Stochastic Resource Allocation

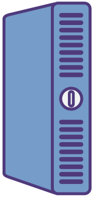
Liran Funaro Orna Agmon Ben-Yehuda Assaf Schuster

Department of Computer Science
TECHNION
Israel Institute
of Technology

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

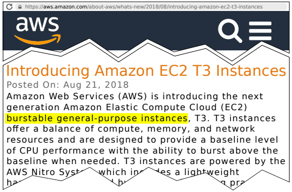


- Clients rent the resources to sustain their highest workload
  - But they do not use the resources all the time
- The provider guarantees with good probability that the clients will be able to use their rented resources at any given time
- It must reserve these resources
  - It cannot resell them or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

Burstable Performance

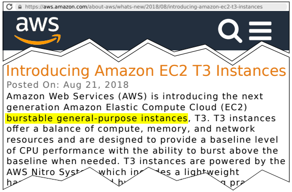
The existing solution is not perfect



- Still reserve the resource
- Instead of an upper limit…
- Limits the client to a certain average resource consumption
- Adopted by many major cloud providers

Burstable Performance

The existing solution is not perfect



- Still reserve the resource
- Instead of an upper limit...
- Limits the client to a certain average resource consumption
- Adopted by many major cloud providers

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

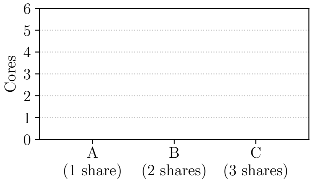
- The provider posts fixed unit-prices for both goods
- And periodically publishes statistics on resource availability for each SA class
- Each client may choose to rent reserved and/or stochastic resources

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

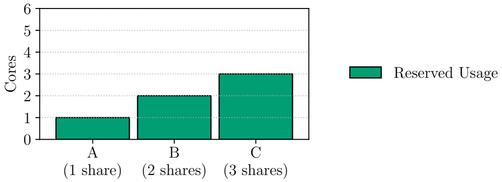
Implementing Stochastic Allocation via Shares

- 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



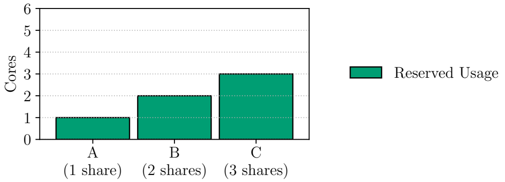
Implementing Stochastic Allocation via Shares

- 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



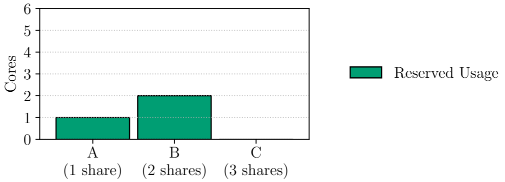
Implementing Stochastic Allocation via Shares

- 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
- Having a portion of the shares is effectively the same as reserving the same portion of the resources



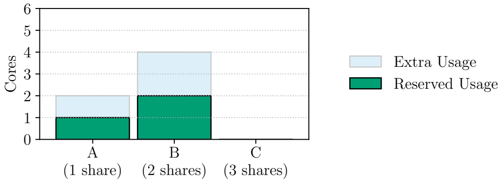
Implementing Stochastic Allocation via Shares

- 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



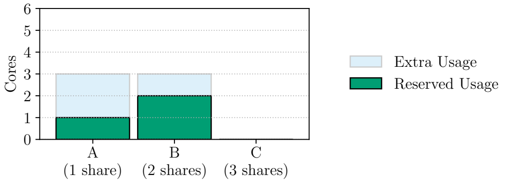
Implementing Stochastic Allocation via Shares

- 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



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**

Azure Public Dataset

- 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...

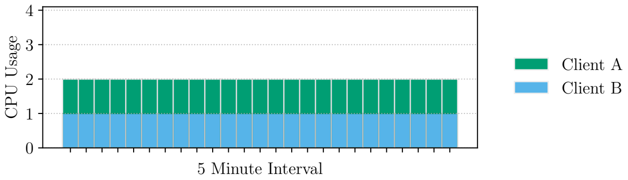
Available from: `https://github.com/Azure/AzurePublicDataset`

