Capri: Achieving Predictable Performance in Cloud Spot Markets

Bogdan Ghiț
Databricks Inc.
Amsterdam, the Netherlands
bogdan.ghit@databricks.com

Asser Tantawi
IBM T.J. Watson Research Center
Yorktown Heights, NY, USA
tantawi@us.ibm.com

***Abstract*—Large cloud providers offer spot instances at attractive prices to improve resource utilization, resulting in a spot market where users bid for resources and providers alter prices dynamically. As prices surpass bid values, resources may be relinquished from users with low bids. Achieving predictable performance on spot markets is challenging for data analytics workloads because they are very sensitive to preemptions due to the excessive cost of recomputations.**

We introduce CAPRI, a scheduling system for running cloud data analytics in spot markets in which users may experience periods of degraded performance. CAPRI dynamically predicts the functional relationship between bid and performance, thus helping with managing expectations and bid advice. We propose a new spot market abstraction called the bribe scheduler which delivers differentiated service levels based on bids. CAPRI uses a prediction mechanism built on a queueing approximation of the bribe scheduler. CAPRI dynamically estimates parameters to adapt the queueing model and provide accurate performance predictions in the face of time-varying workloads.

We collect measurements using CAPRI running two realistic workloads, IMDB and TPCDS, and demonstrate the accuracy of our approximation and parameter estimation methodology. We show that CAPRI achieves a median prediction error below 3% in bursty workloads. We find that CAPRI's service level prediction is pessimistic as users are likely to experience better performance than they should receive for their bids.

I. Introduction

Data analytics applications are powered by a diverse array of computing platforms from clusters to clouds, and more recently, to spot markets [10], [17], [31]. While attractive for offering inexpensive servers, spot markets often fail to deliver predictable performance [4]. As spot prices fluctuate, applications are exposed to interruptions which may result in long delays, forcing the user to wait either for the price to drop or to recover lost state [5], [22], [29]. Because such delays are difficult to anticipate, users may feel discouraged and abandon the spot market [19]. In this paper we present CAPRI, a spot market scheduling system that adaptively determines the functional relationship between bid and performance for data analytics applications, thus enabling users to anticipate with good accuracy their service levels on the spot market.

As depicted in Figure 1 Amazon EC2 offers spot resources with interruption rates that are usually higher than 20% irrespective the instance type and availability zone. Two common solutions for running data analytics on a spot market are either to place large enough bids and reduce the risk of

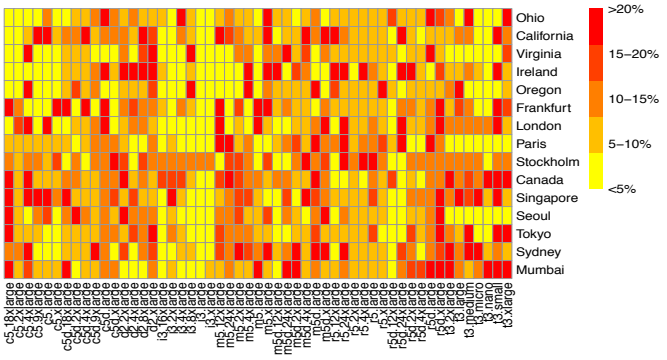


Fig. 1: A heat-map depicting high frequency of instance interruptions across all instance types in each availability zone from the AWS spot advisor in the trailing month.

preemptions [23], [33], or to periodically save the application state to avoid losing work already done [22], [29]. Not only these solutions are cost ineffective, but they also don't provide any performance expectation to users.

We seek to provide *differentiated service levels* to users based on their bids by means of a new spot market abstraction called the *bribe scheduler*. The concept of bribing for compute resources has been previously studied in the context of a single-server queue with users that buy their relative priority by means of a bribe or bid to gain access to a server [15]. Whereas in this model users are granted exclusive access to the server, in our cloud setting we aim at sharing resources across multiple applications at the same time. Therefore, we encapsulate the application runtime system on a set of *spot containers* that are prioritized by the bribe scheduler based on the user bids. To discriminate users based on their bids, the bribe scheduler employs preemption so that only the containers that have the highest bids are in service at any time. Unlike existing cloud spot markets, our bribe scheduler re-queues preempted containers instead of completely aborting them. A queued container may be deployed again whenever resources become available.

We want to be able to anticipate the relationship between bid and performance for data analytics applications. To achieve this goal, we design a *spot advisor* that is motivated by first principles analysis within the context of a theoretical model

for studying bribe scheduling. To assess the performance of a job in our analysis, we use the average job slowdown defined as the ratio between the turn-around time of the job and its uninterrupted runtime. We generalize the closed-form formula of the job slowdown as a function of the bid from a single-server system with rigid jobs to a multi-server system with jobs consisting of multiple inter-dependent tasks. To do so, we incorporate model parameters and we take an adaptive approach for dynamically estimating those parameters by employing an extended Kalman filter on the slowdown and bid values measured over a period of time.

The main contributions of this paper are as follows:

1) We design CAPRI, a scheduling system for cloud data analytics that achieves differentiated service levels and dynamically estimates the functional relationship between bid and performance.
2) We deploy CAPRI on a public cloud and show that it anticipates with high accuracy the job slowdown as a function of bid and delivers better performance than what users should expect for their bids.

