uses 1 hour for checkpointing and 1 hour for restart. (Obviously, these overheads are unrealistic, but defined here for simplicity of the example.) The clock time needed until finishing was 10 hours. During the job's active execution, the spot price fluctuates; there are 3 hours at 0.10 per time unit, 4 hours at 0.20 per time unit, and 1 time unit at 0.30 per time unit, giving a total cost of 1.40. Thus the expected price is $1.40/8 = 0.175$ (USD / hour).

B. Modeling of the Execution

The execution is modeled by the following random variables (RVs):

- *execution time* ET is the total clock time needed to process W_{inst} on a given instance (or, equivalently, to give the user T hours of useful computation on this instance); in the example, ET assumes the value of 10 hours
- *availability time* AT is the total time in-bid; in our example, this is 8 hours
- *expected price* EP is the total amount paid for this instance to perform W_{inst} divided by the total availability time; note that always $EP \leq u_b$
- *monetary cost* M is the amount to be payed by the user per instance, defined by $M = AT \cdot EP$; in the example, we have (in USD) $M = 8 \cdot 0.175 = 1.40$.

Note that as we assume n_{inst} instances of the *same* type, they all are simultaneously in-bid and out-of-bid; therefore, the values of the variables ET, AT, EP are identical for all instances deployed in parallel. In particular, the *whole* job completes after time ET, and so ET is also the job's execution time. Furthermore, all the above RVs depend on the checkpointing strategy. As designing and optimizing such a strategy is a complex undertaking beyond the scope of this work, we assume here one of the following two simple strategies taken from [6]:

- OPT - the *optimal strategy*, where a checkpoint is taken just prior to a failure; this (unrealistic) strategy serves as a base-case for cost comparison
- HOUR - the *hourly strategy*, where a checkpoint is taken is at a boundary of each paid hour (measured from the start of current availability interval).

C. Models Independent of the Task Time T

The following RVs give an alternative characterization of the execution process:

- *availability ratio* $AR = AT/ET$ is the ratio of the total availability time (time in-bid) to execution time; in our example, $AR = 8/10$
- *utilization ratio* UR is T/ET, or ratio of the total useful computation to the execution time; in the example, $UR = 6/10$.

Originally we have attempted to find distributions of AR and UR independently of T, and derive the distributions of RVs in Section III-B using following relations (expected price and monetary cost are only bounded from above):

$$\begin{aligned} ET &= T/UR, \\ AT &= AR \cdot ET = AR \cdot T/UR, \\ EP &\leq u_b, \\ M &\leq u_b \cdot AR \cdot T/UR. \end{aligned}$$

This approach requires to store only the distributions of AR and UR for representative pairs (bid price u_b, instance type); the independence of the the task length T saves a lot of storage and simulation time. However, our experimental findings show that both AR and UR depend significantly on T (this relation is shown in Section IV-B3). Thus, to optimize accurately, we need to store the distributions of these RVs for many values of T. In face of this effect we decided to simulate and store directly the distributions of RVs introduced in Section III-B (for relevant combinations of bid price, instance type, and intervals of T).

D. Decision Models

Verifying feasibility and optimizing of the bid prices require that we can "represent" the distributions of a random variable (RV) X as a (scalar) number. As the Cumulative Distribution Functions (CDFs) of our RVs are strictly increasing (disregarding sampling and simulation errors), we can use to this aim the value $X(y) = F_X^{-1}(x)$ of the inverse CDF $F_X^{-1}(y)$ for the *y-th percentile,* $y \in [0, 1]$. In other words, for a RV X we write $X(y)$ for x s.t. $\Pr(X < x) = y$. For example, $ET(0.5)$ is the median execution time.

For fixed input parameters (Table I), instance type I_{type} and bid price u_b we retrieve first the pre-computed distributions of the RVs ET and M (Table III). The feasibility decisions are then made as follows:

- the deadline constraint can be achieved with confidence c_{dead} iff $t_{dead} \geq ET(c_{dead})$
- the budgetary constraint can be achieved with confidence c_B iff $B \geq M(c_B)$.

To find the optimal instance type and bid price, we compute $ET(c_{dead})$ and $M(c_B)$ and check the feasibility (as above) for all relevant combinations of both parameters. As these computations are basically "look-ups" in tables of previously computed distributions, the processing effort is negligible. Among the feasible cases, we select the one with the smallest $M(c_B)$; if no feasible cases exist, the job cannot be performed under the desired constraints. This process is demonstrated in Sections IV-C and IV-D.

Only ET and M are used in the above decision process. However, the remaining RVs (AT, EP, AR, UR) provide additional characterization of the execution and are beneficial in defining more advanced SLA conditions. For example, AR can be used to guarantee certain minimum resource availability in a time interval, and so ensure a lower bound on execution progress (per time unit).

