Unlocking unallocated cloud capacity with SlackSched

Anup Agarwal
anupa@cs.cmu.edu
Carnegie Mellon University

Sudershan Boovaraghavan
sboovara@cs.cmu.edu
Carnegie Mellon University

Hugo Sadok
hfreitas@cs.cmu.edu
Carnegie Mellon University

1 Introduction

Over the past decade, cloud computing platforms have seen remarkable success in providing Infrastructure, Platform, and Software as a Service (IaaS, PaaS, SaaS) for various applications and have become a default preference for enterprises and even scientific research. Cloud providers typically rent out virtual machines (VMs) under an Infrastructure as a Service (IaaS) model. These VMs can be acquired and released on-demand to suit the users' applications. To accommodate fluctuations in user demands and to transparently mask failures for ensuring high availability and reliability of the infrastructure, providers typically provision data centers for peak demands which commonly leads to unallocated resources in data centers [3, 20].

These unallocated resources are often auctioned using low-priority VMs (often called spot VMs) with the proviso that they can be preempted (evicted) at any time. These preemptions occur when there is an increased demand for reliable (high-priority) VMs. Despite such auctioning, a lot of resources still remain unallocated as spot instances are only conservatively allocated to limit the amount of preemptions. Further, many small spot VMs and large number of VM evictions add to VM management, creation and application initialization overheads [3]. To get around these issues, Ambati et al. [3] have proposed a new experimental VM class called Harvest VMs (HVMs).

Harvest VMs, unlike traditional VM offerings, can grow or shrink in terms of their resources (e.g., core count) depending on the amount of unallocated resources on the server they are deployed on. While elastic jobs can readily exploit these growing and shrinking VMs, there are limits to the elasticity of jobs. To ensure forward progress and to avoid trashing, jobs typically need a minimum amount of resources. To accommodate this, HVMs come in different types which provide different guarantees to jobs. Users can choose a minimum core count (e.g., 2, 4, or, 8) and specify whether they want the HVM to be preemptible. An HVM, will never shrink beyond its minimum size. A preemptible HVM is evicted, if the provider needs to reduce the core count below the minimum threshold.

While HVMs can expose more resources for the same cost compared to traditional VMs, the unique characteristics of HVMs compel us to revisit important resource management problems: (1) Scheduling, and (2) Resource Acquisition. (1) Scheduling decides at each time instance, which job/task should run on which VM. Existing job schedulers, oblivious to HVMs, can be overly optimistic leading to too many task preemptions on VM shrinking events, or overly conservative being unable to exploit VM growth events (§2.2). (2) Resource acquisition deals with what HVM mix should one allocate for a given workload. A mix of HVMs containing HVMs of different types or minimum sizes can lead to different performance in terms of job completion times or makespan as different types differ in their ability to harvest and expose unallocated resources (§2.3).

Our work mainly tries to address the question: *How can we adapt scheduling and resource acquisition mechanisms to cater to the unique behavior of harvest VMs?* We partially answer this question by designing and evaluating SlackSched, an HVM aware scheduler. A major challenge we face while doing so, is that it is hard to model and predict the behavior of HVMs. Their resource availability often depends on factors such as VM arrival and departure process of the cloud provider which are invisible to the user of the HVMs. We circumvent this by observing that even though exact knowledge of HVM behaviors is hard to ascertain, but the distribution of their behaviors changes only slowly over time. Using this observation, SlackSched is able to improve mean, median and tail of job completion times by 10-30%.

We have integrated SlackSched within Hadoop, a popular cluster resource management framework. To deliver scheduling improvements, SlackSched assumes knowledge of job resources and lifetimes. Prior works [5, 13, 16, 17] have shown that workload resource and runtime estimation can be done by leveraging historical runs of periodic jobs as well as the structure of jobs. Such workloads can readily use SlackSched without requiring any application changes. Further, providers can use insights from our design to build back-ends for platform as a service, for example, batch processing and serverless computing (e.g., AWS lambda and Azure functions) as these services also often use resource and/or runtime estimates.

On the resource acquisition front, we analyze HVM traces generated from VM arrival data from 20 clusters spread across 15 geographic regions from the deployment of a large cloud provider. Our major finding from this analysis is that, (a) the server on which the HVM is deployed on, and (b) the cluster the server is present in; also matter apart from the minimum size and type of the HVM. This is unlike all traditional VM offerings, wherein, once a user decides the