Re-Take Cloud Computing

Friday July 22, 2022, 10:15 - 13:15

Examiner: dr. K. F. D. Rietveld

- This exam is **closed book**, so it is not allowed to use the textbook, (printed) slides or your own class notes.
- Only calculators that do not have the possibility to store text files may be used. Use of graphing calculators is **not** allowed.
- The questions must be answered in English.

- When handing in your work you will be requested to note your name, student ID and the number of pages ("booklets") that you are handing in on the attendance list.
- *This question sheet must be handed in at the end of the exam.*

- The number of problems is 4 with a total of 19 items. The exam consists of 6 pages (4 pages question sheets, 2 pages article).
- Behind each item the number of points that can be scored is shown between square brackets. The total amount of points that can be scored is 100.
- The obtained grade counts towards the final grade for the course with a weight of 35%.

Problem I: Fundamentals [24 points]

a. [7 points] Briefly describe the three service models of cloud computing (SaaS, PaaS, IaaS) **and** for each of these service models give a concrete example of an **API** that would be provided.

b. [6 points] Clearly explain the difference between *containers* and *virtual machines*. What is actually virtualized in both cases?

c. [6 points] Vendor lock-in continues to be an issue in Cloud Computing. (a) Explain why this is an issue and which service models are affected. (b) How do you reckon this problem should be solved?

d. [5 points] Define Type-1 and Type-2 hypervisors, and explain the differences.

Problem II: Understanding Literature [25 points]

Attached to this exam is the introduction of the following paper that was published in 2018:

Andrew Chung, Jun Woo Park, and Gregory R. Ganger. 2018. Stratus: cost-aware container scheduling in the public cloud. In Proceedings of SoCC '18: ACM Symposium on Cloud Computing, Carlsbad, CA, USA, October 11–13, 2018 (SoCC '18), 14 pages.

a. [4 points] The paper mentions 'Stratus' may use live migration to migrate tasks to other instances. The live migration of a process is similar to the live migration of a virtual machine (in fact, a virtual machine can be considered a process running under a hypervisor). Concisely describe the main steps that need to occur to live migrate a running process to another host.

b. [8 points] The effectiveness of the scheduling approach presented in the paper hinges on a particular property of the workloads to be scheduled. Which property is this? And for which kinds of workloads will this scheduler likely not be able to achieve the improvements that are claimed?

c. [4 points] Traditional cluster schedulers sometimes allow users to assign priorities to their jobs. Higher priority jobs may be scheduled before lower priority jobs. Given the high-level description of 'Stratus', do you reckon that the specification of priorities for jobs is still useful? If yes, indicate how you think Stratus could prioritize jobs. If no, explain why not.

d. [5 points] As described at the end of the introduction, only two workload traces are considered in the experiments. If you were a (picky) reviewer, what further experiments would you like to see?

e. [4 points] In order to pack tasks onto instances, 'Stratus' uses the specification of the required resources that is submitted together with a job request. What is a potential risk of this approach?

Problem III: Design Scenario [25 points]

Let us assume we are working on the backend of a worldwide multiplayer game. Because we target a worldwide audience, different data centers are used in different regions. The game involves a virtual world through which players can navigate. As it is a multiplayer game, multiple players will be present in this world at the same time. So, when playing, a participant in this game will notice other players moving around. To implement this, the position of a player is periodically stored in a database. The game backend will send (push) player locations to other relevant players, such that the player locations are updated in the virtual worlds that are rendered on the computers of the different participants.

a. [4 points] To be able to guarantee a smooth gameplay experience, QoS parameters need to be monitored. Give a number of examples (at least three) of relevant QoS parameters.

When designing a load balancer infrastructure, one can also consider that the application is hosted in data centers located in different regions. This allows us to perform geographical or region-based load balancing.

b. [5 points] When considering region-based load balancing, we can think of two different optimization targets: either to reduce cost or to reduce latency. Describe a load balancer infrastructure for both targets. Based on which properties should the load be balanced? Where to send the incoming traffic to?