II. System Model

In this section we describe the scheduling system and its model which provides differentiated service levels in a cloud spot market based on bidding.

We consider data analytics applications running in a multi-resource environment. Such applications are typically decomposed into multiple tasks that run on workers that are deployed on units of the available capacity called *compute slots*. A worker runs in a container and is responsible for the execution of the tasks assigned to it by the task scheduler [1]. Figure 2 depicts the scheduling of jobs in this environment. An incoming job requests a given number of containers and is placed in a queue where jobs are ordered based on their bid values, so that lower bid values are in the back of the queue. Jobs wait in the queue until they are allocated at least one container before they start receiving service. While *in service*, the number of allocated containers may grow and shrink depending on the container availability and system load. A partially executing job has only a fraction of requested containers allocated and is progressing with degraded performance. When all containers of a job are revoked, the job goes back to the waiting queue.

Let job arrivals constitute a Poisson process with rate λ and let X be the random variable representing the job bid value. Without loss of generality, we assume that the bid value is in the set $\mathcal{X} = [0, 1]$. The probability distribution function of X is denoted by $B(x) = Pr[X \leq x], x \in \mathcal{X}$. We assume that $B(x)$ is continuous and differentiable. Let $S(x)$ denote the job slowdown defined as the ratio between the average response time and the average service time. An approximate expression for the job slowdown $S(x)$ is given by [11]:

$$S(x; B, \Theta) = \frac{1}{[1 - \theta_0(1 - B(x))^{\theta_1}]^2}, \quad (1)$$

[1]We use the terms worker and container interchangeably throughout the paper.

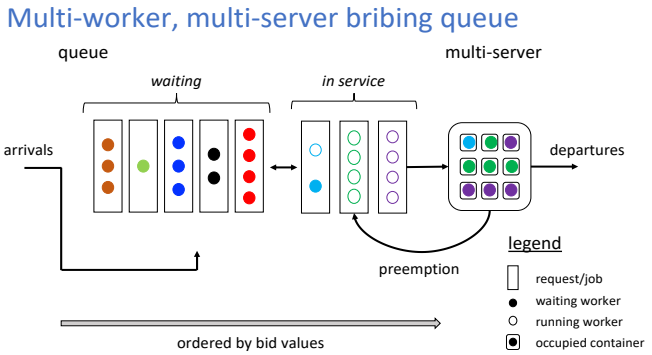


Fig. 2: An overview of our queueing system. Jobs and containers are drawn as rectangles and circles within the rectangles, respectively. The system has a certain container capacity, represented as squares. A circle within a square represents a worker assigned to a container.

where Θ is a two-parameter vector, $\Theta = [\theta_0, \theta_1]$, acting as scale and shape parameters, respectively, $0 \leq \theta_0 < 1$ and $\theta_1 > 0$. First, θ_0 acts as a (virtual) replacement for the server utilization, which may not be available to an external observer. Second, θ_1 captures additional model features, when compared to a simple M/M/1 bribe queue [15].

III. CAPRI Spot Market

We want to provision transient spot resources to data analytics applications while providing performance expectations to these applications. To achieve this, we assign to each application a relative priority in the scheduler queue equal to the bid value placed by the user at submission time. Our spot market scheduling system called CAPRI consists of a bribe scheduler that provides differentiated service levels based on bids and a spot advisor that incorporates the relationship between the job slowdown and bid value described in Section II. Figure 3 depicts at a high-level the functional components and interfaces needed by CAPRI to operate in a real environment.

A. Job Management

We present the typical interaction between users and the CAPRI spot market through the job management framework. The data analytics job model assumed by CAPRI requires configuring a specialized runtime system which consists of a *driver* process that coordinates a set of *workers* used to execute the job. The driver and worker processes run in isolation inside containers which are allocated on compute and memory resources as specified by the user.

The driver generates an abstract representation of the user program that we call the *logical plan* of the job. In particular, the driver decomposes the job into multiple tasks some of which are *independent* and may run in parallel as they operate on a different data partition of the same dataset (map tasks) and others which may have *data dependencies* among them (map tasks before reduce tasks). In this way, we obtain a *direct-acyclic graph of tasks* (DAG) which is the logical plan of the job. The driver has an internal DAG-aware scheduler

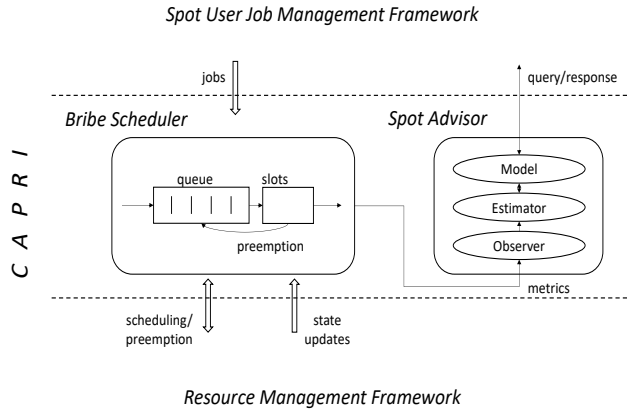


Fig. 3: An overview of the functional components and interfaces employed by CAPRI.

that further generates a *physical plan* of the job which is a mapping of tasks to compute slots.

