Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms

Marcus Fontoura

Eli Cortez, Anand Bonde, Alexandre Muzio, Thomas Moscibroda, Gabriel Magalhaes, Mark Russinovich, Ricardo Bianchini
Outline

**Motivation**
Container Scheduler
Characterization Azure VM Workload
Resource Central
Evaluation
Demo
Taxonomy
Conclusions
Machine learning everywhere

**ML-based services:**
- Image recognition in Facebook Moments
- Video analysis in YouTube captions

**ML techniques:**
- Regression
- Classification
- Reinforcement learning

We can leverage ML techniques to optimize the cloud platforms that run these services

- Correlation analysis in movie recommendations

...
Public cloud platforms

- Microsoft Azure
- Google Cloud Platform
- Amazon Web Services

Perf, reliability, availability

Cost $$$
Lower Costs Via Resource Management

Pack VMs tightly
Oversubscribe resources
Increase server density
Reduce energy consumption
Reduce management overhead
Scavenge idle resources

Practical challenges:
- Complexity and scale
- VM performance impact
- VM availability impact
Lower Costs Via Resource Management

Pack VMs tightly
Oversubscribe resources
Reduce management overhead
Scavenge idle resources

Practical challenges:

We can address these challenges by deeply understanding and predicting the characteristics of the VM workload!
Overview of the Azure Compute platform

Virtual machine (VM) offerings:
- IaaS, PaaS, and SaaS VMs
- Diverse workloads
- Massive scale
- Expensive to build and operate

Where and how should we add ML intelligence to lower costs without hurting QoS?

Resource managers:
- Expensive to build and operate

Diagram showing resource managers: Container Repairs, Container Scheduler, Network Control Plane, Migration Manager, Physical Resource Management: Device Repairs, Deployment, Power Manager, Bootstrapping, Device Manager, Health Store, PKI
Where? Many managers can benefit

Container scheduler
- Pack tightly [ASPLOS’13]
- Oversubscribe [Later, SOSP’17]
- Scavenge [OSDI’16]

Power manager
- Cap power
- Save energy [Google]

Migration manager
- Defragment servers
- Free up misbehaving servers

Practical challenges:
- Complexity and scale
- No info about apps
- Performance impact
- Availability impact

ML can help!

We need a general framework
Outline

Motivation

**Container Scheduler**

Characterization Azure VM Workload

Resource Central

Evaluation

Demo

Taxonomy

Conclusions
Virtual Machine Types

Azure has several VM families, for instance:

<table>
<thead>
<tr>
<th>Type</th>
<th>Cores</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>1</td>
<td>0.768</td>
</tr>
<tr>
<td>A1</td>
<td>1</td>
<td>1.75</td>
</tr>
<tr>
<td>A2</td>
<td>2</td>
<td>3.5</td>
</tr>
<tr>
<td>A3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>A4</td>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>A5</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>A6</td>
<td>4</td>
<td>28</td>
</tr>
<tr>
<td>A7</td>
<td>8</td>
<td>56</td>
</tr>
<tr>
<td>A8</td>
<td>8</td>
<td>56</td>
</tr>
<tr>
<td>A9</td>
<td>16</td>
<td>112</td>
</tr>
<tr>
<td>A10</td>
<td>8</td>
<td>56</td>
</tr>
<tr>
<td>A11</td>
<td>16</td>
<td>112</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Cores</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>1</td>
<td>3.5</td>
</tr>
<tr>
<td>D2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>D3</td>
<td>4</td>
<td>14</td>
</tr>
<tr>
<td>D4</td>
<td>8</td>
<td>28</td>
</tr>
<tr>
<td>D11</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>D12</td>
<td>4</td>
<td>28</td>
</tr>
<tr>
<td>D13</td>
<td>8</td>
<td>56</td>
</tr>
<tr>
<td>D14</td>
<td>16</td>
<td>112</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Cores</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>2</td>
<td>28</td>
</tr>
<tr>
<td>G2</td>
<td>4</td>
<td>56</td>
</tr>
<tr>
<td>G3</td>
<td>8</td>
<td>112</td>
</tr>
<tr>
<td>G4</td>
<td>16</td>
<td>224</td>
</tr>
<tr>
<td>G5</td>
<td>32</td>
<td>448</td>
</tr>
</tbody>
</table>