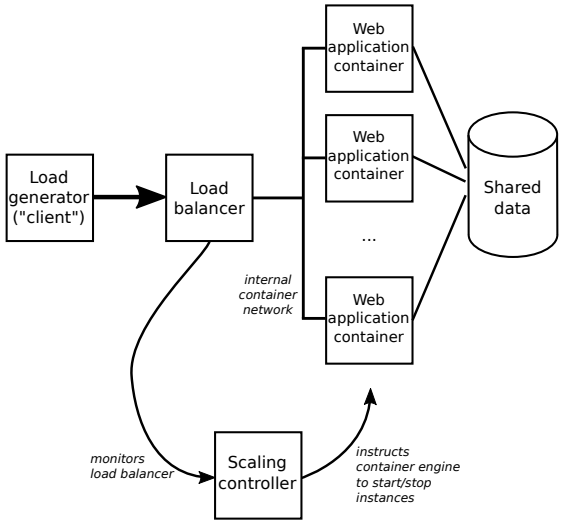
c. [3 points] Given the design scenario, what optimization target would you pick for the load balancer and why?

d. [8 points] Storing all player positions in a central database and streaming the positions from there will very quickly become a significant bottleneck. Solving this problem is key to be able to implement this application successfully. Think about this problem and propose an architecture for this application that mitigates this bottleneck. In other words: how you would store data and where, how is consistency guaranteed, how to efficiently stream positions to other players?

e. [5 points] In order to reduce cost, we are considering to use workload prediction for this application. Do you think workload prediction will be effective for this particular application? Why/why not? How would you use workload prediction / or give a counterexample why you think it will not work in this case?

Problem IV: Lab Assignment [26 points]

The first lab assignment concerned the development of a small autoscaling object store application based on Linux containers provided through podman. The general overview of the developed system is shown in the figure below:



a. [5 points] podman distinguishes between "images" and "containers". Explain this difference and how this enables easy scaling.

We would now like to build an application on top of the object store developed for the lab assignment. The idea is to create an image (photo) upload service, where anonymous users can upload images (photos) and receive a unique URL through which the uploaded image can be accessed. In order to achieve this, an additional front-end web service is to be developed. External users interact with this front-end to upload and retrieve images. The front-end uses the developed object store as a back-end to store images. For now, we assume that the shared storage will be able to keep up with the anticipated load and this will be left unchanged. We would like to be able to run multiple copies of the front-end, in order to distribute the load.

b. [5 points] It is important that every image that is uploaded receives a unique ID. This ID can also be used to store and identify the image in the object store. As multiple front-ends may be uploading images and thus be generating IDs, special measures need to be taken in order to guarantee uniqueness of the IDs. Propose a solution to this problem.

c. [4 points] Do you think it will be possible to make the web front-end stateless? Motivate why / why not.

d. [5 points] What would the final architecture of the application, that combines the front-end and object store, and your answer for **b**, look like? Draw a diagram and briefly explain.

e. [7 points] Naturally, we would like to be able to automatically scale the entire (i.e. all components of the) application. Consider this problem and propose modifications required to the autoscaler. Clearly indicate what to measure, what to scale, how much to scale, when to scale, etc.

Stratus: cost-aware container scheduling in the public cloud

Andrew Chung
Carnegie Mellon University

Jun Woo Park
Carnegie Mellon University

Gregory R. Ganger
Carnegie Mellon University

ABSTRACT

Stratus is a new cluster scheduler specialized for orchestrating batch job execution on *virtual clusters*, dynamically allocated collections of virtual machine instances on public IaaS platforms. Unlike schedulers for conventional clusters, Stratus focuses primarily on dollar cost considerations, since public clouds provide effectively unlimited, highly heterogeneous resources allocated on demand. But, since resources are charged-for while allocated, Stratus aggressively packs tasks onto machines, guided by job runtime estimates, trying to make allocated resources be either mostly full (highly utilized) or empty (so they can be released to save money). Simulation experiments based on cluster workload traces from Google and TwoSigma show that Stratus reduces cost by 17–44% compared to state-of-the-art approaches to virtual cluster scheduling.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Software and its engineering** → *Scheduling*;

KEYWORDS

cloud computing, cluster scheduling, transient server

ACM Reference Format:
Andrew Chung, Jun Woo Park, and Gregory R. Ganger. 2018. Stratus: cost-aware container scheduling in the public cloud. In *Proceedings of SoCC '18: ACM Symposium on Cloud Computing , Carlsbad, CA, USA, October 11–13, 2018 (SoCC '18),* 14 pages.
https://doi.org/10.1145/3267809.3267819

1 INTRODUCTION

Public cloud computing has matured to the point that many organizations rely on it to offload workload bursts from traditional on-premise clusters (so-called "cloud bursting") or even to replace on-premise clusters entirely. Although traditional cluster schedulers could be used to manage a mostly static allocation of public cloud virtual machine (VM) instances,[1] such an arrangement would fail to exploit the public cloud's elastic on-demand properties and thus be unnecessarily expensive.