Clients' Load

- We simulated scenarios where clients share CPU at fine granularity

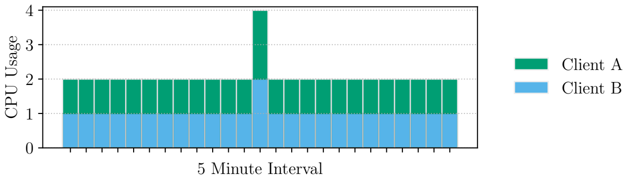
Clients' Load

- We simulated scenarios where clients share CPU at fine granularity



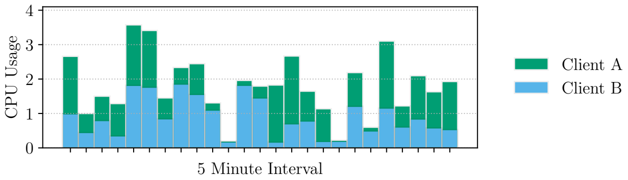
Clients' Load

- We simulated scenarios where clients share CPU at fine granularity



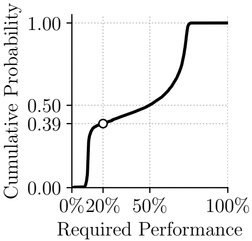
Clients' Load

- 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

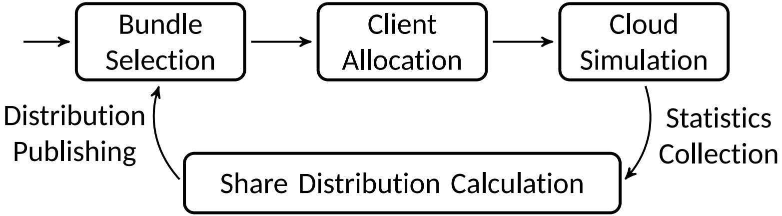


Clients' Performance

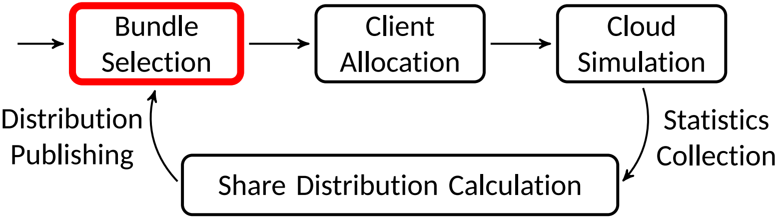
- What is the performance the clients gain from the CPU?
  - To allow the clients to make an informed decision when selecting a bundle, we generated a **required performance distribution function**
    - Cumulative distribution function
    - It is inspired by performance functions for real applications

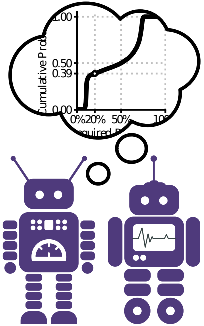


Evaluation Methodology



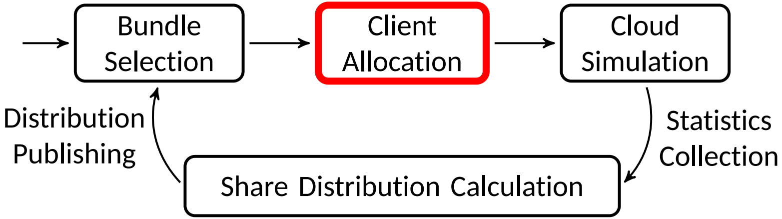
Evaluation Methodology





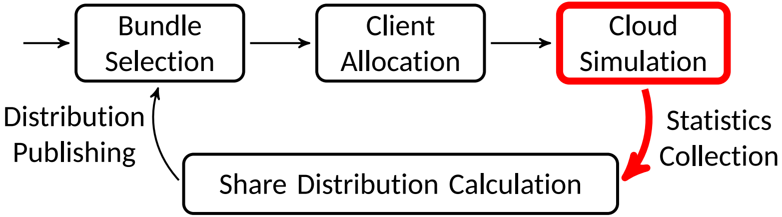
- Each simulated client selected the most **profitable** bundle for its load and resource requirements
- It used its own **load statistics** to make a decision

Evaluation Methodology



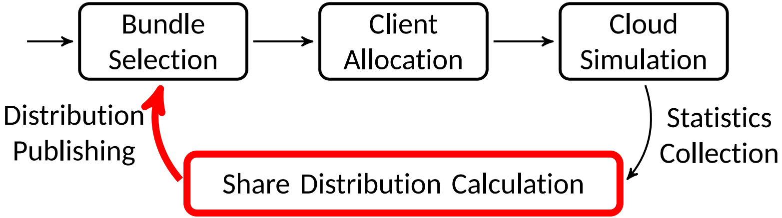
- To allocate clients to 64-core servers, we randomly shuffled them
- Then, one at a time, each client was assigned to the first server that could accommodate its bundle

