Cloud providers aim to make more money out of the same hardware.

Rigid allocation prevents optimal resource utilization.

Cloud providers aim to make more money off the same hardware
Cloud providers aim to make more money off the same hardware

Rigid allocation prevents optimal resource utilization

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 hypervisor, delivering practically:

- Burstable performance offers CPU elasticity
- Clients can "burst" to a higher level when required
- Allow changing resource consumption on the fly
- Exploiting resources that are momentarily unused by others
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 hypervisor, delivering practically

Introducing Amazon EC2 T3 Instances
Posted On: Aug 21, 2018

- Burstable performance offers CPU elasticity
- Clients can "burst" to a higher level when required
  - Allow changing resource consumption on the fly
  - Exploiting resources that are momentarily unused by others

- More clients can be allocated to the same physical servers

Memory is the new bottleneck

- It is an expensive resource that limits machine occupancy
Memory is the new bottleneck

- It is an expensive resource that limits machine occupancy
- Memory elasticity schemes should be a natural extension to CPU elasticity
  - Allowing clients to use more memory in the same VM/container than their initial memory allocation
Memory Elastic Applications

- Applications that can burst
Memory Elastic Applications

Applications that can burst

- Apache
- Spark
- Apache HTTP Server Project
- N

Memory Elasticity Benchmark
Memory Elastic Applications

- Applications that can burst
- Whose performance is proportional to their memory usage
Memory-elastic applications are scarce
  ▶ Maximal memory footprint is dictated by the current application workload
  ▶ The OS’s swapping allows seamless application operation
  ▶ Even a minor memory loss may degrade the performance significantly
Where are the Memory Elastic Applications?

- Why most applications can scale with CPU? But not for memory?
Where are the Memory Elastic Applications?

- Why most applications can scale with CPU? But not for memory?
- Multi-core architectures and CPU schedulers were the incentive
Circular Dependency

Evaluation requires benchmarks

Making applications memory elastic

Putting effort requires incentive

Developing memory elasticity systems

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

Memory Elasticity Benchmark
Mechanisms that were designed to allow trade-off between memory and other resources can be used to provide memory elasticity.
Memory as Cache

Applications that use the RAM to cache computation results, network traffic, and so on (e.g., using Memcached)

- Improve cache hit-rate when more memory is available to the operating system

![Graph showing the relationship between memory (MB) and items per second]
Intermediate Buffers

Applications that use intermediate buffers (e.g., Hadoop, Spark)

- Can use larger memory buffers to reduce disk access and speed up temporarily data-heavy operations
  - E.g., sorting and large matrix multiplication

![Graph showing TPS vs Memory (MB)]
Applications with automatic memory management (e.g., Java applications)

- May need fewer garbage-collection cycles with a larger heap, and improve their performance
Multiple Short-Lived Jobs

Applications that have multiple short-lived jobs, each with different memory requirements (e.g., Nginx)

- Web servers might require a certain memory to handle each session
- They may be able to handle more concurrent sessions when more memory is available
Memory-Aware Applications

- Memory-aware applications adjust their memory consumption according to the available memory observed during their initiation period
  - But cannot adjust it during runtime
- Most of the commonly used memory trade-offs we mentioned are predefined and implemented as memory-aware applications
Memory-Aware Applications

- Memory-aware applications adjust their memory consumption according to the available memory observed during their initiation period
  - But cannot adjust it during runtime
- Most of the commonly used memory trade-offs we mentioned are predefined and implemented as memory-aware applications
- Can be made memory-elastic by restarting them when the memory changes
  - Not suitable when the application needs to be continuously available
Memory-Aware Applications

- Memory-aware applications adjust their memory consumption according to the available memory observed during their initiation period
  - But cannot adjust it during runtime
- Most of the commonly used memory trade-offs we mentioned are predefined and implemented as memory-aware applications
- Can be made memory-elastic by restarting them when the memory changes
  - Not suitable when the application needs to be continuously available
- With a small effort, these applications can be tweaked to become memory-elastic
Elastic memcached supports changing its memory footprint upon receiving a command via a socket.
- Compare the performance of two applications under the same dynamic memory conditions and consider the one with the better results as more memory-elastic.
▶ Compare the performance of two applications under the same dynamic memory conditions and consider the one with the better results as more memory-elastic

▶ The results may be sensitive to the order or frequency of memory allocations
Compare the performance of two applications under the same dynamic memory conditions and consider the one with the better results as more memory-elastic.

The results may be sensitive to the order or frequency of memory allocations.
Naive Metric

- Compare the performance of two applications under the same dynamic memory conditions and consider the one with the better results as more memory-elastic
- The results may be sensitive to the order or frequency of memory allocations
Compare the performance of two applications under the same dynamic memory conditions and consider the one with the better results as more memory-elastic.

The results may be sensitive to the order or frequency of memory allocations.

