Optimizing the Cost of Executing Mixed Interactive and Batch Workloads on Transient VMs

Pradeep Ambati
University of Massachusetts Amherst
lambati@umass.edu

David Irwin
University of Massachusetts Amherst
irwin@ecs.umass.edu

ABSTRACT

Container Orchestration Platforms (COPs), such as Kubernetes, are increasingly used to manage large-scale clusters by automating resource allocation between applications encapsulated in containers. Increasingly, the resources underlying COPs are virtual machines (VMs) dynamically acquired from cloud platforms. COPs may choose from many different types of VMs offered by cloud platforms, which differ in their cost, performance, and availability. While *transient VMs* cost significantly less than on-demand VMs, platforms may revoke them at any time, causing them to become unavailable. While transient VMs' price is attractive, their unreliability is a problem for COPs designed to support mixed workloads composed of, not only delay-tolerant batch jobs, but also long-lived interactive services with high availability requirements.

To address the problem, we design TR-Kubernetes, a COP that optimizes the cost of executing mixed interactive and batch workloads on cloud platforms using transient VMs. To do so, TR-Kubernetes enforces arbitrary availability requirements specified by interactive services despite transient VM unavailability by acquiring *many more* transient VMs than necessary most of the time, which it then leverages to opportunistically execute batch jobs when excess resources are available. When cloud platforms revoke transient VMs, TR-Kubernetes relies on existing Kubernetes functions to internally revoke resources from batch jobs to maintain interactive services' availability requirements. We show that TR-Kubernetes requires minimal extensions to Kubernetes, and is capable of lowering the cost (by 53%) and improving the availability (99.999%) of a representative interactive/batch workload on Amazon EC2 when using transient compared to on-demand VMs.

CCS CONCEPTS

• **Software and its engineering → Cloud computing**.

ACM Reference Format:
Pradeep Ambati and David Irwin. 2019. Optimizing the Cost of Executing Mixed Interactive and Batch Workloads on Transient VMs. In *ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '19 Abstracts), June 24–28, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3309697.3331489

1 INTRODUCTION

Container Orchestration Platforms (COPs), such as Kubernetes [2], Mesos, and others, have evolved into de facto cluster "operating systems" by automating the deployment of distributed applications encapsulated in containers, and managing the allocation of resources between them. COPs manage clusters of tens of thousands of machines, and serve as the primary interface users interact with to harness cluster resources. Thus, COPs must support the availability and performance requirements of a wide range of applications, including long-lived interactive services and non-interactive batch jobs, while also maintaining high cluster utilization.

COPs were originally developed for managing a mostly static set of dedicated physical machines in data centers. However, increasingly, the resources that underly COPs are virtual machines (VMs) dynamically acquired from cloud platforms. These platforms offer many types of VMs under a variety of different contracts, which differ in their cost, performance, and availability. In particular, transient VMs are an increasingly popular VM type, since they typically cost 50-90% less than on-demand VMs. However cloud platforms reserve the right to reclaim transient VMs at any time to satisfy higher priority tasks. Thus, while transient VMs' low price is attractive, their unreliability makes them unsuitable for COPs that must support long-lived interactive services with high availability requirements. As a result, prior work focuses primarily on optimizing only batch workloads for transient VMs.

To address the problem, we design TR-Kubernetes, a transient-aware COP that supports both batch jobs and interactive services with high availability requirements at low cost using transient VMs. To do so, TR-Kubernetes enables interactive services to explicitly specify their capacity availability requirements. TR-Kubernetes's provisioning policy then selects a mix of different transient VMs, from among the hundreds offered by cloud platforms, that satisfies the capacity availability requirement with high probability. As we discuss, to enforce high availability using unrealiable transient VMs, TR-Kubernetes must acquire *many more* transient VMs than necessary most of the time. TR-Kubernetes automatically leverages the excess capacity to run batch jobs.

2 BACKGROUND

Container Orchestration Platforms. There are many publicly-available COPs that offer similar functionality and support diverse workloads on large, mixed-use clusters, including Kubernetes [2], Mesos (with Marathon), and Docker Swarm. These platforms not only manage the allocation of cluster resources, but also provide a rich set of functions for supporting distributed applications and tightly integrate with cloud platforms. A key assumption COPs make is that distributed applications that run on them can handle i) the failure or revocation of containers, and ii) the allocation of new replacement containers. Since TR-Kubernetes extends an existing COP in Kubernetes, it makes the same assumptions as above.

Transient Cloud VMs. Transient VMs are available temporarily for an uncertain amount time, as platforms may revoke them at any time with little warning. As discussed above, since COPs support revocations, they implicitly allocate transient VMs to lower-priority jobs, which are generally batch jobs. Each of the major cloud platforms—Google Cloud Platform, Microsoft Azure, and Amazon EC2—now offer a variant of transient VMs.