Table IV
USED INSTANCE TYPES (ZONE: EU-WEST-1; OS: LINUX; CLASS: HI-CPU = HIGH-CPU, STD = STANDARD, HI-MEM = HIGH-MEMORY)

Symbol	Class	API Name	Mem. (GB)	Total Units	Num. Cores	Units / Core
A	hi-cpu	c1.medium	1.7	5	2	2.5
B	hi-cpu	c1.xlarge	7	20	8	2.5
C	std	m1.small	1.7	1	1	1
D	std	m1.large	7.5	4	2	2
E	std	m1.xlarge	15	8	4	2
F	hi-mem	m2.2xlarge	34.2	13	4	3.25
G	hi-mem	m2.4xlarge	68.4	26	8	3.25

E. Simulation Method

We implemented a simulator that uses real Spot Instance price traces to find the distributions of the RVs shown in Table III via the Monte-Carlo method. The distributions are obtained via $10,000$ experiments (per unique set of input parameters). Each experiment corresponds to a single task execution (on one instance) as outlined in Figure 2, where the starting point is selected randomly between Jan. 11th and Mar. 18th, 2010, and the execution terminates as soon as the cumulative "useful execution" time of T has been reached. A set of input parameters consists of the instance type I_{type}, the bid price u_b, task length T and the checkpointing strategy (OPT or HOUR). The simulator was written in the C language with standard libraries, and can be ported to any UNIX-type operating system with minimal effort.

IV. RESULTS

A. Evaluation Settings

In our study we used all seven types of Spot Instances, which were available starting in December 2009. We considered prices of instance types that run under Linux/UNIX operating system (OS) and are deployed in the zone eu-west-1. Table IV shows the symbols, class (high-CPU, standard, high-memory), API names, RAM memory (in GB), total processing capacity in EC2 Compute Units (*units*), number of cores per instance, and processing capacity per core (in units) (see [4] for details).

If not stated otherwise, we use the instance type A with task length T of 276 minutes (4.6 hours) and the (realistic) checkpointing policy HOUR (also abbreviated H). We used the same settings of the remaining parameters, such as checkpointing cost and rollback cost in time as in [6]. Furthermore, our models assume that a job is executed on a single instance only, as running several instances (of same type) in parallel yields the identical time and proportional cost behavior.

B. *Impact of Input Factors*

In this section we study how the input parameters from Table I influence the distributions of the random variables (especially ET and M), and investigate the overhead of the checkpointing strategies.

1) Execution Time and Monetary Cost: Figure 3 (left) shows (in hours) execution time $ET(p)$ for various values of p and bid prices u_b. Note that instead of assuming a fixed deadline t_{dead}, we study here which deadline - represented by $ET(p)$ - can be achieved with confidence $p = c_{dead}$, see Section III-D. Obviously low bid prices in conjunction with high values of c_{dead} lead to extremely long execution times - up to factor 100 compared to the task length T. For sufficiently high bid prices ($u_b \geq 0.077$ USD) the execution time drops to half of the peak value, but only in the "top range" of the bid prices the execution time is on the order of T. Figure 3 (right) shows the monetary cost $M(p)$.

Differently from the execution time, $M(p)$ increases only slightly with the bid price, and is relatively indifferent to the percentile p (corresponding to the budget confidence c_B). We explain this by the fact that a long execution time comes primarily from out-of-bid time for which the user is not charged: even during an execution time of 400 hours there might be only small in-bid time (on the order of $T = 4.6$ hours) which is charged by EC2. In summary, a user does not save much (about 10% of the costs) when bidding low but risks very high execution times. Analogous results hold for the case of the optimal checkpointing strategy (OPT) (not shown).

2) Checkpointing Overhead: Figure 4 shows (in hours) the difference $AT(p) - T$ for optimal checkpointing OPT (left) and the hourly checkpointing HOUR (right), where both the percentile p and the bid price u_b are varied. The value $AT(p) - T$ represents the time overhead due to checkpointing, lost work prior to a failure, and recovery (as $AT(p)$ is time provided by EC2). Clearly, the HOUR approach has much higher overhead, which can amount to over 40% of the task length T for low bid prices and high p's. Low prices lead to more frequent failures, which increases the checkpointing overhead.

3) Influence of Task Length: Figure 5 illustrates how the distributions of the random variable AR depends on the task

