Stochastic Resource Allocation

Liran Funaro  Orna Agmon Ben-Yehuda  Assaf Schuster

Department of Computer Science

International Conference on Virtual Execution Environments (VEE)
April 14, 2019, Providence, RI, USA
The Problem: Fixed Resource Bundles

- Resources in the cloud are underutilized
- The main cause of resource underutilization is fixed performance bundles

- Clients rent the resources to sustain their highest workload
  - But they do not use the resources all the time
- The provider guarantees with good probability that the clients will be able to use their rented resources at any given time
- It must reserve these resources
  - It cannot resell them or use them to other purposes

- Incentivizing clients to reduce their fixed reserved resource requirements might solve the problem by allowing more clients per physical machine
The existing solution is not perfect

- A client gains credits periodically, at an even rate
- The client either
  - Consumes credits by using the resource
  - Hoards the credits and "bursts" later
A client gains credits periodically, at an even rate

The client either
- Consumes credits by using the resource
- Hoards the credits and "bursts" later

<table>
<thead>
<tr>
<th>Cores</th>
<th>Accumulated Credit</th>
<th>CPU Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The credit mechanism limits the client to a certain average resource consumption.
Burstable Performance

The existing solution is not perfect

- A client gains credits periodically, at an even rate
- The client either
  - Consumes credits by using the resource
  - Hoards the credits and "bursts" later

![Diagram showing credit accumulation and CPU usage over time](image-url)

L. Funaro, O. Agmon Ben-Yehuda, A. Schuster (Technion)

Stochastic Resource Allocation
Burstable Performance

The existing solution is not perfect

- A client gains credits periodically, at an even rate
- The client either
  - Consumes credits by using the resource
  - Hoards the credits and "bursts" later

![Graph showing accumulated credit and CPU usage over time](image-url)
Burstable Performance

The existing solution is not perfect

▸ A client gains credits periodically, at an even rate
▸ The client either
  ▸ Consumes credits by using the resource
  ▸ Hoards the credits and "bursts" later
Burstable Performance

The existing solution is not perfect

- A client gains credits periodically, at an even rate
- The client either
  - Consumes credits by using the resource
  - Hoards the credits and "bursts" later

- The credit mechanism limits the client to a certain average resource consumption

![Chart 1: Accumulated Credit and CPU Usage](chart1)

![Chart 2: Accumulated Credit and CPU Usage](chart2)
The existing solution is not perfect.

- A client gains credits periodically, at an even rate.
- The client either:
  - Consumes credits by using the resource.
  - Hoards the credits and "bursts" later.
- The credit mechanism limits the client to a certain average resource consumption.

Disadvantages:
- **Hidden information** regarding resource availability.
- **Coupling** of reserved resources and average usage.
Under the SA mechanism, the provider offers clients a combination:

- an amount of reserved resources
- with a choice of a stochastic allocation class

The provider posts fixed unit-prices for both goods

And periodically publishes statistics on resource availability for each SA class

Each client may choose to rent reserved and/or stochastic resources
Implementing Stochastic Allocation via Shares

- Linux’s completely fair scheduler (CFS) combines a share-based resource allocation system with a hard rate limit.
- Each task is assigned a number of shares, which entitle it to a portion of the resources proportional to the number of allocated shares.
Linux’s completely fair scheduler (CFS) combines a share-based resource allocation system with a hard rate limit.

Each task is assigned a number of shares, which entitle it to a portion of the resources proportional to the number of allocated shares.
Implementing Stochastic Allocation via Shares

- Linux’s completely fair scheduler (CFS) combines a share-based resource allocation system with a hard rate limit.
- Each task is assigned a number of shares, which entitle it to a portion of the resources proportional to the number of allocated shares.
Linux’s completely fair scheduler (CFS) combines a share-based resource allocation system with a hard rate limit. Each task is assigned a number of shares, which entitle it to a portion of the resources proportional to the number of allocated shares. Having a portion of the shares is effectively the same as reserving the same portion of the resources.
Linux’s completely fair scheduler (CFS) combines a share-based resource allocation system with a hard rate limit. Each task is assigned a number of shares, which entitle it to a portion of the resources proportional to the number of allocated shares. Having a portion of the shares is effectively the same as reserving the same portion of the resources.
Implementing Stochastic Allocation via Shares