A common approach [15, 36, 38, 54] is to allocate an instance for each submitted task and then release that instance when the task completes. Although straightforward, this new-instance-per-task approach misses significant opportunities to reduce cost by packing tasks onto fewer and perhaps larger instances. Doing so can increase utilization of rented resources and enable exploitation of varying price differences among instance types.

What is needed is a *virtual cluster (VC) scheduler* that packs work onto instances, as is done by traditional schedulers, without assuming that a fixed pool of resources is being managed. The concerns for such a scheduler are different than for traditional clusters, with resource rental costs being added and queueing delay being removed by the ability to acquire additional resources on demand rather than forcing some jobs to wait for others to finish. Minimizing cost requires good decisions regarding which tasks to pack together on instances as well as when to add more instances, which instance types to add, and when to release previously allocated instances.

Stratus is a scheduler specialized for virtual clusters on public IaaS platforms. Stratus adaptively grows and shrinks its allocated set of instances, carefully selected to minimize cost and accommodate high-utilization packing of tasks. To minimize cost over time, Stratus endeavors to get as close as possible to the ideal of having every instance be either 100% utilized by submitted work or 0% utilized so it can be immediately released (to discontinue paying for it). Via aggressive use of a new method we call *runtime binning*, Stratus groups and packs tasks based on when they are predicted to complete. Done well, such-packed tasks will fully utilize an instance, complete around the same time, and allow release of the then-idle instance with minimal under-utilization. To avoid extended retention of low-utilization instances due to mispredicted runtimes, Stratus migrates still-running tasks to clear out such instances.

Stratus's scale-out decisions are also designed to exploit both instance type diversity and instance pricing variation (static and dynamic). When additional instances are needed in the virtual cluster in order to immediately run submitted tasks, Stratus requests instance types that cost-effectively fit sets of predicted-completion-time-similar tasks. We have found that achieving good cost savings requires considering packings of pending tasks in tandem with the cost-per-resource-used of instances on which the tasks could fit; considering either alone before the other leads to many fewer <packing, instance-type> combinations considered and thereby higher costs. Stratus co-determines how many tasks to pack onto instances and which instance types to use.

Simulation experiments of virtual clusters in AWS spot markets, driven by cluster workload traces from Google and TwoSigma, confirm Stratus's efficacy. Stratus reduces total cost by 25% (Google) and 31% (TwoSigma) compared to an aggressive state-of-the-art non-packing task-per-VM approach [47]. Compared to two state-of-the-art VC schedulers that combine dynamic virtual cluster scaling with job packing, Stratus reduces cost by 17–44%. Even with static instance pricing, such as is used for AWS's on-demand instances as

[1] We use "instance" as a generic term to refer to a virtual machine resource rented in a public IaaS cloud.

SoCC '18, October 11–13, 2018, Carlsbad, CA, USA

ACM ISBN 978-1-4503-6011-1/18/10…$15.00
https://doi.org/10.1145/3267809.3267819

well as Google Compute Engine and Microsoft Azure, Stratus reduces cost by 10–29%. Attribution of Stratus's benefits indicates that significant value comes from each of its primary elements—runtime-conscious packing, instance diversity-awareness, and under-utilization-driven migration. Further, we find that the combination is more than the sum of the parts and that failure to co-decide packing and instance type selection significantly reduces cost savings.

This paper makes four primary contributions. **(1)** It identifies the unique mix of characteristics that indicate a role for a new job scheduler specialized for virtual clusters (VCs). **(2)** It describes how runtime-conscious packing can be used to minimize under-utilization of rented instances and techniques for making it work well in practice, including with imperfect runtime predictions. **(3)** It exposes the inter-dependence of packing decisions and instance type selection, showing the dollar cost benefits of co-determining them. **(4)** It describes a batch-job scheduler (Stratus) using novel packing and instance acquisition policies, and demonstrates the effectiveness of its policies with trace-driven simulations of two large-scale, real-world cluster workloads.