Evaluation Methodology

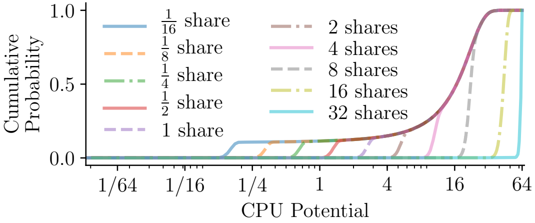


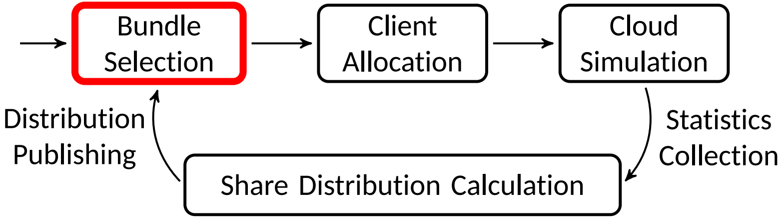
- Each client's load for the current day (iteration) was selected cyclically from its data over multiple days
- The provider collected statistics on the resource utilization in each server

Evaluation Methodology

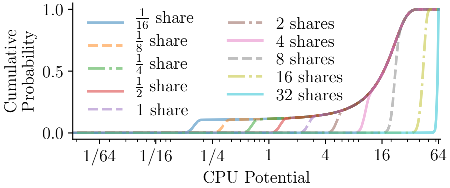


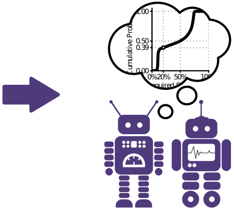
- The cloud provider supplies **statistical information** regarding the maximal resource amount that a client might obtain over a short period with the commensurate number of shares



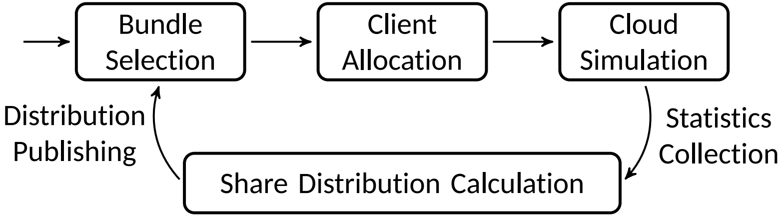


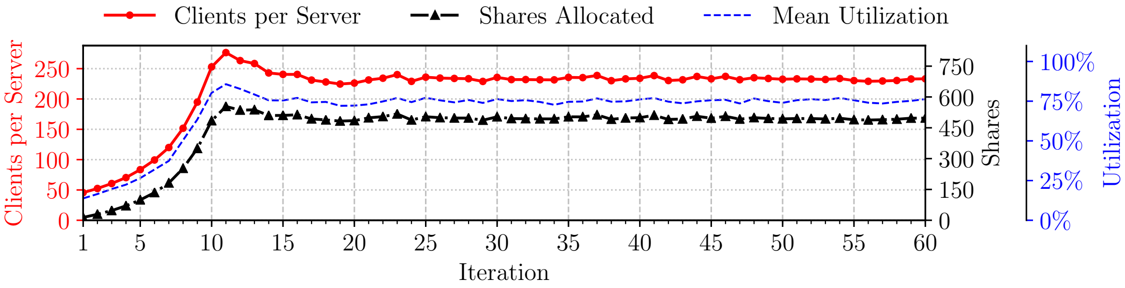
- A number of clients were allowed to switch their bundle in each iteration
- They used their own **load statistics** and the provider's statistical description of the resources that every bundle yields





Evaluation Methodology

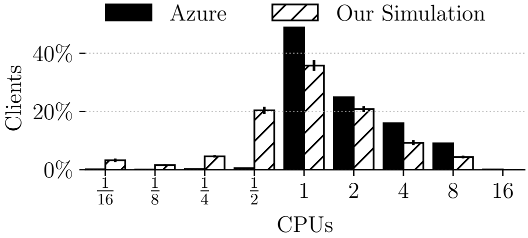




Evaluating Our Framework

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

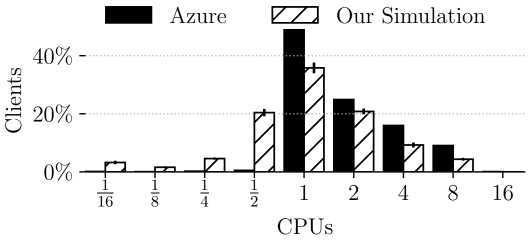
- 15%-20% CPU utilization
- Bundle distribution (right)
  - The selected number of virtual cores in our simulation and in the Azure dataset