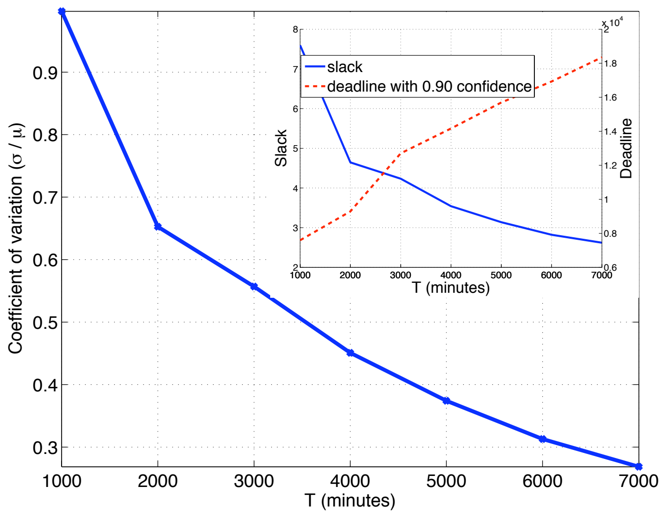
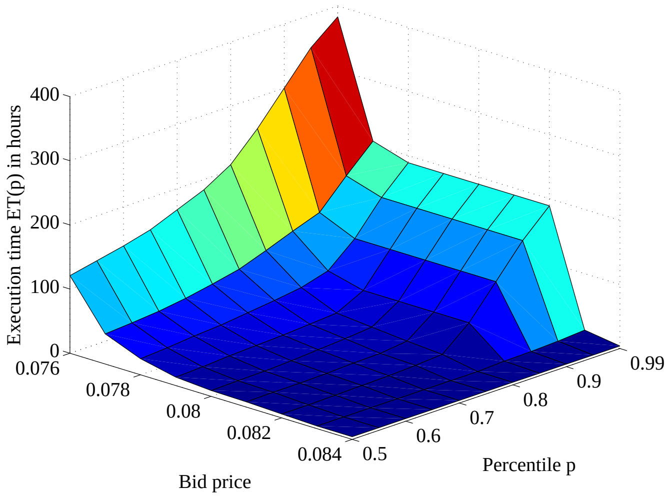


Figure 6. Outer: Coefficient of variation of execution as task length T increases with bid price u_b=0.80. Inner: Slack and deadline as T increases

length T. The left figure shows the median of AR as a function of the bid price and T. Obviously T does not influence $AR(0.5)$. For comparison, the right figure shows $AR(0.9)$ (i.e. value v larger / equal than 90% of values assumed by AR, see Section III-D). Here the influence of T is strongly visible, especially for low bid prices. As a consequence, distribution of AR depend on T, and cannot be stored only as functions of bid price and instance type (see Section III-C). A very similar effect occurs for the utilization ratio UR. We also studied the effect of T on the monetary cost M but we did not identify any relation except for an almost linear increase of $M(p)$ (for any fixed p) with growing T.

We also find that the coefficient of variation (standard deviation / mean) of ET decreases sub-linearly with T (see Figure 6) where we use a bid price u_b of 0.80 USD. This can be explained by the *Law of Large Numbers*, which states that the sample mean approaches the expected value as as number of samples (in this case availability durations) goes towards infinity. Also, the sample variance (the standard deviation squared) is the ratio of the distribution variance squared to the sample size. We observe that as the sample size becomes large, the variance becomes relatively low.

This fact can be leveraged by the user to determine initial values of the deadline relative to T. In Figure 6, we show the 90th percentile for ET (i.e. $ET(0.9)$) in the red plot, and slack with the blue plot. The slack is defined as the ratio of the 90th percentile for ET to T. The rapid decrease of slack is due to the decrease of ET's variance as the number availability durations increases. Intuitively, Figure 6 gives guidance as to how much room a user must give between the task deadline and amount of work T. For instance, if T is roughly 5500 minutes, a reasonable deadline that can be achieved with 0.90 confidence is $3 * 5500$ minutes.



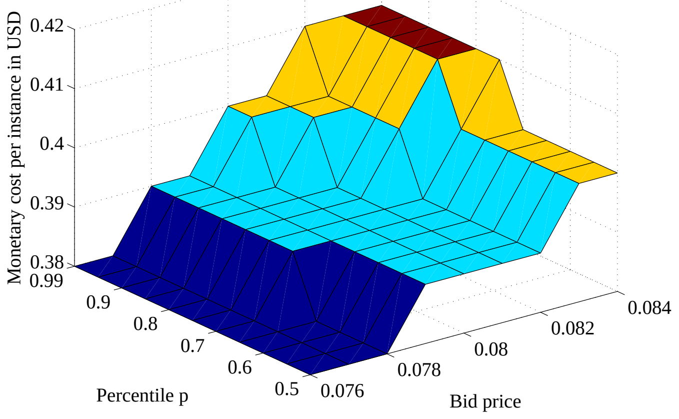
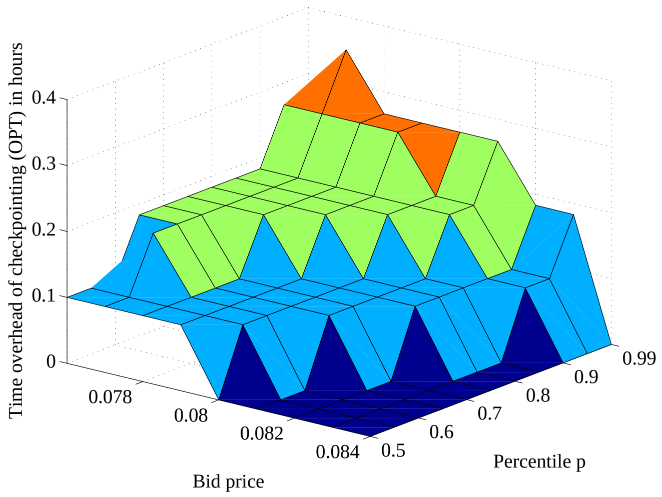