CAPRI enables users to compete for the available resources through *bidding*. The bid is set upfront by the user and represents the cost per compute slot that the user will pay as long as its job is running excluding any waiting time that the job may experience. Because the amount of spot resources is limited, CAPRI may revoke a fraction of the resource quota allocated to a given user if there are higher bidders in the system. This is a key difference from existing spot markets where the user is charged based on a spot price that is completely under the control of the cloud provider.

B. Bribe Scheduler

To provide differentiated service levels to multiple users, CAPRI defines a new abstraction of a cloud spot market called the *bribe scheduler*. The bribe scheduler is a materialization of the scheduling policy analyzed in Section II. The bribe scheduler registers with a cluster-wide resource manager from which it receives a *spot offer* – a list of free resources on multiple (virtual) machines that CAPRI can use to serve incoming user requests.

We employ two complementary mechanisms that target a subset of the resource requests of existing spot users, and operate at different timescales. On the arrival of a job, CAPRI places the individual resource requests for containers in a *waiting* queue which keeps these requests ordered by the user bid with the driver always placed ahead of its workers so that it is scheduled first. As long as there are idle spot resources, CAPRI schedules the request at the head of the queue and *allocates* an isolated bundle of its resource request. The job can use this bundle to deploy either a driver if it doesn't already have a running one or a worker otherwise.

We ensure that no waiting request has a higher bid than any other user request in service by means of *preemption*. When the spot offer is fully utilized and an arriving request places a higher bid than some of the user requests that are currently serviced, CAPRI may preempt multiple requests with lower bids in order to make room for the new request. Preempted requests are placed back in the waiting queue where they will wait for their turn as the higher bidders complete their work and leave the system.

CAPRI seeks to guide its scheduling decisions only on the user bid, and so it may not be able to service all resource requests of a given job at the same time. Because the job model assumed by CAPRI is elastic, it can run on as many resources as it can get. The job cannot run without a driver and it will immediately release all its workers when the driver is revoked. However, a job may continue running even if (a subset of) its workers are revoked by CAPRI. As a consequence, the work already done and lost due to preemptions needs to be rescheduled by the driver and restarted from scratch on the remaining set of workers.

CAPRI may fully or partially preempt the resource requests of a job multiple times until it completes. However, in order to limit their costs, users can control the lifetime of their jobs by setting a maximum number of driver restarts. When that limit is reached, the user will abandon the spot market even though its job is incomplete. In addition, users may guard their jobs from failures through a periodic checkpointing mechanism that can be incorporated into the runtime system of the job [29]. Deciding how and when to checkpoint their jobs based on the volatility of the spot market is however orthogonal to the work presented in this paper.

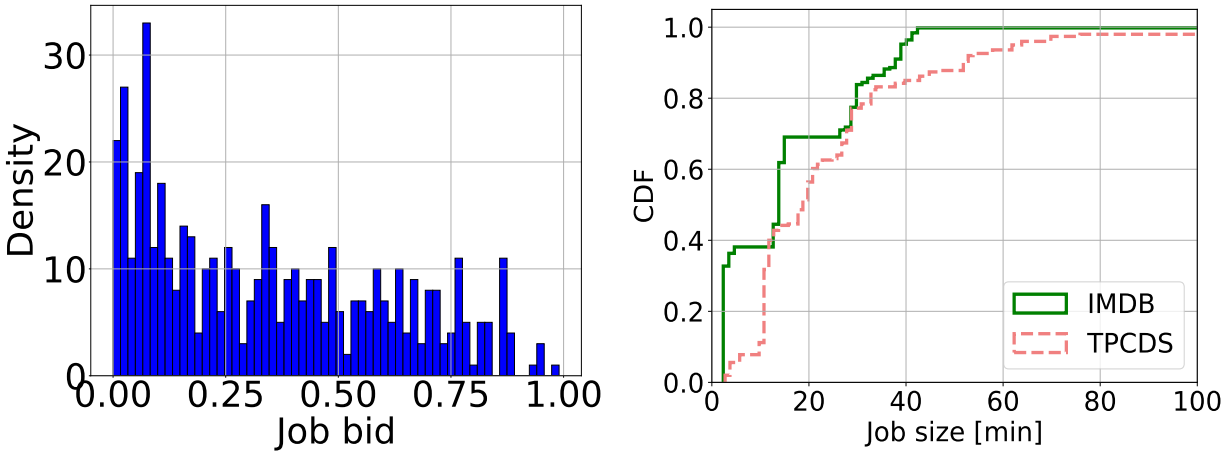
C. Spot Advisor

The key feature we propose is the *spot advisor* which is a mechanism that allows users to get insights into their expected level of performance on the spot market. CAPRI incorporates the relationship between the job slowdown and bid value which we have obtained in our analysis of the bribing queue in Equation 1. In order to adapt the model parameters to workload changes, CAPRI collects samples of the user bid, job runtime, and response time. While the bid is determined at submission time, the latter two samples are collected once the job completes. Using these samples, CAPRI dynamically estimates and updates the model parameters over time using an extended Kalman filter as described in [11].

Independent of collecting samples and updating the model parameters, CAPRI allows users to query the spot advisor. In particular, CAPRI can answer two types of queries. When the user provides a bid value, we can predict the average slowdown by replacing in Equation 1 the fraction of previous users that had a lower bid than the current user. Furthermore, we can provide a bid advice to a user that sets an expectation for the desired level of service. To so so, we employ Equation 1 from which we can obtain the user bid as a function of the job slowdown by simply inverting the function.