Evaluating Our Framework

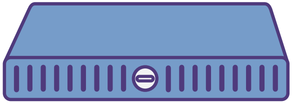
- We simulated a fixed-performance allocation scheme
- Our results were similar to known cloud data before burstable performance was introduced

- 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

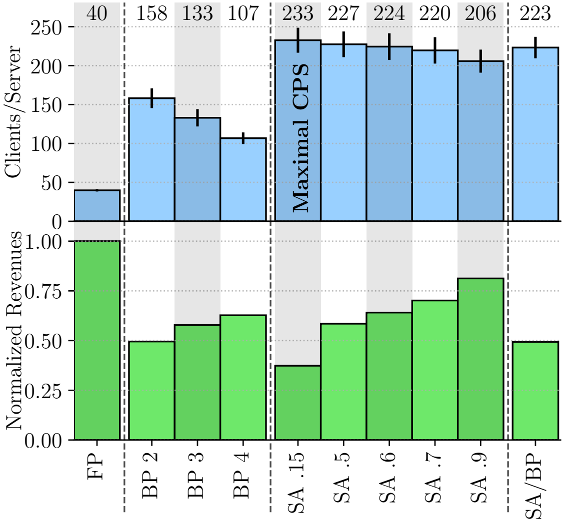


- Fixed Performance (FP)
  - FP always offered to the clients as an alternative
  - A CPU unit costs $1

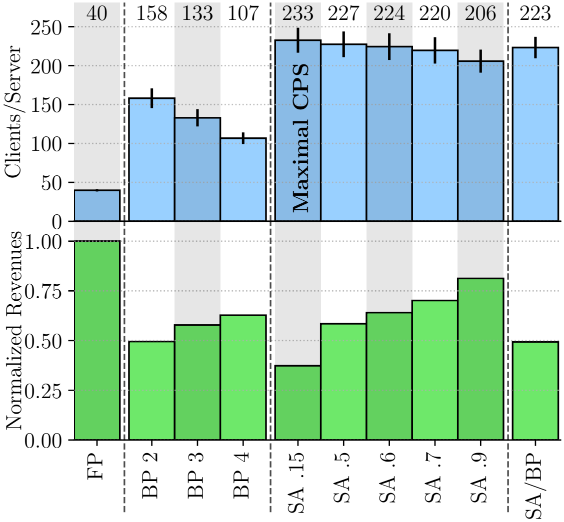
- Burstable Performance (BP) — share costs $2 to $4
  - The client can rent bundles in which the number of reserved resources equals the number of shares
  - The shares can be utilized without limitations
  - The client's average consumption is limited

- 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)

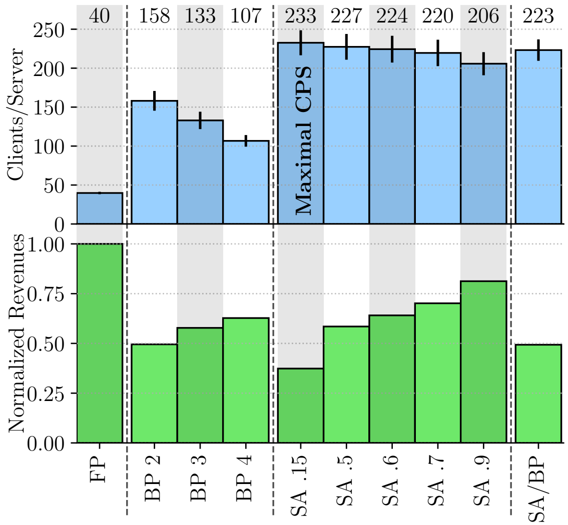


Clients per Server (CPS)



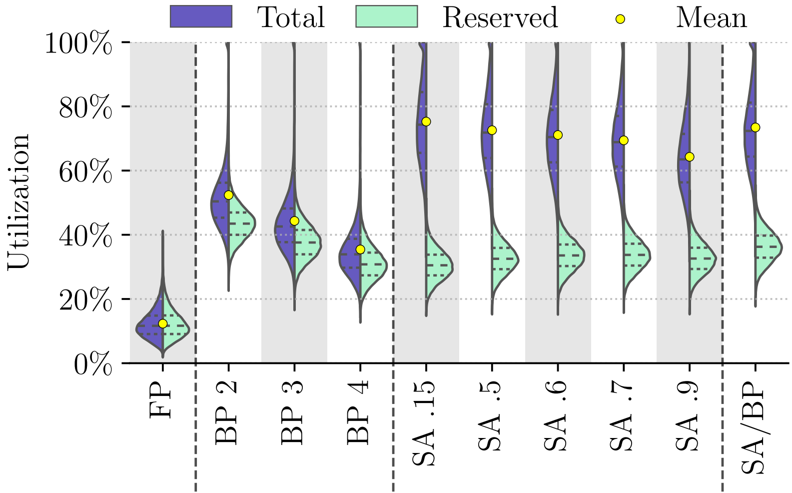
- **70%** more clients per server compared to **BP**