Figure 3. Influence of bid price and percentile p on the execution time $ET(p)$ (left) and monetary cost per instance $M(p)$ (right) (p corresponds to confidence c_{dead} for ET and to confidence c_B for M)



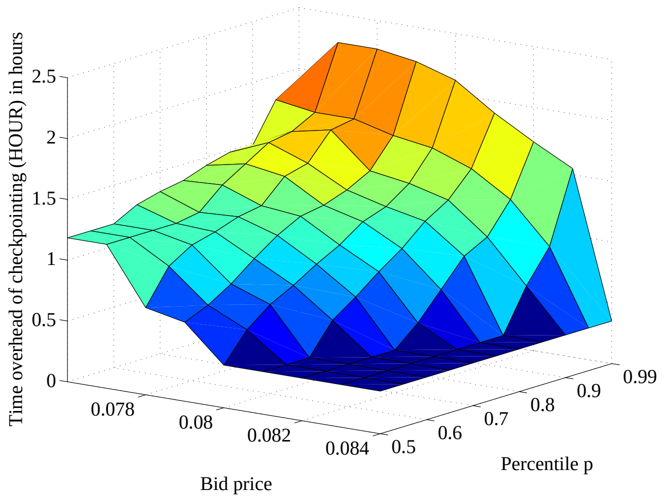


Figure 4. Overhead $(AT(p) - T)$ of the checkpointing for the strategy OPT (left) and HOUR (right) for various bid prices and percentiles p

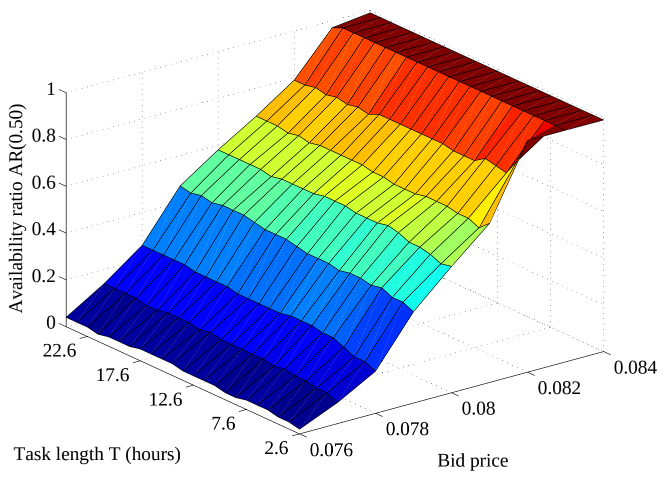
C. *Meeting Deadline and Budgetary Constraints for W1*

In this section we study distributions of the execution time and the monetary constraints for the workload W1 (Section II-D). We also demonstrate how these interplay with the constraints introduced in Section II-B.

Figure 7 shows the cumulative distribution function (CDF) of the execution time ET and the monetary costs per instance M according to different values of the bid price u_b, checkpointing strategy, and the task length T. The red vertical lines represent the given deadline t_{dead} (Table II) and the budget B, while the blue horizontal lines represent their required confidence. In the results of Figure 7 (c) the lowest bid price (0.076 USD) cannot meet the user's given deadline constraints t_{dead} and c_{dead}, while the two highest bid prices (0.083 ~ 0.084 USD) cannot meet the budget limit constraints B and c_B. Note that the value of T also affects the possible range of bid prices.

The constraints t_{dead} and c_{dead} act as a "high-pass filter" of possible bid prices, and the other constraints B and c_B act as a "low-pass filter". Figure 8 shows the range of bid prices according to all results in Figure 7. As we can observe from Figure 8 some bid prices are not feasible. In these cases we need to either decrease the confidence values or set higher limits on the deadline and the budget.

Figure 9 shows the CDF of the execution time and the monetary cost for $T = 246$ minutes. Table V shows the lowest monetary costs in the case of this figure according to different values of c_{dead}. This result demonstrates that the total costs can be significantly affected by changing the degree of the confidence value. By comparing the two cases, $c_{dead} = 0.90$ and $c_{dead} = 0.82$, we observe that using slightly lower confidence can reduce more than 21% of the monetary costs.



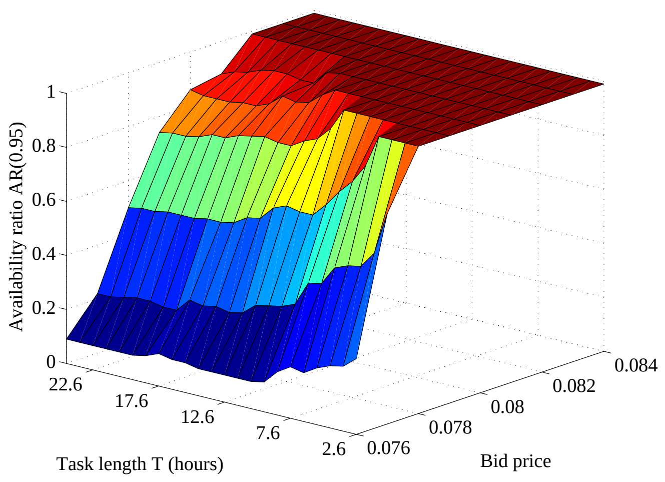


Figure 5. Availability ratio $AR(p)$ for $p = 0.5$ (median) (left) and for $p = 0.9$ (right) depending on the bid price and the task length T