D. Resource Allocator

To operate the spot market, CAPRI assumes ownership over a set of machines offered by a cluster-wide resource manager. CAPRI uses those machines to confine the runtime systems of the incoming jobs in isolated bundles of resources such as containers. Such resource bundles are specified in terms of a



(a) (b)

Fig. 4: The distribution of bid values (a) and the cumulative distribution function of the job size (b) for both the IMDB and TPCDS workloads.

quota of compute slots and memory. In this section we present the job isolation properties of CAPRI.

CAPRI provides performance isolation between different jobs by leveraging container isolation mechanisms. Using containers enables fine-grained resource-sharing, and so we can confine the resource usage of a process tree to any amount of compute slots and/or memory. Whereas other isolation mechanisms exist, containers are attractive for CAPRI because they simplify the deployment of a job runtime system. In particular, the user only needs to set the instructions that CAPRI can employ to generate a container image which is a portable file that can be further used to instantiate containers.

CAPRI delegates the task of allocating resources at the granularity of containers to a container management platform. Besides creating and preempting containers, we also want CAPRI to be able to monitor the status of its containers in order to get accurate measurements of the job performance. However, because providing isolation is platform dependent, we make the resource allocation module pluggable. Therefore, we maintain a low-level interface that enables CAPRI to delegate the (de-)allocation and monitoring requests to the underlying container management platform.

IV. Experimental Setup

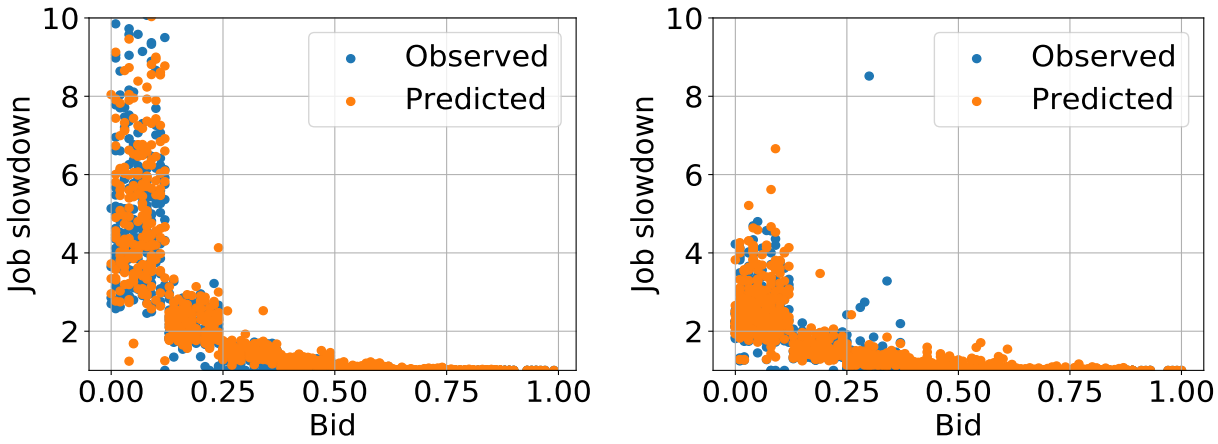
In this section, we present the configuration of our cloud deployment and the data analytics workloads we use in our evaluation of CAPRI. Our experimental setup consists of a Kubernetes cluster with six Amazon EC2 `t3.2xlarge` virtual machines each of which is configured with 8 vCPU slots, 32 GiB memory, and a network performance of up to 5 Gbps.

We use two workloads which consist of mixes of queries from the IMDB [16] and TPCDS [20] benchmarks. The IMDB benchmark includes 111 different queries with input data from 21 tables with a total size of 3.6 GB stored in CSV format. Similarly, the TPCDS benchmark has 104 queries with input data from 24 tables with a total size of 1 GB stored in Parquet format. Each workload consists of a stream of 1,000 jobs with a Poisson arrival process and an imposed average load of 0.9. A job request consists of five containers with 1 vCPU and 1 GiB each and may be preempted at most four times before abandonment. The bid values in both workloads are sampled from a synthetic geometric distribution with $p = 0.1$ which is dominated by relatively low bids with a median value of 0.3 as shown in Figure 4a.

Figure 4b depicts the job size distribution in the IMDB and TPCDS workloads. On average, the jobs sizes in the TPCDS workload are twice as large as in the IMDB workload. The IMDB workload is dominated by short-interactive queries with more than 40% of all jobs taking less than 5 minutes and very few long-running jobs. In particular, roughly 20% of the jobs in the IMDB workload account for almost 50% of the total load. In contrast, the TPCDS workload is less variable than IMDB, with more than half of the jobs having sizes between 20 and 140 minutes. Running the IMDB and TCPDS workloads on CAPRI in our setup took 9.66 h and 15.91 h, respectively.