Clients per Server (CPS)

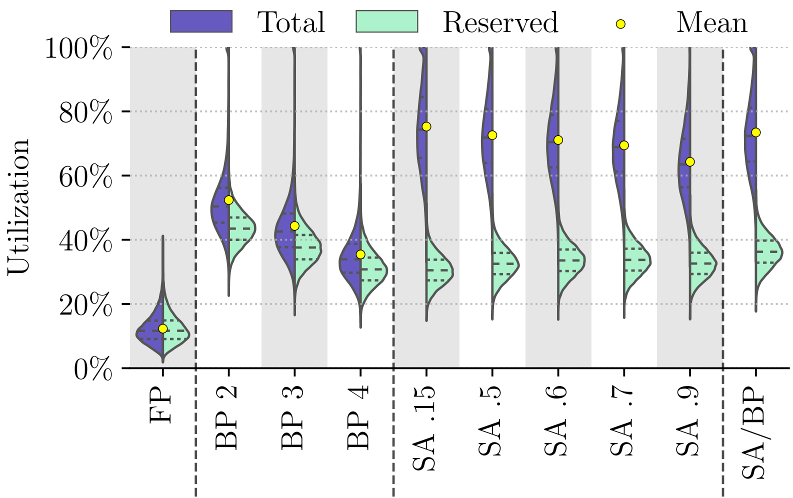


- **70%** more clients per server compared to **BP**
- **92%–99%** of the clients preferred **SA** over **FP**
- **56%** of the clients preferred **SA** over **BP** and **FP**

Utilization

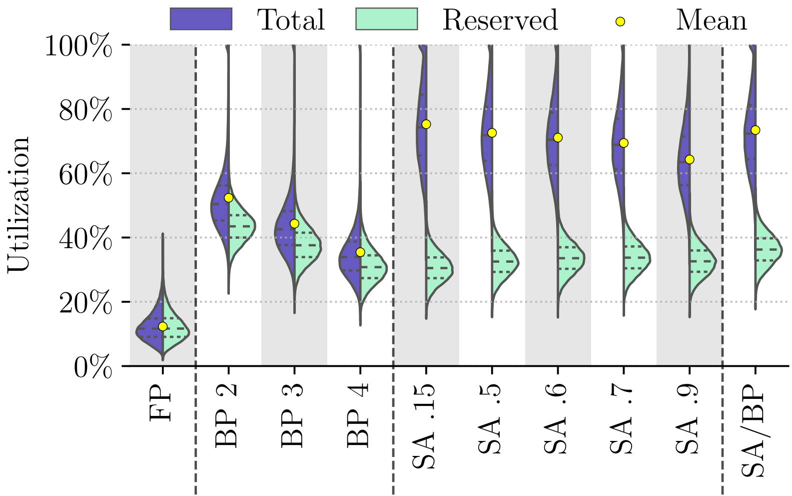


Utilization



- **SA** mean total utilization (73%) is higher than for **BP** (44%)

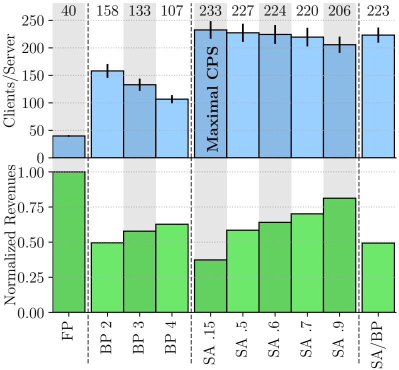
Utilization



- **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?

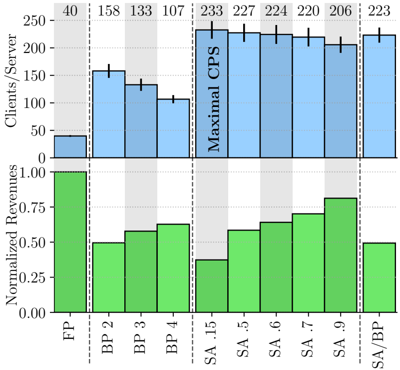
Introducing Amazon EC2 T3 Instances

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

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?