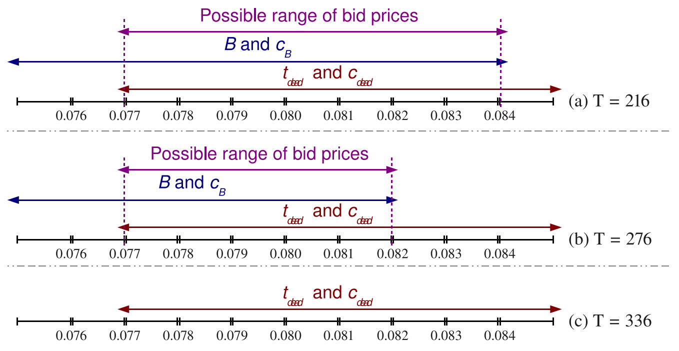


Figure 8. The ranges of the bid prices according to the results in Figure 7

Table V
THE LOWEST MONETARY COSTS (USD) IN CASE OF FIGURE 9 FOR DIFFERENT VALUES OF c_{dead} AND BID PRICE u_b

c_{dead}	bid = 0.076		bid = 0.077		bid = 0.078		bid = 0.079	
	OPT	H(our)	OPT	H	OPT	H	OPT	H
0.99	-	-	-	-	**0.39**	**0.39**	0.39	0.39
0.90	-	-	**0.38**	**0.38**	0.39	0.39	0.39	0.39
0.82	**0.30**	**0.38**	0.38	0.38	0.39	0.39	0.39	0.39

D. Meeting Deadline and Budgetary Constraints for W2

In this section we present a study according to the parameters for the workload W2. Figure 10 shows the CDF of the total execution time and the total monetary costs per instance according to each bid price, checkpointing strategy, and the task time T. To compensate that the deadline ($c_{dead} = 1074$ minutes) is much smaller than the deadline for workload W1, the confidence c_{dead} of meeting t_{dead} is assumed to be lower.

Table VI shows the lowest execution time derived from Figure 10 according to the different budget B and the confidence c_B values. We find that a slight change of the budgetary confidence c_B has significant impact on the execution time. In addition, there is a significant cut-off on the total budget. If the user assumes 0.01 USD more for the budget B, she will benefit from a significant reduction of execution time at the same confidence value.

Table VI
THE LOWEST EXECUTION TIME (MINUTES) ACCORDING TO FIGURE 10 FOR DIFFERENT VALUES OF B AND c_B

c_b	Budget per instance B (USD)					
	≤ 0.22		0.23		≥ 0.24	
	OPT	H(our)	OPT	H	OPT	H
0.90	-	-	1080	1140	180	180
0.80	-	-	840	900	180	180
0.70	-	-	660	720	180	180
0.60	-	-	180	180	180	180
0.50	-	-	180	180	180	180

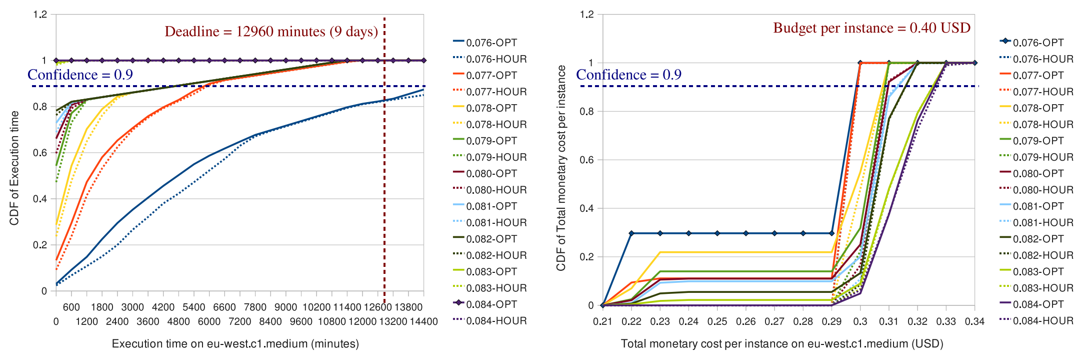
Table VII
BIDING PRICE COMPARISON ACROSS INSTANCE TYPES (IN US-CENTS)

Symbol	Class	Total Units	Low Bid	High Bid	Low / Unit	High / Unit	Ratio in %
A	hi-cpu	5	7.6	8.4	1.52	1.68	10.5
B	hi-cpu	20	30.4	33.6	1.52	1.68	10.5
C	std	1	3.77	4.2	3.77	4.2	11.4
D	std	4	15	16.8	3.75	4.2	12.0
E	std	8	30.4	33.6	3.8	4.2	10.5
F	hi-mem	13	53.3	58.8	4.1	4.52	10.3
G	hi-mem	26	106	118	4.08	4.54	11.3