We implemented our CAPRI scheduling system using Spark on Kubernetes. To run Spark on Kubernetes we employ a job submission framework which bundles the Spark runtime system inside Kubernetes containers [2]. The job submission framework receives the user request for running an application with a certain bid and number of containers. The framework creates a Spark driver which in turn launches multiple workers all of which run inside Kubernetes containers. The driver and worker container scheduling are handled by our CAPRI scheduling system which maintains a waiting queue ordered by the container bid. In order to control the life-cycle of containers, CAPRI uses the Kubernetes Java Client [1], which provides access to the Kubernetes API for creating and deleting bindings between containers and cluster resources.

We packaged the spot advisor in an external library that dynamically tracks the model parameters as it learns of new data collected when jobs complete, namely the total queueing time and the wall-clock time during which the job receives



(a) IMDB **(b)** TPCDS

Fig. 5: Scatter plots depicting the observed and predicted job slowdowns for the complete range of bids. Jobs may suffer large slowdowns if they bid low less than 0.1, be lucky with relatively low slowdowns if their bids are between 0.1 and 0.25, or achieve on-demand performance if their bids are higher than > 0.5.

service from the system. To integrate the spot advisor with our scheduling system we expose a simple API that can be used either for updating job samples or to get a slowdown prediction given a bid value. Whereas the job submission framework is specific to Spark applications, both the bribe scheduler and the spot advisor are agnostic to the application type.

V. Experimental Results

In this section we analyze different aspects of CAPRI's operation in the face of time-varying workloads such as the prediction accuracy, cost savings, and preemption sensitivity.

A. Prediction Accuracy

We first evaluate CAPRI's ability to deliver differentiated service levels based on the submitted bids and to anticipate the job slowdown attained given a bid.

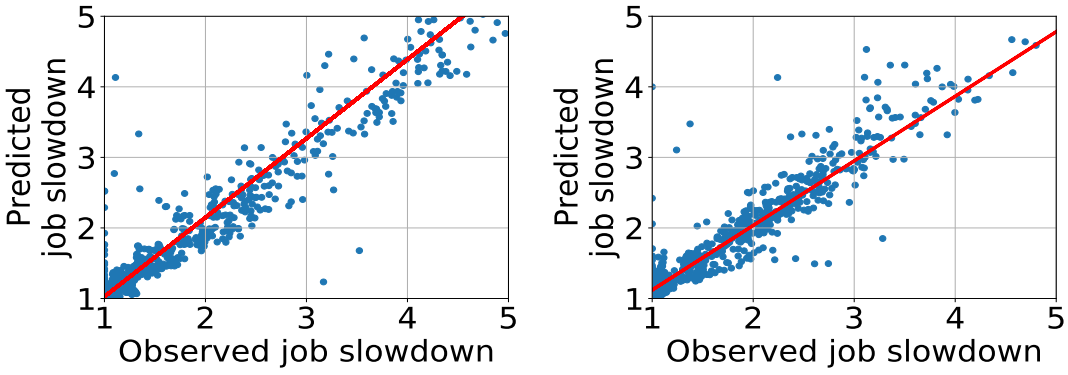
Figure 5 compares the observed and predicted job slowdowns for the complete range of bid values with both workloads. Jobs that bid higher than 0.5 experience close to ideal slowdowns. As expected, jobs that bid below 0.25 are more likely to experience large slowdowns because they face a higher risk of preemption. However, we observe that the range of slowdowns is much wider in IMDB than in TPCDS. The IMDB workload is dominated by short-interactive jobs that take in the order of minutes and so, jobs that experience delays due to other jobs with higher bids typically have slowdowns much higher than 2. In contrast, the job sizes in the TPCDS workload are larger and more homogeneous, which means their slowdowns are less sensitive to delays. Jobs with relatively low bids may still be lucky and experience good service levels during periods of lower utilization. This is the case for the cluster of jobs in TPCDS that bid below 0.25 and experience slowdowns between 1 and 2. In contrast, such lucky jobs are less frequent in IMDB because the system has a higher utilization when running this workload.

Figure 6 shows the relationship between the predicted and observed job slowdowns. We use the R-squared value, also called coefficient of determination, to assess how close CAPRI's job slowdown prediction is to an ideal prediction. The R-squared value ranges between 0 and 1 and provides a good measure of how well the observed job slowdowns are replicated by CAPRI's prediction model. High R-squared values that are close to 1 denote a strong correlation between the observed and predicted values. An important observation is that job slowdowns in IMDB are more predictable than in TPCDS. The system load is more stable and less sensitive to job preemptions in IMDB. These short jobs are hit by preemptions relatively late in their execution and so, the amount of work they waste and need to recompute compensates for the decrease in utilization caused by jobs that reach the preemption threshold and get abandoned. In contrast, job preemptions in TPCDS result in larger fractions of work that is abandoned, thus decreasing more drastically the utilization of the system.

In Figure 7 we depict the distribution of the relative prediction error delivered by CAPRI with each workload. CAPRI anticipates the service levels of half of the jobs with less than 3% prediction accuracy for both workloads. Moreover, CAPRI's prediction error is very dense in the range between 0 and 10% and very rarely exceeds 30%, which confirms our model is accurate and robust. We also see that the distribution of relative errors is more dense on the positive side, which means that CAPRI tends to overestimate the job slowdown. Thus, on average jobs may expect to experience lower slowdowns than their anticipated levels of performance.