- **A**: High-Value
- **D**: Low-Latency, SSD
- **G**: Extreme Performance, SSD

Infiniband

- High Memory
- Faster CPUs

Cores | Memory | SSD

---

VM VM VM VM VM
Virtual Machine Architecture

- Network, local and remote storage are additional allocation dimensions

- Ephemeral storage: uses local storage bandwidth and space
  - Backed by local HDD or SSD

- Persistent storage: uses network bandwidth
  - Cached on local server RAM, HDD or SSD
  - Backed by Azure Storage page blobs
  - “S” variants (e.g. “DS14”) can use SSD-backed Premium Storage
Fabric Clusters

- Fabric Controller: Hardware and VM manager for a “cluster” of servers
  - Uses 5-server Paxos-type replication for high availability
  - Exposes API for deploying, deleting and updating VMs
  - Keeps track of server and VM health

- Fabric Controller can autonomously “heal” a VM
  - Detects server has failed and restarts VM on a healthy server
Container Scheduler

- Composed of cluster-selection, admission-control, and intra-cluster allocation algorithms

- Multi-level:
  - First, select FC cluster
  - Then, FC cluster allocator places VMs on servers
Constraints

- Placement constraints
  - Resource constraints: The sum of resources of all VMs on a node cannot exceed server resources (CPU, memory, disk, SSD, network IO,...) → Bin-Packing
  - Failure domain constraint: VMs of the same tenant must be spread across many failure domains
  - Co-location constraints: Certain types of VMs cannot be co-located together
Resource Utilization

• VM Packing should achieve high utilization across all resource dimensions
  Multi-dimensional resource packing

Container scheduler should be aware of Multiple Resource Dimensions:

• We use multi-dimensional best-fit.
  [Heuristics for Vector Bin Packing, Panigrahy et al., MSR Tech Report 2011]
  • Each resource dimension $d$ is assigned a weight $w_d \rightarrow$ scarcity of the resource.
  • $r_d$ is the residual resource of a node
  • Allocate the VM to the node that minimizes $\sum_d w_d \cdot r_d$
Multi-Dimension Optimization

- Container scheduling should achieve high utilization across all resource dimensions
  1. Multi-dimensional resource packing
  2. Take into account online nature of service allocation

- Simple example: Assume every VM has probability of $\frac{1}{2}$ of leaving until time $T$.
- Probability that we can deploy VM$_b$?
  - If new VM is placed on Node 1:
    \[
    \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3 = \frac{6}{16}
    \]
  - If new VM is placed on Node 2:
    \[
    \left(\frac{1}{2}\right)^4 = \frac{9}{16}
    \]

→ Placing new VM on Node 2 is better!
Resource utilization in Azure

• Each 1% of utilization gain results in millions of $ savings

Container scheduling algorithms are crucial for operating Azure effectively!
Outline

Motivation

Container Scheduler

**Characterization Azure VM Workload**

Resource Central

Evaluation

Demo

Taxonomy

Conclusions
Background: Main Azure characteristics

Azure hosts:
- 1<sup>st</sup>-party VMs – Microsoft dev, test, internal services
- 1<sup>st</sup>-party services offered to 3<sup>rd</sup>-party customers – Office 365, Xbox, Skype, ...
- 3<sup>rd</sup>-party VMs – External users’ VMs, Daimler, Geico, Adobe, ...

Customers create “subscriptions”, deploy VMs to regions in “deployments”

Our study: Full VM workload of Azure over 3 months (trace available!)
Characterization – VM size (CPU cores)

Observations:

• Small VMs with scale-out pattern
• CPU cores and memory are correlated
• 1st- and 3rd-party are similar

Resource management:

• Easier to fill holes
Characterization – VM CPU utilization

Observations:
- Large % with low avg. utilization
- Large % with high P95 util., esp. 3rd party
- Large % with low utilization even at P95

Resource management:
- High utils ➔ may limit packing
- Low utils ➔ oversubscription is possible
Observations:
• Short VMs dominate, esp. for 1st-party
• Non-trivial percentage of long VMs
• Long VMs = 95% of core hours!