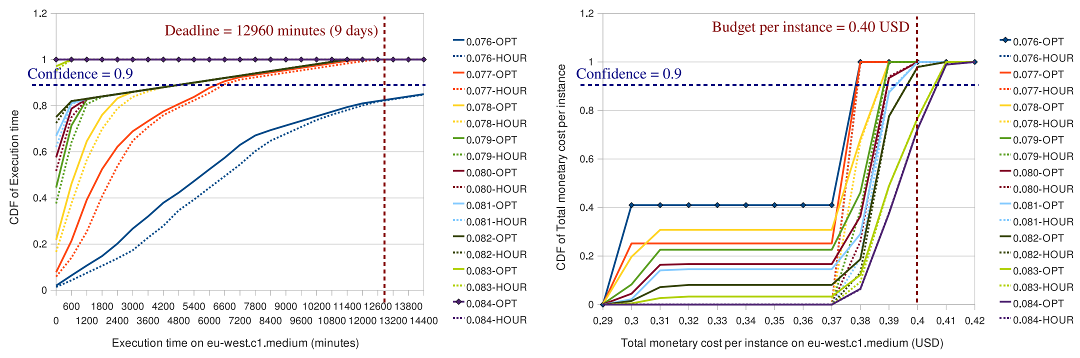
We also found that there is a big difference on the monetary costs between this case ($T = 164$ minutes) and a simulation for $T = 184$ minutes (not shown). This is explained by the fact that monetary cost is highly depending on Amazon's pricing policy, because the granularity of calculating price is an hour, and thus, if we exceed the hour-boundary we need to pay the last partial hour.

E. Comparing Instance Types

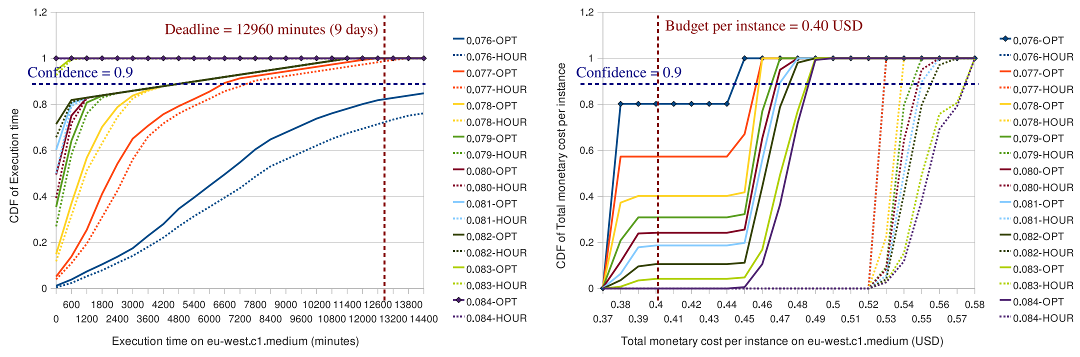
Table VII attempts to answer two questions: *what is the variation of the typical bid prices per instance type?* (i.e. *how much can we save by bidding low compared to bidding high?*) and *how much can we save by changing the instance type?* The first three columns are the same as in Table IV.



(a) when task length T = 216 minutes



(b) when task length T = 276 minutes



(c) when task length T = 336 minutes

Figure 7. CDF of execution time (ET, left) and monetary cost (M, right) for various task lengths on instance type A (workload W1)

The "typical" price range ["Low Bid", "High Bid"] has been determined on the price history from Jan. 11, 2010 to March 18, 2010; we plotted this history (as in Figure 1), removed obviously anomalous prices (high peaks or long intervals of constant prices), and took the minimum L ("Low Bid") and maximum H ("High Bid"). The last column ("Ratio in %") shows $(H - L)/L * 100$ per instance type, i.e. the range of bid prices divided by "Low Bid" (in %). This answers the first question: the variation of the typical bid prices is only about 10% to 12% accross all instance types.

In Table VII, column "Low / Unit" shows the "Low Bid" price divided by the total number of EC2 Computing Units (units) of this instance type. The column "High / Unit" is computed analogously. For workload types assumed here, this allows one to estimate the cost of processing one unit-hour (in US-cent) disregarding the checkpointing and failure/recovery overheads. Obviously, instance types within the high-CPU class [4] have lowest cost of unit-hour - only about 40% of the standard class. For the high-memory instance types a user has to pay a small premium - approx. 8% more than for the standard class. Interestingly, all instance types within each class have almost identical cost of one unit-hour. In summary, switching to a high-CPU class (if amenable to the workload type) can reduce the cost of unit-hour by approx. 60% while bidding low saves only 10% of the cost, with a potentially extreme increase of the execution time.