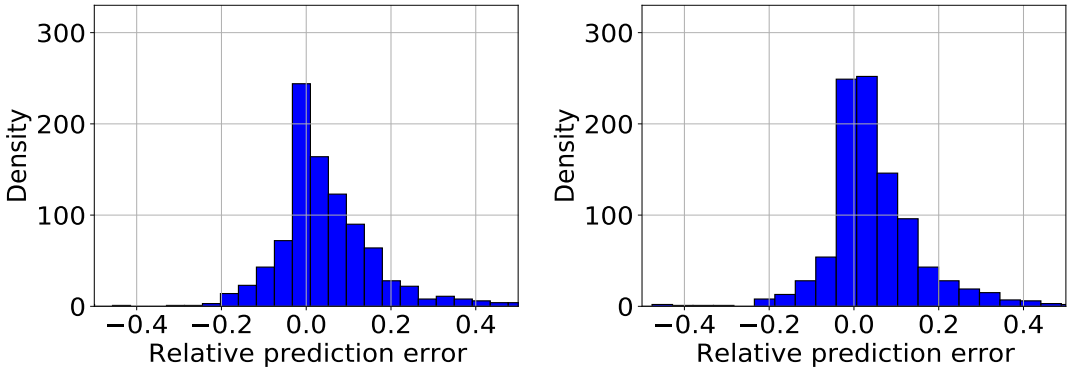
B. Cost Savings

To analyze the cost of using CAPRI we investigate the evolution of the spot price over time. Unlike AWS spot markets, CAPRI is a free market in which users are charged based on their bids. Thus, in our experiments with CAPRI,



(a) IMDB **(b)** TPCDS

Fig. 6: The correlation between the observed and predicted job slowdowns. CAPRI delivers good prediction accuracy of the job slowdown as most jobs are very close to the regression line. The R-squared values are 0.91 and 0.82 for IMDB and TPCDS, respectively.



(a) IMDB **(b)** TPCDS

Fig. 7: The density of the relative error in job slowdown prediction with CAPRI for each workload.

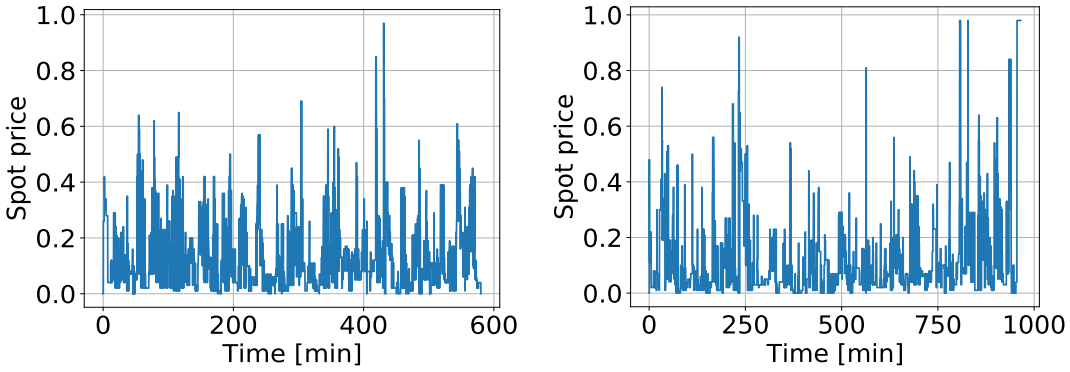
the spot market price is determined by the current system load and the user bids, rather than a hidden process that dynamically changes it periodically as in the case of AWS spot markets [4]. As we see in Figure 8, the spot price on CAPRI mostly fluctuates within the range between 0 and 0.3 and rarely increases above 0.3, which is the median bid value in our workloads, even though we operate the market under high system loads. When compared to the cost of the on-demand uninterrupted execution, CAPRI achieves an 80% and 65% cost reduction for IMDB and TPCDS, respectively.

C. Sensitivity Analysis

In all our previous experiments we have set the maximum number of restarts before a job is abandoned by CAPRI to 4. To understand the impact of the preemption-restart mechanism, we perform a sensitivity analysis of the number of retries a job may attempt after preemptions. To this end, we run a set of micro-experiments using 10% of all jobs in each workload. We set the number of job restarts between 1 and 16. Any job that is preempted more times than the predefined number of job restarts is completely abandoned by CAPRI.

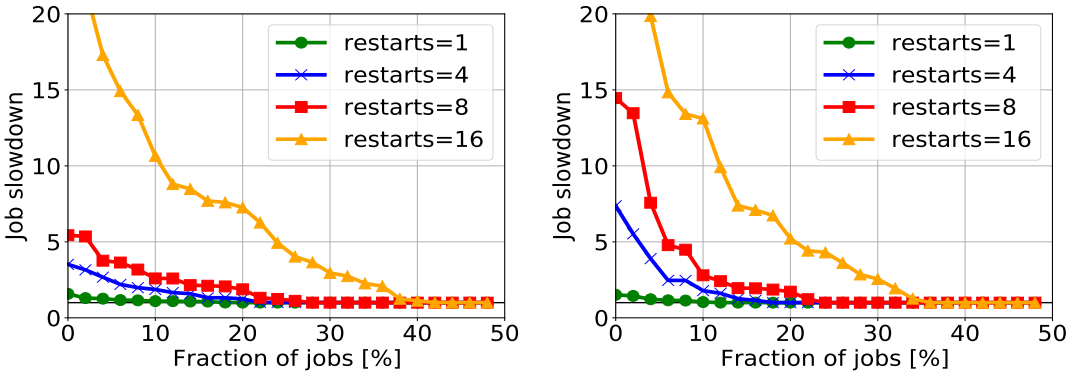
Figure 9 depicts the slowdown achieved by jobs at different percentiles in the job slowdown distribution when the number of job restarts increases from 1 to 16 for each workload. We find that CAPRI is not overly sensitive to the number of restarts a job may attempt. We can see that in both workloads only 10% of all jobs have slowdowns that are higher than 2.5. More importantly, roughly 80% of all jobs experience close to ideal slowdowns when the number of retries is less than 8. Setting a lower number of retries per job increases the chance of job abandonment. Without any retries like on AWS spot markets, we have roughly 25% of all jobs that are completely abandoned in both workloads. However, setting maximum 16 retries per job reduces the number of abandoned jobs in CAPRI to only 4.