Resource management:
• If VM lasts 1 day, it will live much longer
• Non-urgent maintenance
• Lifetime-aware VM scheduling
Other VM workload characteristics

VM type (IaaS vs PaaS)
VM memory size
VM deployment size
VM arrivals
VM workload class (interactive vs delay-insensitive)

Correlations between characteristics

Please refer to our paper for details
Outline

Motivation
Container Scheduler
Characterization Azure VM Workload
Resource Central
Evaluation
Demo
Taxonomy
Conclusions
Resource Central

ML and prediction-serving system for improving resource management

Potential RC clients: Platform resource managers

- VM scheduling
- Cluster selection
- Power oversubscription
- Server maintenance
- VM rightsizing recommendation
Resource Central architecture

**Design principles:**
- Off critical perf & availability paths
- Simple; based on stable systems
- General; easy to use by clients

**Status:**
- Manually used by engineers
- Clients in production

**Offline**

- Telemetry data
- Data aggregation, cleanup, and validation
  - ML model training, generation, validation
  - Feature data generation, validation

**Online**

- Resource manager client, e.g. VM scheduler
- Prediction-serving: Model and data caching
- Highly available store
- Persistent local cache
Current ML models

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Modeling approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU utilization</td>
<td>Random Forests</td>
</tr>
<tr>
<td>Deployment size</td>
<td>Extreme Gradient Boosting Trees</td>
</tr>
<tr>
<td>Lifetime</td>
<td>Extreme Gradient Boosting Trees</td>
</tr>
<tr>
<td>Workload class</td>
<td>FFT, Extreme Gradient Boosting Tree</td>
</tr>
</tbody>
</table>

Classification algorithms

- Numeric models predict “buckets”
- Prediction comes with a “confidence score”
Outline

Motivation
Container Scheduler
Characterization Azure VM Workload
Resource Central

Evaluation
Demo
Taxonomy
Conclusions
Prediction quality

Accuracy ≥ 79%

Precision$^\theta$ ≥ 85%

Recall$^\theta$ ≥ 73%
Prediction - VM CPU P95 max

- Overall accuracy = 0.83
- \( Precision^{\theta} = 0.94 \)
- \( Recall^{\theta} = 0.73 \)

Important attributes:
- % previous VMs in bucket (subscription)
- Operating system

Deployment time is irrelevant
Performance – Model Execution

- Low latency
- Predictable
- 99th percentile: 258 µsec max
Deep Learning in RC

Azure Batch AI

Azure Workload Information
Telemetry, Metadata

Deep Learning Models

Keras

Deep Learning Models
Deep Learning in RC

Task: VM Lifetime Prediction

**Inputs:**
(~500 features)

- VP Count
- Memory
- OS
- VM Type
- Subscription
  (...)

**Output (classification):**
VM Lifetime (in 4 buckets)

Neural net

Activation Function: LeakyRelu
Prediction Quality

Accuracy ≥ 83%
Precision\(^\theta\) ≥ 87%
Recall\(^\theta\) ≥ 89%
Case study: Smart CPU oversubscription

Goals:
- Be conservative! Stick with P95, 1st-party loads
- Don’t oversubscribe servers running prod VMs
- Oversubscribe other servers up to a percentage over capacity and a max predicted (P95) utilization

New rule checking the sum of the P95 utilizations

Mispredictions: only issue is consistent under-prediction
### RC-informed CPU oversubscription

#### Simulation results

<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>No oversubscription</td>
<td>Low capacity; many VM allocation failures</td>
</tr>
<tr>
<td>Naive</td>
<td>25% oversub without predictions</td>
<td>No failures; 6x resource exhaustion</td>
</tr>
<tr>
<td>RC-informed</td>
<td>25% oversub with RC predictions</td>
<td>No failures; rare exhaustion</td>
</tr>
<tr>
<td>RC-right</td>
<td>25% oversub with oracle predictions</td>
<td>No failures; same exhaustion</td>
</tr>
</tbody>
</table>
Multi-Dimension Optimization

- Container scheduling should achieve high utilization across all resource dimensions
  1. Multi-dimensional resource packing
  2. Take into account online nature of service allocation

- **Simple example:** Assume every VM has

  Lifetime prediction is important for container scheduling

\[
\left( \frac{1}{2} \right)^2 + \left( \frac{1}{2} \right)^3 = \frac{6}{16}
\]

- If new VM is placed on Node 2:

\[
\left( \frac{1}{2} \right)^2 + \left( \frac{1}{2} \right)^4 = \frac{9}{16}
\]