- Linux’s completely fair scheduler (CFS) combines a share-based resource allocation system with a hard rate limit.
- Each task is assigned a number of shares, which entitle it to a portion of the resources proportional to the number of allocated shares.
- Having a portion of the shares is effectively the same as reserving the same portion of the resources.
Implementing Stochastic Allocation via Shares

- Linux’s completely fair scheduler (CFS) combines a share-based resource allocation system with a hard rate limit.
- Each task is assigned a number of shares, which entitle it to a portion of the resources proportional to the number of allocated shares.
- Having a portion of the shares is effectively the same as reserving the same portion of the resources.
- CFS does not support a key feature of SA: defining a different consumption share for the leftover CPUs.
Implementing Stochastic Allocation via Shares

- Linux’s completely fair scheduler (CFS) combines a share-based resource allocation system with a hard rate limit.
- Each task is assigned a number of shares, which entitle it to a portion of the resources proportional to the number of allocated shares.
- Having a portion of the shares is effectively the same as reserving the same portion of the resources.
- CFS does not support a key feature of SA: defining a different consumption share for the leftover CPUs.

We adapted CFS to support asymmetric reserved resources and share allocations.

We duplicated the CFS logic, to have a second, alternative, CFS.
Evaluating Our Solution

How can we compare our solution to **burstable performance** and **fixed performance**?

- Will it improve the **utilization**?
- Will it be more **popular among clients**?
- Will it be more **profitable to the providers**?
Evaluating Our Solution

How can we compare our solution to **burstable performance** and **fixed performance**?

- Will it improve the **utilization**?
- Will it be more **popular among clients**?
- Will it be more **profitable to the providers**?

- We developed a framework to evaluate new resource allocation schemes
- It simulates a **realistic data center** with **realistic servers** and **clients**
We used data from the **Azure public dataset**

- Includes data for over 2 million clients
  - Purchased **bundle** (fixed-performance)
  - **CPU usage** every 5 minutes (min, max and average)
  - and more...

Available from: [https://github.com/Azure/AzurePublicDataset](https://github.com/Azure/AzurePublicDataset)
Clients’ Load

- We simulated scenarios where clients share CPU at fine granularity
We simulated scenarios where clients share CPU at fine granularity.
We simulated scenarios where clients share CPU at fine granularity.
We simulated scenarios where clients share CPU at fine granularity

- We generated 25 samples from each 5 minute sample such that their minimum, maximum and average match the sample
- We used beta distribution, which can be defined by its average and bounds.
What is the performance the clients gain from the CPU?

To allow the clients to make an informed decision when selecting a bundle, we generated a *required performance distribution function*

- Cumulative distribution function
- It is inspired by performance functions for real applications
Evaluation Methodology

Each simulated client selected the most profitable bundle for its load and resource requirements.

It used its own load statistics to make a decision.
To allocate clients to 64-core servers, we randomly shuffled them.

Then, one at a time, each client was assigned to the first server that could accommodate its bundle.
Each client’s load for the current day (iteration) was selected cyclically from its data over multiple days.

The provider collected statistics on the resource utilization in each server.
The cloud provider supplies **statistical information** regarding the maximal resource amount that a client might obtain over a short period with the commensurate number of shares.
A number of clients were allowed to switch their bundle in each iteration.

They used their own **load statistics** and the provider’s statistical description of the resources that every bundle yields.
Evaluation Methodology

- Bundle Selection
- Client Allocation
- Cloud Simulation
- Share Distribution Calculation
- Statistics Collection
- Distribution Publishing

Graph showing clients per server, shares allocated, and mean utilization over iterations.
Evaluating Our Framework

- We simulated a fixed-performance allocation scheme
- Our results were similar to known cloud data before burstable performance was introduced

- 15%-20% CPU utilization
- Bundle distribution (right)
  - The selected number of virtual cores in our simulation and in the Azure dataset
Evaluating Our Framework

- We simulated a fixed-performance allocation scheme
- Our results were similar to known cloud data before burstable performance was introduced
- 15%-20% CPU utilization
- Bundle distribution (right)
  - The selected number of virtual cores in our simulation and in the Azure dataset

