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
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

<table>
<thead>
<tr>
<th>Cycles</th>
<th>Accumulated Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st sec</td>
<td>1</td>
</tr>
<tr>
<td>2nd sec</td>
<td>2</td>
</tr>
<tr>
<td>3rd sec</td>
<td>3</td>
</tr>
</tbody>
</table>

- 100% CPU Usage
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

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

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

Stochastic Resource Allocation

VEE '19
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 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
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

Disadvantages:
- **Hidden information** regarding resource availability
- **Coupling** of reserved resources and average usage
Stochastic Allocation (SA)

- 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.
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.

![Diagram showing tree with nodes and shares:](image)

6
5
4
3
2
1
0

A (1 share) B (2 shares) C (3 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.

<table>
<thead>
<tr>
<th>Cores</th>
<th>Reserved Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>(1 share)</td>
</tr>
<tr>
<td>B</td>
<td>(2 shares)</td>
</tr>
<tr>
<td>C</td>
<td>(3 shares)</td>
</tr>
</tbody>
</table>

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

![Reserved Usage Diagram]

A (1 share)  B (2 shares)  C (3 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.
- Having a portion of the shares is effectively the same as reserving the same portion of the resources.

![Diagram showing extra and reserved usage for tasks A, B, and C with 1, 2, and 3 shares respectively.](image-url)
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

![Diagram showing Extra Usage and Reserved Usage for A (1 share), B (2 shares), and C (3 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.
- 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 adopted 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)
We simulated scenarios where clients share CPU at fine granularity.
Clients’ Load

We simulated scenarios where clients share CPU at fine granularity.

![CPU Usage Chart]

5 Minute Interval

Client A

Client B
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

- Bundle Selection
- Client Allocation
- Cloud Simulation

Distribution Publishing

Share Distribution Calculation

Statistics Collection
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.

Distribution Publishing

Share Distribution Calculation

Statistics Collection

Cloud Simulation

Client Allocation

Bundle Selection

[Diagram of workflow]

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

Stochastic Resource Allocation
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.
Evaluation Methodology

- 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
- Distribution Publishing
- Statistics Collection

The graph shows the relationship between Clients per Server, Shares Allocated, and Mean Utilization over iterations. The x-axis represents iterations, while the y-axis shows the number of Clients per Server and Shares Allocated. The graph also includes a y-axis for Utilization, with percentages ranging from 0% to 100%.
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)

Maximal CPS: 40, 158, 133, 107, 233, 227, 224, 220, 206, 223

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

Normalized Revenues: 0, 0.00, 0.25, 0.50, 0.75, 1.00

More clients per server compared to BP—nine percent of the clients preferred SA over FP—five percent of the clients preferred SA over BP and FP.
Clients per Server (CPS)

- 70% more clients per server compared to BP
Clients per Server (CPS)

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

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
Provider’s Profit

- SA can increase the profit of the public cloud provider by over 80% compared to BP

- Normalized Profit

- $47
- $39
- $31
- Break Even

<table>
<thead>
<tr>
<th>Normalized Profit</th>
<th>$47</th>
<th>$39</th>
<th>$31</th>
<th>Break Even</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BP 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BP 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BP 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA .15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA .5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA .6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA .7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA .9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA/BP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

L. Funaro, O. Agmon Ben-Yehuda, A. Schuster (Technion)
Stochastic Resource Allocation
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
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?