3 DESIGN

TR-Kubernetes's design relies heavily on existing functions built into Kubernetes, as well as other COPs. The primary difference is that TR-Kubernetes enables users to specify a *capacity availability requirement* for an aggregate amount of computational capacity for an interactive service. These include an offline tool that runs TR-Kubernetes's *provisioning algorithm* to generate service descriptions, which specify the transient VMs necessary to satisfy the capacity availability requirement. This service description is then submitted via the Kubernetes command-line tool.

Provisioning Algorithm. TR-Kubernetes enables users to specify an availability requirement for a specified capacity in their service description for an interactive task. In this work, we assume interactive services are stateless and leverage Kubernetes's built-in load balancer to distribute requests across VMs. Our provisioning policy addresses the problem of selecting transient VMs by jointly optimizing both cost and availability subject to the availability target. To do so, TR-Kubernetes maintains a table of price and availability estimates for each transient VM. Given the table, computing the aggregate availability of different capacities for a pool of transient VMs is non-trivial, especially if transient VM availability is highly correlated. Fortunately, our analysis on EC2 spot VMs showed that availability for spot VMs is not highly correlated.

Computing the Availability of a Target Capacity. Since transient VMs of a given type are either available or unavailable, our approach, described below, essentially computes the probability of all possible available/unavailable combinations, and then sums the probabilities of all combinations that yield a capacity $\geq C$. To do so, we first denote each transient VM's availability as p_i and its capacity as c_i. We can then represent a transient VM i as a polynomial $Q_i(x)$ of degree $n_i \times c_i$, where we have n_i for each transient VM i.

$$Q_i(x) = (1 - p_i)x^0 + p_i x^{n_i c_i} \tag{1}$$

Here, the exponents of x represent the capacity of transient VMs of type i, while their coefficients represent the probability that a certain capacity is available (either zero or $n_i \times c_i$). This polynomial representation indirectly represents the probability mass function (PMF) of transient VMs of type i.

To compute availability of a capacity C for a pool of N different transient VM types, with n_i of each type, we derive our representation of the PMF of the transient VM pool by simply multiplying the polynomials of each transient VM, as they are independent.

$$Q_{pool}(x) = \prod_{i=1}^{N} Q_i(x) \tag{2}$$

From this equation, we can compute the availability at a target capacity m ($m \leq C$) by simply adding the coefficients of x's exponents, where the respective exponent is $\geq m$, as these are the combinations of transient VMs that satisfy the capacity requirement.

Greedy Algorithm. We next outline how TR-Kubernetes selects transient VMs for the pool to minimize cost, while satisfying the target level of availability for the specified capacity. The problem is complex, since there are hundreds of transient VM types within each cloud region, and we may select multiple instances of any one transient VM. As a result, there are an exponential number of possible pools that satisfy the capacity availability requirement.

Our problem appears similar to a multi-dimensional unbounded knapsack problem, where the VMs are akin to items, ECUs and availabilities are akin to weight dimensions, VM pools are akin to knapsacks, and costs are akin to item value. However, there are two primary differences that prevent applying common techniques, such as dynamic programming, to the problem. First, in our problem we do not know know the final number of ECUs (knapsack size) required for a given target availability and secondly, availability dimension in our problem is not strictly additive.

Thus we employ a greedy approach, which greedily selects transient VMs one by one until the pool satisfies the specified capacity availability requirement, or its cost exceeds the cost of using on-demand VMs to satisfy the requirement, in which case TR-Kubernetes requests on-demand VMs.

4 EVALUATION SUMMARY

We evaluate TR-Kubernetes at small scale on EC2 using our prototype, and at large scale over a long period using publicly-available spot price traces and a month-long production job trace from Google [3]. We run all simulation experiments using spot price data from all 14 EC2 AZs in the U.S. Please see the full paper for detailed results [1].

Prototype Results. Our prototype results focus on the application performance and reliability impact of revocations. For these experiments, we use a distributed web server that serves static content as a representative application. Our results show that even in the extreme case of three revocations per minute (which translates to replacing 30% of the server replicas each minute), TR-Kubernetes results in only 0.002% requests failing and its throughput degrades by 17% only when using a single tier interactive service.

Simulation Results. Our simulation experiments analyze the potential cost and availability of an interactive service using TR-Kubernetes over 3 months using spot price data to infer realistic cost characteristics. Experiment results shows that TR-Kubernetes can achieve higher availabilities than using on-demand VMs at a lower cost, ranging from 20% to 80% of the on-demand price depending on the availability requirement.

Acknowledgements. This work is supported by NSF grant #1802523 and Amazon's AWS Cloud Credits for Research program.

REFERENCES

[1] Pradeep Ambati and David Irwin. 2019. Optimizing the Cost of Executing Mixed Interactive and Batch Workloads on Transient VMs. *Proc. ACM Meas. Anal. Comput. Syst.* 3, 2, Article 28 (June 2019). https://doi.org/10.1145/3326143

[2] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes. 2016. Borg, Omega, and Kubernetes. *ACM Queue - Containers* 14, 1 (January-February 2016).

[3] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. 2011. *Google cluster-usage traces: format + schema.* Technical Report. Google Inc.