This is because we try to infer memory elasticity from observations of metrics that only hint about elasticity, but do not measure it directly.
Our goal is
- To quantify an application’s behavior in a dynamic memory scenario
- To compare it to other applications
- Using metrics that directly relate to memory elasticity
Static Metrics

- Static memory → performance function \( P_{\text{mem}} \) that describes the performance of the application given a static memory allocation

- Elasticity domain: \([\text{mem}_L, \text{mem}_H]\)

- Elasticity range: \( \text{mem}_H - \text{mem}_L \)

![Diagram showing memory usage and performance](image)
Memcached Static Metrics

- Elasticity domain: from 1 GB to 3.5 GB
- Elasticity range: 2.5 GB
Dynamic Metrics

Phase A: $\text{mem} = \alpha$

Phase B: $\text{mem} = \beta$

Phase C: $\text{mem} = \alpha$

$T_{\text{mem}}(\alpha, \beta)$

$T_{\text{mem}}(\beta, \alpha)$

Performance $P_{\text{mem}}(\alpha)$

Performance $P_{\text{mem}}(\beta)$

Increase memory

Decrease memory

L. Funaro, O. Agmon Ben-Yehuda, A. Schuster (Technion)
Performance Loss During the Transient Period (1)

\[
\begin{align*}
P_{\text{mem}}(l) & \\
P_{\text{mem}}(h) & \\
0 & \leq T_{\text{mem}}
\end{align*}
\]

- A Perf.
- B Perf.
- A Perf. Loss
- B Perf. Loss

Performance vs. Time

L. Funaro, O. Agmon Ben-Yehuda, A. Schuster (Technion)
Performance Loss During the Transient Period (1)

\[ P_{\text{mem}}(l) \]

\[ P_{\text{mem}}(h) \]

\[ T_{\text{mem}} \]

A Perf.

B Perf.

A Perf. Loss

B Perf. Loss

L. Funaro, O. Agmon Ben-Yehuda, A. Schuster (Technion)
Performance Loss During the Transient Period (2)
Performance Loss During the Transient Period (2)

![Diagram showing performance metrics and loss during the transient period.]

L. Funaro, O. Agmon Ben-Yehuda, A. Schuster (Technion)
Performance Loss During the Transient Period (2)
Performance Loss During the Transient Period (3)

\[ P_{mem}(l) \]

\[ P_{mem}(h) \]

\[ A: E_{mem} \]

\[ B: E_{mem} \]

\[ T_{mem} \]
Elastic Memcached Average Measured $E_{mem}$

<table>
<thead>
<tr>
<th>From Memory (MB)</th>
<th>To Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>896</td>
<td>896</td>
</tr>
<tr>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>1536</td>
<td>1536</td>
</tr>
<tr>
<td>2048</td>
<td>2048</td>
</tr>
<tr>
<td>2560</td>
<td>2560</td>
</tr>
<tr>
<td>3072</td>
<td>3072</td>
</tr>
<tr>
<td>3584</td>
<td>3584</td>
</tr>
</tbody>
</table>

$E_{mem}$ (Seconds)

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

Memory Elasticity Benchmark
Elastic vs. Off-the-shelf Memcached

Elastic

<table>
<thead>
<tr>
<th>To Memory (MB)</th>
<th>3584</th>
<th>3072</th>
<th>2560</th>
<th>2048</th>
<th>1536</th>
<th>1024</th>
<th>896</th>
</tr>
</thead>
<tbody>
<tr>
<td>3584</td>
<td>112</td>
<td>96</td>
<td>87</td>
<td>71</td>
<td>59</td>
<td>48</td>
<td>8</td>
</tr>
<tr>
<td>3072</td>
<td>90</td>
<td>89</td>
<td>69</td>
<td>63</td>
<td>47</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>2560</td>
<td>83</td>
<td>73</td>
<td>64</td>
<td>31</td>
<td>9</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>2048</td>
<td>54</td>
<td>60</td>
<td>19</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>1536</td>
<td>35</td>
<td>31</td>
<td>8</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>1024</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>896</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Off-the-shelf

<table>
<thead>
<tr>
<th>To Memory (MB)</th>
<th>3584</th>
<th>3072</th>
<th>2560</th>
<th>2048</th>
<th>1536</th>
<th>1024</th>
<th>896</th>
</tr>
</thead>
<tbody>
<tr>
<td>3584</td>
<td>167</td>
<td>185</td>
<td>249</td>
<td>327</td>
<td>598</td>
<td>1321</td>
<td>8</td>
</tr>
<tr>
<td>3072</td>
<td>152</td>
<td>177</td>
<td>225</td>
<td>367</td>
<td>748</td>
<td>1070</td>
<td>1250</td>
</tr>
<tr>
<td>2560</td>
<td>160</td>
<td>147</td>
<td>233</td>
<td>569</td>
<td>527</td>
<td>374</td>
<td>1000</td>
</tr>
<tr>
<td>2048</td>
<td>116</td>
<td>153</td>
<td>286</td>
<td>344</td>
<td>163</td>
<td>133</td>
<td>750</td>
</tr>
<tr>
<td>1536</td>
<td>97</td>
<td>153</td>
<td>130</td>
<td>77</td>
<td>52</td>
<td>61</td>
<td>500</td>
</tr>
<tr>
<td>1024</td>
<td>164</td>
<td>58</td>
<td>25</td>
<td>18</td>
<td>12</td>
<td>14</td>
<td>250</td>
</tr>
<tr>
<td>896</td>
<td>98</td>
<td>21</td>
<td>14</td>
<td>17</td>
<td>12</td>
<td>14</td>
<td>250</td>
</tr>
</tbody>
</table>
Conclusions

- We showed a few major building blocks that can be made memory elastic
  - Cache, intermediate buffers, garbage-collection and schedulers
- We defined metrics that are comparable across applications
  - Elasticity range and $E_{mem}$
- We defined characteristics that can be used by clients to configure their virtual machine and their application in a memory elastic cloud environment
  - $P_{mem}$ and $T_{mem}$
- Our framework is available from
  github.com/liran-funaro/elastic-benchmarks

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