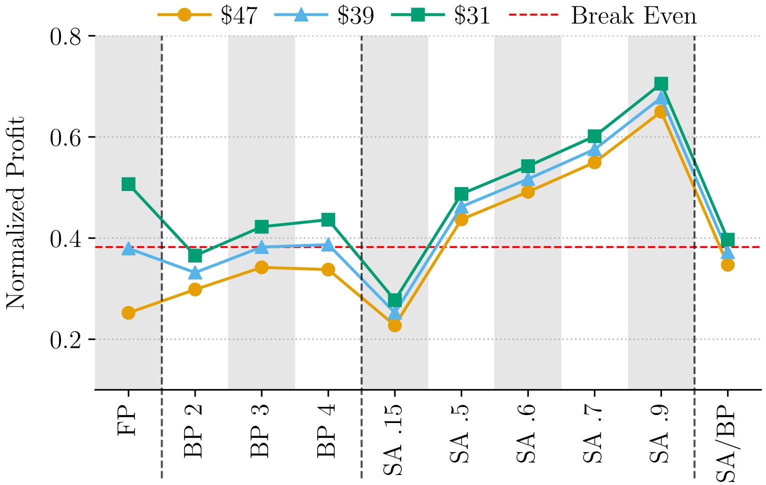
Probably not...

Introducing Amazon EC2 T3 Instances

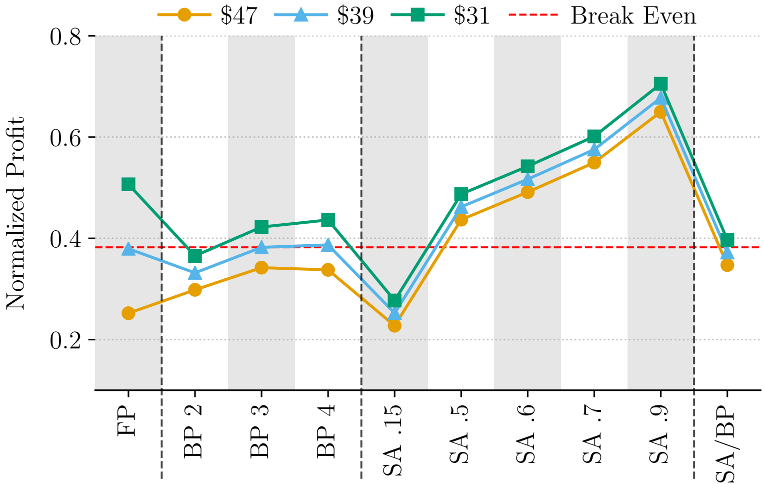
Posted On: Aug 21, 2018

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



Provider's Profit



- **SA** can increase the profits of the public cloud provider by over **28%** compared to **BP**

Provider Goals

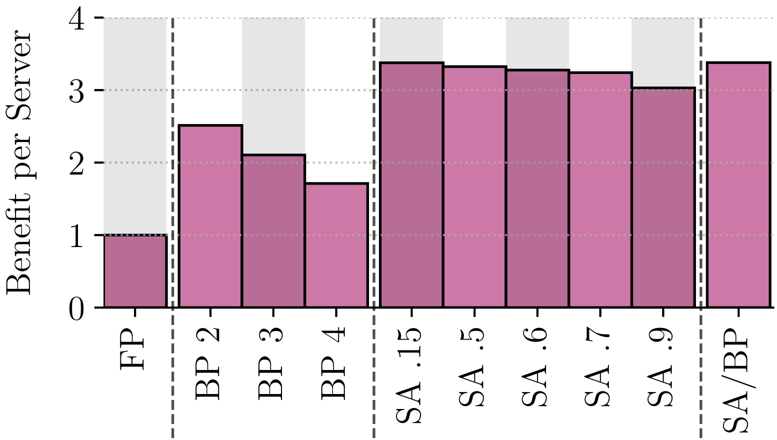
Public cloud providers:

- 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

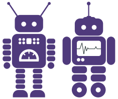
Provider's Aggregated Benefit per Server



- **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%**



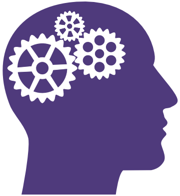
- Our evaluation framework is validated against real cloud data
- It is available as an open source:
  `https://bitbucket.org/funaro/stochastic-allocation`

Questions?