Allowing jobs to retry multiple times increases the amount of work that is lost due to preemptions and needs to be recomputed when jobs get restarted. Having extra work to recompute because of the preempt-restart mechanism adds more pressure on CAPRI which may lead to congestion and starvation of jobs that bid relatively low. When we set the number of retries to 4, the samples of the IMDB and TPCDS



(a) IMDB **(b)** TPCDS

Fig. 8: The evolution of the spot price for the duration of each the workload.



(a) IMDB **(b)** TPCDS

Fig. 9: The job slowdown distribution for different number of job restarts. AWS spot markets allow no job restarts.

workloads impose an average system utilization of 0.86 and 0.77, respectively. Because the system is almost never idle for the duration of the experiments the likelihood of having queueing delays is relatively high, hence the few jobs that have slowdowns above 2.

VI. Related Work

In this section we summarize the related work from three aspects: auction mechanisms for resource allocation on the spot market, bidding strategies for spot resources, and dynamic parameter estimation techniques.

Auction Mechanisms. Running auctions to sell unused cloud resources may improve system efficiency and/or the provider revenue. To this end, efficient resource provisioning policies have been extensively studied in prior work from a theoretical perspective. In order to discriminate users accessing their services, cloud providers can employ two main mechanisms: priority-based allocation and preemption based on user bids [3], [15], [18]. Departing from the single-server queues, other spot market designs have advocated for modeling auctions with heterogeneous instances rather than the previous type agnostic approaches [25]. The analysis of the online auction problem in this setting resulted into a mechanism that is optimal with respect to system efficiency across the temporal domain and is also able to dynamically provision heterogeneous resources.

Another practical approach to auctions that allows users to bid for their resources and bundles heterogeneous machines while taking into account operational costs of servers, has been proposed to optimize the social welfare and the provider's net profit [32]. Cloud providers may also operate proportional allocation schemes in which a user receives a fraction of resources that is proportional to the bid. In this area, game theoretic techniques have been used for service provisioning [6], price anticipation [9], and the introduction of resource allocation methods [21]. Merkat [8] is a resource manager that employs weighted proportional allocation [21] to dynamically allocate cloud resources among applications.

Bidding the spot market. Since Amazon EC2 released its spot markets in 2009, a sizable body of research analyzed the operation of such systems in the cloud. The characterization and prediction of spot prices of the AWS spot markets [4] inspired the design of user bidding strategies that optimize cost while also achieving uninterrupted service. Such strategies can be derived either by means of statistical analysis of historical spot prices [14], [33] or through more advanced modelling techniques such as Markov chains [7], [30]. However, recent

work has shown that complex bidding strategies [12], [26] are often ineffective in practice because cloud providers allow users to place maximum bids while charging them for a much smaller price that constitutes the spot price [23].

Dynamic parameter estimation. Modeling and predicting the performance of MapReduce-based applications has been studied in various settings [13], [28]. Common approaches build an estimator by choosing a relationship between an output variable that needs to be predicted and several system properties that can be measured and used for prediction. Most of these solutions build the estimator with machine learning techniques that use large amounts of training data based on low-level application performance characteristics [24], [27].

VII. Conclusion

CAPRI is an alternative spot market to existing public clouds for containerized data analtytics that employs bribe scheduling to provide differentiated service levels to its users. CAPRI is designed from first principles in the context of an analytic performance model that estimates the functional relationship between performance and bid. With a control theoretical approach based on Kalman filtering, we dynamically tune this model using runtime performance and bid measurements. We incorporate our adaptive model in a spot advisor that CAPRI employs either to set an expectation for the performance of a job given a particular bid value or to suggest a minimum bid value required to attain a given service level.

Our evaluation shows that CAPRI rapidly adapts the performance model to workload changes and exhibits a strong relationship between the predicted and measured job slowdown performance. In particular, CAPRI achieves a median prediction error below 3% which in most cases is pessimistic. In CAPRI jobs are likely to experience better service levels than what they should expect given their bids, which results in an average cost reduction of 65% or higher. We also show that CAPRI is rather insensitive to the number of job restarts caused by preemptions.

References

[1] https://kubernetes.io/docs/reference/using-api/client-libraries.

[2] https://spark.apache.org/docs/latest/running-on-kubernetes.html.

[3] P. Afèche and H. Mendelson. Pricing and Priority Auctions in Queueing Systems with a Generalized Delay Cost Structure. *Management Science*, 50(7):869–882, 2004.