→ Placing new VM on Node 2 is better!
# Production Dashboard

**Resource Central - Short Lived VM Prediction Quality in Production**

<table>
<thead>
<tr>
<th>Date</th>
<th>VMsCreatedCount</th>
<th>ShortLivedVMsCount</th>
<th>ShortLivedResourceCentralPredictedCount</th>
<th>ShortLivedAndPredictedCount</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/2/2018 12:00:00 AM</td>
<td>1503947</td>
<td>1222003</td>
<td>608535</td>
<td>801188</td>
<td>0.99</td>
<td>0.56</td>
</tr>
<tr>
<td>4/4/2018 12:00:00 AM</td>
<td>1823831</td>
<td>1536078</td>
<td>948810</td>
<td>941233</td>
<td>0.99</td>
<td>0.61</td>
</tr>
<tr>
<td>4/5/2018 12:00:00 AM</td>
<td>1832023</td>
<td>1549938</td>
<td>1002354</td>
<td>994740</td>
<td>0.99</td>
<td>0.64</td>
</tr>
<tr>
<td>4/6/2018 12:00:00 AM</td>
<td>1618996</td>
<td>1344647</td>
<td>828360</td>
<td>829911</td>
<td>0.99</td>
<td>0.62</td>
</tr>
<tr>
<td>4/7/2018 12:00:00 AM</td>
<td>1105448</td>
<td>1018982</td>
<td>617846</td>
<td>607752</td>
<td>0.99</td>
<td>0.60</td>
</tr>
<tr>
<td>4/8/2018 12:00:00 AM</td>
<td>1137287</td>
<td>991428</td>
<td>611711</td>
<td>607731</td>
<td>0.99</td>
<td>0.61</td>
</tr>
<tr>
<td>4/9/2018 12:00:00 AM</td>
<td>1790569</td>
<td>1451170</td>
<td>887340</td>
<td>880931</td>
<td>0.99</td>
<td>0.61</td>
</tr>
<tr>
<td>4/10/2018 12:00:00 AM</td>
<td>1801473</td>
<td>1503857</td>
<td>982545</td>
<td>972590</td>
<td>0.99</td>
<td>0.65</td>
</tr>
<tr>
<td>4/11/2018 12:00:00 AM</td>
<td>1606677</td>
<td>1302024</td>
<td>760178</td>
<td>728059</td>
<td>0.88</td>
<td>0.57</td>
</tr>
<tr>
<td>4/12/2018 12:00:00 AM</td>
<td>1817186</td>
<td>1457009</td>
<td>868355</td>
<td>860817</td>
<td>0.99</td>
<td>0.59</td>
</tr>
<tr>
<td>4/13/2018 12:00:00 AM</td>
<td>1736058</td>
<td>1406395</td>
<td>856522</td>
<td>845358</td>
<td>0.99</td>
<td>0.58</td>
</tr>
<tr>
<td>4/14/2018 12:00:00 AM</td>
<td>1221487</td>
<td>1092817</td>
<td>641981</td>
<td>639921</td>
<td>0.99</td>
<td>0.60</td>
</tr>
</tbody>
</table>

---

**Graph**

- **Precision**
- **Recall**
- **VMsCreatedCount**
- **ShortLivedVMsCount**

![Graph Image](image-url)
Demo
Approaches to adding ML

Passive, external to managers:
- Predict load intensity, utilization
- Cluster workloads, resources
- ML as an insight provider

Active, built into managers:
- Adjust parameters of policies
- Select actions to be performed
- ML has deep knowledge of policies

I = Inputs; A = Actions
RM = Resource Manager

Debuggable; simpler RM.s
Along a different dimension

**Iterative observe and decide:**
After each action, observe & decide
Management as a control problem

**Delayed observation:**
Generate model offline, run it online
Re-generate model periodically

I = Inputs; A = Actions
RM = Resource Manager

Simpler
Summary of our approach

A general, passive and delayed-observation framework for all ML tasks

Management
Container scheduling

Useful predictions
VM resource utilization

Scavenging
VM workload class

We are building Resource Central and modifying resource managers to use its predictions in Azure Compute
RC at the center of Azure Compute
Conclusions

ML can improve resource management in cloud platforms

Understanding cloud workload is key for identifying improvements

Resource Central produces high quality workload predictions

Passive and delayed-observation framework is simpler. Scale is the problem!

Predictions enable lower costs while retaining good QoS
Thanks

Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms

VM Traces -- https://github.com/Azure/AzurePublicDataset/

Marcus Fontoura

Eli Cortez, Anand Bonde, Alexandre Muzio, Thomas Moscibroda, Gabriel Magalhaes, Mark Russinovich, Ricardo Bianchini