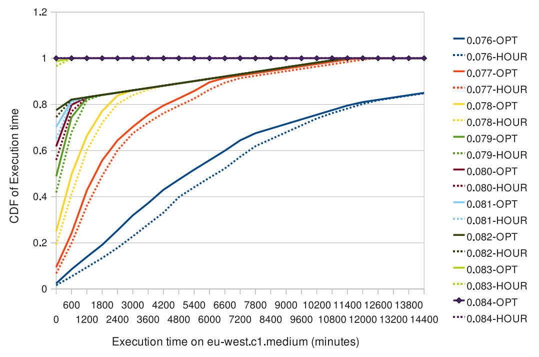
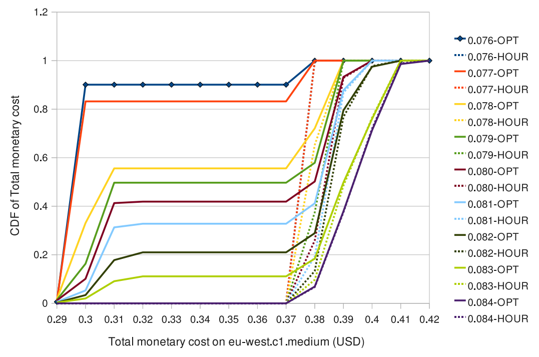



Figure 9. CDF of execution time (ET, left) and monetary cost (M, right) for task length of $T = 246$ minutes on instance type A (workload W1)

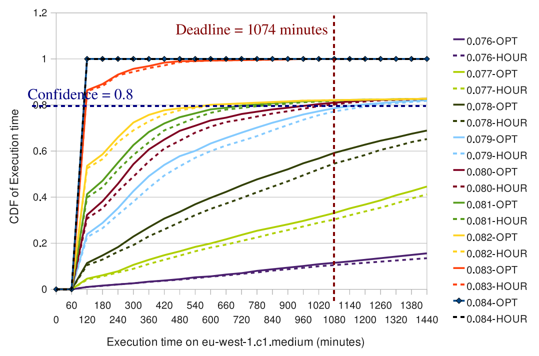
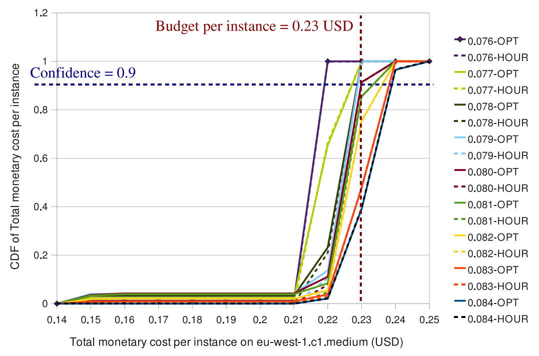



Figure 10. CDF of execution time (ET, left) and monetary cost (M, right) for task length of $T = 164$ minutes on instance type A (workload W2)

F. Summary

We have shown in this Section several interesting findings of potential value for users deploying Spot Instances:

- Bidding low prices reduces the monetary cost typically only by about 10% but can lead to extremely high execution times (or, equivalently, realistic deadlines) - up to 400x the task length (Section IV-B1).
- The execution times (or equivalently, realistic deadlines) increase rapidly especially for confidence values (on the deadline) above 0.9; on the other hand, lowering the confidence value by only 0.1 (to 0.8) can lead to substantial cost savings (Table V, Section IV-C).
- Similar to the effect of increasing bid price, increasing the budget slightly can reduce the execution time by a large factor (Table VI, Section IV-D).
- The coefficient of variation (standard deviation / mean) of execution time decreases sub-linearly with the task length. In other words, with longer task lengths the execution time becomes more predictable (Section IV-B3).
- The availability ratio and usage ratio (see Section III-B) depend on the task length, especially for low bid prices (Section IV-B3).
- The time overhead of checkpointing, failures and recovery becomes significant only for low bid prices and very high confidence values; in general, the hourly checkpointing strategy is efficient in terms of time and monetary cost of overhead (Figure 4, Section IV-B2).
- Selecting an instance type from the high-CPU class can yield cost savings up to 60% (for the considered workload type) compared to other classes without any increase of the execution time. If possible, this measure should be preferred over bidding low prices.

V. Related Work

Branches of related work include cloud computing economics and resource management systems. With respect to cloud economics, several previous works focus on the performance and monetary cost-benefits of Cloud Computing compared to other traditional computing platforms, such as Grids, Clusters, and ISPs [11], [12], [13], [14], [15]. These economic studies are useful for understanding the general economic and performance trade-offs among those computing platforms. However, the same studies do not address the specific and concrete decisions an application scientist must make with respect to bid price and resource allocation when using a market-based Cloud platform, such as Spot Instances.

With respect to resource management systems, several batch schedulers such as OAR [16], and DIRAC [17] exist for the management of job submissions and deployment of resources. Some of these batch scheduler even enable the deployment of applications across Clouds (such as EC2) dynamically. However, these resource management systems do not provide any guidance to users submitting jobs, which is critical to making decisions under SLA and budget constraints.

Specifically, with the the advent of market-based resource allocation systems, new trade-offs in performance, reliability, and monetary costs exist. But these resource management systems currently are too rigid and do not take into account these new variables such as bid prices and the ability to request an arbitrary number of instances; these new variable are often not present in traditional computing systems. Even companies, such as RightScale Inc. [18], that provide monitoring and scheduling services for the Cloud do not guide the application scientists in this respect, to the best of our knowledge.