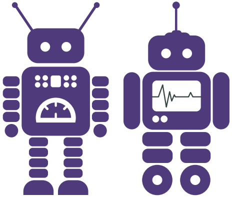
Liran Funaro: `funaro@cs.technion.ac.il`

Backup

Simulating Clients



- Real (human) clients may choose an offering in any way they like
- They may choose randomly, take some time to make a decision, or go through a long iterative process of selection and improvement



- In the simulation we needed to create realistic **artificial intelligence agents** which mimic the behavior of real clients

Clients' Requirements

Long-term Requirements

- Have **non-interactive** workloads
- Value finishing the workload by or before a deadline
- Not value getting partial results ahead of time

Clients' Requirements

Long-term Requirements

- Have **non-interactive** workloads
- Value finishing the workload by or before a deadline
- Not value getting partial results ahead of time

Immediate Requirements

- Runs **brief** independent workloads or an **interactive** workload, and sleeps the rest of the time
- The failure or fulfillment of one workload does not affect the client's future requirements

Clients' Requirements

Long-term Requirements

- Have **non-interactive** workloads
- Value finishing the workload by or before a deadline
- Not value getting partial results ahead of time

Immediate Requirements

- Runs **brief** independent workloads or an **interactive** workload, and sleeps the rest of the time
- The failure or fulfillment of one workload does not affect the client's future requirements

Example: Web Application

- Partition their budget proportionately to the gain from satisfying these dual requirements
- Would not like to miss an opportunity to show an advertisement to their visitors
- Preserve their customers' visit rate

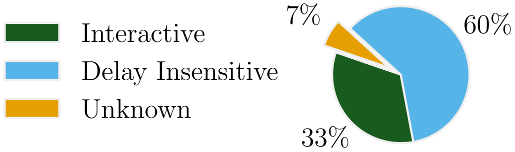
Budget

- We used the client's **purchased fixed-performance** bundle as an lower bound on their budget and draw a budget out of a **Pareto** distribution that is higher than this bound

Budget

- We used the client's **purchased fixed-performance** bundle as an lower bound on their budget and draw a budget out of a **Pareto** distribution that is higher than this bound

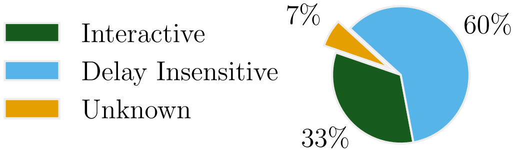
- To determine how to **split the budget** between the two requirements we used Azure **classification**



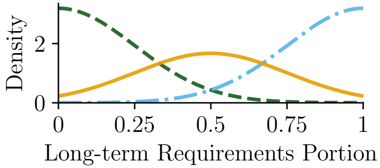
Budget

- We used the client's **purchased fixed-performance** bundle as an lower bound on their budget and draw a budget out of a **Pareto** distribution that is higher than this bound

- To determine how to **split the budget** between the two requirements we used Azure **classification**



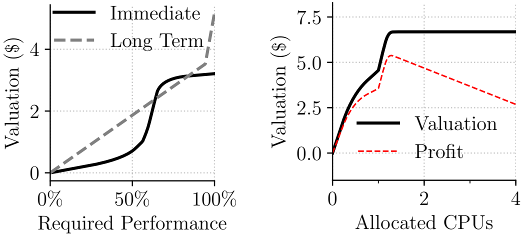
- A **probability density function** (PDF) of the portion between the two valuation types
- The immediate requirements portion completes it to 1



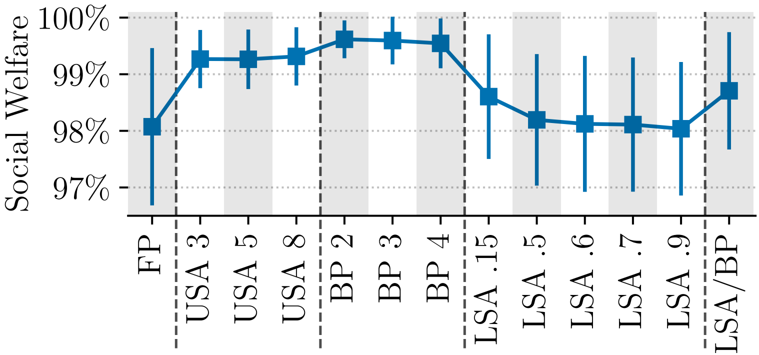
Valuation Functions

- For each type of requirement, we created a valuation function
- V_{imm} and V_{lt} yields a monetary value for an **immediate** and **long-term** performance
- The client's profit: $E\left(V_{imm}(P_{r,s})\right) + V_{lt}\left(E(P_{r,s})\right) - Cost_{r,s}$
  - s and r denotes the number of shares and reserved resources
  - $P_{r,s}$ denotes a random variable which describes the clients' performance with r and s