Our framework is validated and consistent with real data
Evaluating our Solution

- **Fixed Performance (FP)**
  - FP always offered to the clients as an alternative
  - A CPU unit costs $1

- **Burstable Performance (BP) — share costs $2 to $4**
  - The client can rent bundles in which the number of reserved resources equals the number of shares
  - The shares can be utilized without limitations
  - The client’s average consumption is limited

- **Stochastic Allocation (SA) — share costs $0.15 to $0.9**
  - The client can rent shares alongside reserved resources
  - A share can be utilized only up to its absolute value
Clients per Server (CPS)

The graph shows the Clients/Server and the Maximal CPS for different scenarios:
- 40 clients per server
- 158 clients per server
- 133 clients per server
- 107 clients per server
- 233 clients per server
- 227 clients per server
- 224 clients per server
- 220 clients per server
- 206 clients per server
- 223 clients per server

Normalized Revenues:
- 0
- 0.25
- 0.50
- 0.75
- 1.00

FP
BP 2
BP 3
BP 4
SA .15
SA .5
SA .6
SA .7
SA .9
SA/BP

More clients per server compared to BP of the clients preferred SA over FP.

L. Funaro, O. Agmon Ben-Yehuda, A. Schuster (Technion)

Stochastic Resource Allocation
Clients per Server (CPS)

- ▶ 70% more clients per server compared to BP
70% more clients per server compared to BP
92%–99% of the clients preferred SA over FP
56% of the clients preferred SA over BP and FP
The mean total utilization (\%seven.pnum.oldstyle/three.pnum.oldstyle) is higher than for \%four.pnum.oldstyle/\%four.pnum.oldstyle. Reserved utilization is similar to its total utilization.
SA mean total utilization (73%) is higher than for BP (44%)
SA mean total utilization (73%) is higher than for BP (44%).

BP reserved utilization is similar to its total utilization.
Provider Goals

Public cloud providers:

- Maximize their profit from renting their machines
- Take servers' operational costs into account

Is it possible that Amazon has lost money by introducing burstable performance?

Introducing Amazon EC2 T3 Instances

Posted On: Aug 21, 2018

Amazon Web Services (AWS) is introducing the next generation Amazon Elastic Compute Cloud (EC2) burstable general-purpose instances, T3. T3 instances offer a balance of compute, memory, and network resources and are designed to provide a baseline level of CPU performance with the ability to burst above the baseline when needed. T3 instances are powered by the AWS Nitro System which includes a lightweight...
Provider Goals

Public cloud providers:

- Maximize their profit from renting their machines
- Take servers’ operational costs into account

Is it possible that Amazon has lost money by introducing burstable performance?

Probably not...

Introducing Amazon EC2 T3 Instances

Posted On: Aug 21, 2018

Amazon Web Services (AWS) is introducing the next generation Amazon Elastic Compute Cloud (EC2) burstable general-purpose instances, T3. T3 instances offer a balance of compute, memory, and network resources and are designed to provide a baseline level of CPU performance with the ability to burst above the baseline when needed. T3 instances are powered by the AWS Nitro System which includes a lightweight
SA can increase the provider’s profits of the public cloud provider by over \( \frac{2}{8} \) compared to BP.
SA can increase the profits of the public cloud provider by over 28% compared to BP.
Provider Goals

Public cloud providers:
- Maximize their profit from renting their machines
- Take servers’ operational costs into account

Private cloud providers:
- Maximize the aggregated benefit all their clients draw from a single server
Provider’s Aggregated Benefit per Server

- **SA** increases the value each server generates for the corporation by over **55%** compared to **BP**
- **SA** achieved over **98%** of the optimal social welfare
Conclusions

▶ Stochastic CPU allocation via shares allows clients to reduce their reserved resource requirements

▶ SA increases the number of clients per server by more than 70% compared to BP

▶ SA increases the profits of the public cloud provider by over 28% compared to BP

▶ SA increases the value each server generates for the corporation by over 55%

▶ Our evaluation framework is validated against real cloud data

▶ It is available as an open source: https://bitbucket.org/funaro/stochastic-allocation

Questions?

Liran Funaro: funaro@cs.technion.ac.il