VI. Conclusions and Future Work

Market-based resource allocation is becoming increasingly prevalent in Cloud Computing systems. With Spot Instances of Amazon Inc., users bid prices for resources in an auction-like system. A major challenge that arises is how to bid given the user's SLA constraints, which may include resource availability and deadline for job completion. We formulated a probabilistic model that enables a user to optimize monetary costs, performance, and reliability as desired. With simulation driven by real price traces of Amazon's Spot Instance and workloads of real applications, we evaluated and showed the utility of this model.

Our specific recommendations and general implications of this model are as follows:

- Users can achieve largest cost savings (for considered workload types) by using the high-CPU instance types instead of standard or high-memory instance types.
- Bidding low prices typically yields cost savings of about 10% but creates extremely large realistic deadlines (or, expected execution time) - up to factor 100x of the task length. This is especially the case in conjunction with high confidence values on the deadline.
- With growing task time, the variance of the execution time decreases.
- A user can change several of these "knobs" (parameters) in order to achieve a suitable balance between monetary cost and desired service levels, such as deadline for job execution or average availability. Our model indicates how to tune these different parameters and the effects.

For future work, we would like to determine if our model can be generalized for other types of applications. We would also like to study the optimization problem when allowing for the mixing of instance types (in terms of size or availability zones, for example), the dynamic adjustment of the number of instances during run-time, and rebidding and restarting the computation at this different bid price. Also we are currently developing mechanisms to predict forthcoming bid prices and unavailability durations. This would be helpful to reduce the checkpointing and rollback overhead during task execution. Finally, we plan to offer our model as a (web) service to help users in improving their bidding strategies.

Acknowledgments

This work is carried out in part under the EC project eXtreemOS (FP6-033576) and the ANR project Clouds@home (ANR-09-JCJC-0056-01).

Reproducibility of Results

All data used in this study, the full source code of the simulator and additional results are available under the following URL:

http://spotmodel.sourceforge.net

References

[1] M. Stokely, J. Winget, E. Keyes, C. Grimes, and B. Yolken, "Using a Market Economy to Provision Compute Resources Across Planet-wide Clusters," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'09)*, 2009.

[2] J. Hamilton, "Using a Market Economy," http://perspectives.mvdirona.com/2010/03/23/UsingAMarketEconomy.aspx.

[3] Amazon EC2 Spot Instances, *http://aws.amazon.com/ec2/spot-instances/*, 2010.

[4] Amazon EC2 Instance Types, *http://aws.amazon.com/ec2/instance-types*, 2010.

[5] Amazon Simple Storage Service FAQs, *http://aws.amazon.com/s3/faqs/*, 2010.

[6] S. Yi, D. Kondo, and A. Andrzejak, "Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud," March 2010, submitted.

[7] Y. Yang and H. Casanova, "Umr: A multi-round algorithm for scheduling divisible workloads." in *IPDPS*, 2003, p. 24.

[8] "Catalog of boinc projects," http://boinc-wiki.ath.cx/index.php?title=Catalog_of_BOINC_Powered_Projects.

[9] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. J. Epema, "The grid workloads archive," *Future Generation Comp. Syst.*, vol. 24, no. 7, pp. 672–686, 2008.

[10] A. Iosup, O. O. Sonmez, S. Anoep, and D. H. J. Epema, "The performance of bags-of-tasks in large-scale distributed systems," in *HPDC*, 2008, pp. 97–108.

[11] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson, "Cost-benefit analysis of cloud computing versus desktop grids," in *18th International Heterogeneity in Computing Workshop*, Rome, Italy, May 2009. [Online]. Available: http://mescal.imag.fr/membres/derrick.kondo/pubs/kondo_hcw09.pdf

[12] A. Andrzejak, D. Kondo, and D. P. Anderson, "Exploiting non-dedicated resources for cloud computing," in *12th IEEE/IFIP Network Operations & Management Symposium (NOMS 2010)*, Osaka, Japan, Apr 19–23 2010.

[13] M. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon S3 for Science Grids: a Viable Solution?" in *Data-Aware Distributed Computing Workshop (DADC)*, 2008.

[14] S. Garfinkel, "Commodity grid computing with amazons s3 and ec2," in *login*, 2007.

[15] E. Deelman, S. Gurmeet, M. Livny, J. Good, and B. Berriman, "The Cost of Doing Science in the Cloud: The Montage Example," in *Proc. of Supercomputing'08, Austin*, 2008.

[16] N. Capit, G. D. Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounié, P. Neyron, and O. Richard, "A batch scheduler with high level components," in *CCGRID*, 2005, pp. 776–783.

[17] V. Garonne, A. Tsaregorodtsev, and I. Stokes-Rees, "DIRAC: Workload Management System," *Computing in High Energy Physics and Nuclear Physics 2004, Interlaken, Switzerland, 27 Sep-1 Oct 2004*, p. 1041.

[18] RightScale: Cloud Computing Management Platform, *http://www.rightscale.com/*, 2010.