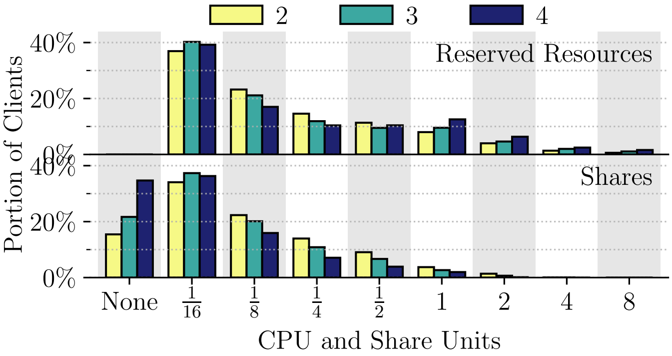
- We generated for each client V_{imm} and V_{lt} such that its profit will be higher for their bundle of reserved resources than any other bundle of reserved resources ($s \equiv 0$)
- Solved implicitly



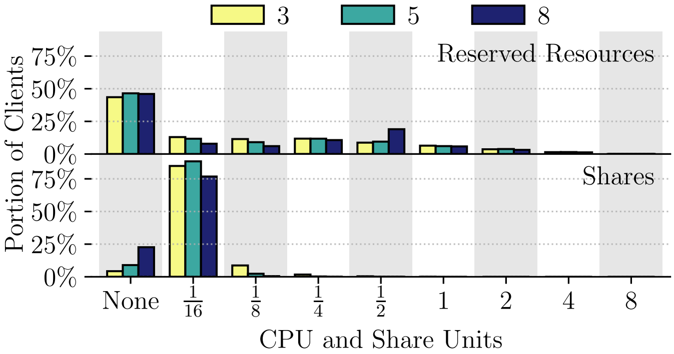
Comparison of the Social Welfare



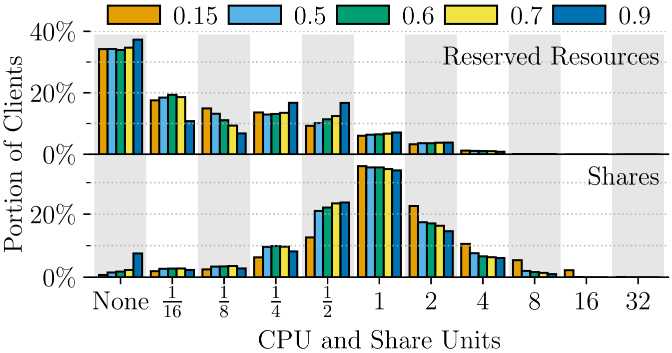
Distribution of burstable performance bundle



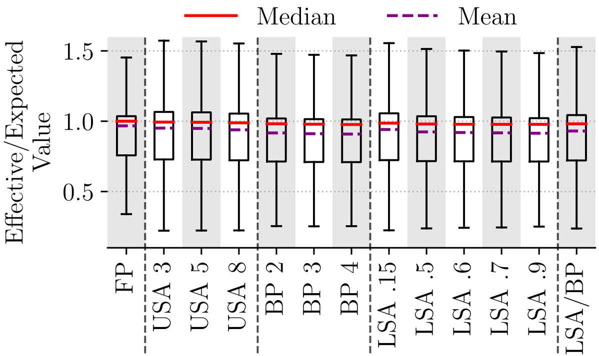
Distribution of unlimited shares bundle



Distribution of limited shares bundle



Effective vs. expected Clients' Revenue



Validation (1)

- Over the course of the last 60 iterations, up to 12% of the clients changed their selected bundle from the first iteration to the last, in all the tested cases.
- The standard deviation of the selected bundles' distribution over these iterations was under 0.6% and the standard deviation of the shares CDF was under 0.01%, in all the tested cases.

Validation (2)

- When we modified the **beta density** to be 0.5, 10 or 50, CPS was increased by up to 6% for SA, on the one hand, and reduced by up to 5% for BP, on the other, compared with the main value (1).
- When we avoided the **over-the-top** extrapolation of the generated load values, CPS was reduced by up to 7%.
- When we modified the **performance functions** so they were linear and concave, CPS was reduced by up to 3% compared with monotonically increasing ones.
- When we modified the **Pareto index**, CPS was reduced by up to 6% for a Pareto index of 0.8 and increased by up to 1% for an index of 1.3, compared with the main index (1.1).

Validation (3)

- We also modified the number of clients that can **change their bundle**.
- The average CPS was not affected when 384 (or less) clients changed their bundle at once.
- When more than 256 clients changed their bundle, however, the results fluctuated.
- When more than 384 clients changed their bundle, the results failed to converge.
- Throughout the above-mentioned modifications, this ratio turned out higher than in the main results presented earlier.