[4] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing Amazon EC2 Spot Instance Pricing. *ACM Transactions on Economics and Computation*, 1(3):16:1–16:20, 2013.

[5] P. Ambati and D. Irwin. Optimizing the Cost of Executing Mixed Interactive and Batch Workloads on Transient VMs. *ACM SIGMETRICS*, 2019.

[6] D. Ardagna, B. Panicucci, and M. Passacantando. A Game Theoretic Formulation of the Service Provisioning Problem in Cloud Systems. *ACM WWW*, 2011.

[7] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. N. Tantawi, and C. Krintz. See Spot Run: Using Spot Instances for MapReduce Workflows. *USENIX HotCloud*, 2010.

[8] S. Costache, N. Parlavantzas, C. Morin, and S. Kortas. Merkat: A Market-based SLO-driven Cloud Platform. *IEEE Cloud Computing Technology and Science (CloudCom)*, 2013.

[9] M. Feldman, K. Lai, and L. Zhang. A Price-Anticipating Resource Allocation Mechanism for Distributed Shared Clusters. *ACM EC*, 2005.

[10] B. Ghit and D. Epema. Better Safe than Sorry: Grappling with Failures of In-Memory Data Analytics Frameworks. *ACM HPDC*, 2017.

[11] B. Ghit and A. Tantawi. An approximate bribe queueing model for bid advising in cloud spot markets. In *Quantitative Evaluation of Systems (QEST)*. Springer International Publishing, August 2021.

[12] W. Guo, K. Chen, Y. Wu, and W. Zheng. Bidding for Highly Available Services with Low Price in Spot Instance Market. *ACM HPDC*, 2015.

[13] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A Self-tuning System for Big Data Analytics. *CIDR*, 11(2011):261–272, 2011.

[14] B. Javadi, R. K. Thulasiramy, and R. Buyya. Statistical modeling of spot instance prices in public cloud environments. In *IEEE Utility and Cloud Computing (UCC)*, pages 219–228. IEEE, 2011.

[15] L. Kleinrock. Optimum bribing for queue position. *Operations Research*, 15(2):304–318, 1967.

[16] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. How Good Are Query Optimizers, Really? *VLDB Endowment*, 9(3):204–215, 2015.

[17] H. Liu. Cutting MapReduce Cost with Spot Market. *HotCloud*, 2011.

[18] F. T. Lui. An Equilibrium Queuing Model of Bribery. *Journal of Political Economy*, 93(4):760–781, 1985.

[19] I. Menache, O. Shamir, and N. Jain. On-Demand, Spot, or Both: Dynamic Resource Allocation for Executing Batch Jobs in the Cloud. *IEEE Autonomic Computing (ICAC)*, 2014.

[20] R. O. Nambiar and M. Poess. The Making of TPC-DS. *VLDB*, 6:1049–1058, 2006.

[21] T. Nguyen and M. Vojnovic. Weighted Proportional Allocation. *ACM SIGMETRICS*, 2011.

[22] P. Sharma, T. Guo, X. He, D. Irwin, and P. Shenoy. Flint: Batch-Interactive Data-Intensive Processing on Transient Servers. *ACM EuroSys*, 2016.

[23] P. Sharma, D. Irwin, and P. Shenoy. How Not to Bid the Cloud. *USENIX HotCloud*, 2016.

[24] J. Shi, J. Zou, J. Lu, Z. Cao, S. Li, and C. Wang. MRTuner: A Toolkit to Enable Holistic Optimization for MapReduce Jobs. *VLDB Endowment*, 7(13):1319–1330, 2014.

[25] W. Shi, L. Zhang, C. Wu, Z. Li, and F. Lau. An Online Auction Framework for Dynamic Resource Provisioning in Cloud Computing. *ACM SIGMETRICS Performance Evaluation Review*, 42(1):71–83, 2014.

[26] S. Tang, J. Yuan, and X.-Y. Li. Towards Optimal Bidding Strategy for Amazon EC2 Cloud Spot Instance. *IEEE CLOUD*, 2012.

[27] S. Venkataraman, Z. Yang, M. J. Franklin, B. Recht, and I. Stoica. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. *USENIX NSDI*, 2016.

[28] A. Verma, L. Cherkasova, and R. H. Campbell. ARIA: Automatic Resource Inference and Allocation for MapReduce Environments. *ACM ICAC*, 2011.

[29] Y. Yan, Y. Gao, Y. Chen, Z. Guo, B. Chen, and T. Moscibroda. TR-Spark: Transient Computing for Big Data Analytics. *ACM SoCC*, 2016.

[30] M. Zafer, Y. Song, and K.-W. Lee. Optimal Bids for Spot VMs in a Cloud for Deadline Constrained Jobs. *IEEE CLOUD*, 2012.

[31] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving MapReduce Performance in Heterogeneous Environments. *USENIX OSDI*, 2008.

[32] X. Zhang, Z. Huang, C. Wu, Z. Li, and F. C. Lau. Online auctions in iaas clouds: Welfare and profit maximization with server costs. In *ACM SIGMETRICS*, 2015.

[33] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang. How to Bid the Cloud. *ACM SIGCOMM Computer Communication Review*, 45(4):71–84, 2015.