Ginseng: Market-Driven LLC Allocation

Liran Funaro  Orna Agmon Ben-Yehuda  Assaf Schuster

Department of Computer Science

USENIX ATC ’16
Infrastructure-as-a-Service (IaaS) Model

- IaaS cloud providers rent a bundle of resources in the form of a guest virtual machine (VM) to their clients.

- Cloud clients need to rent VMs with the resources to sustain their highest workload.

- They will prefer to rent resources only when it is really necessary.
  - This will reduce idle resources.
  - Hence, the provider can consolidate more clients per physical machine.
The future of the Infrastructure-as-a-Service (IaaS) cloud is the RaaS cloud, characterized by:

- Fine resource granularity
- Fine time granularity
- Market-driven resource pricing

More details in:

We want to dynamically allocate LLC using the RaaS model
- Fine allocation granularity
- Fine time granularity
- Market-driven pricing

We can utilize Intel’s new LLC allocation technology for that end
Upon a memory access, the cache follows this algorithm:

- Calculate the **set**: hash value of the memory address
- Scan the **ways** over that **set** for this memory address
- If not found:
  - Read it from the memory
  - Store it in the least-recently used (LRU) **way** over that **set**

<table>
<thead>
<tr>
<th>Way 1</th>
<th>Way 2</th>
<th>...</th>
<th>Way m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 1</td>
<td>line</td>
<td>line</td>
<td>...</td>
</tr>
<tr>
<td>Set 2</td>
<td>line</td>
<td>line</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>line</td>
<td>line</td>
<td>...</td>
</tr>
<tr>
<td>Set n</td>
<td>line</td>
<td>line</td>
<td>...</td>
</tr>
</tbody>
</table>
Upon a memory access, the cache follows this algorithm:

- Calculate the **set**: hash value of the memory address
- Scan the **ways** over that **set** for this memory address
- If not found:
  - Read it from the memory
  - Store it in the least-recently used (LRU) **way** over that **set**

<table>
<thead>
<tr>
<th>Set 1</th>
<th>Way 1</th>
<th>Way 2</th>
<th>...</th>
<th>Way m</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>line</td>
<td>line</td>
<td>...</td>
<td>line</td>
</tr>
<tr>
<td>Set 2</td>
<td>line</td>
<td>line</td>
<td>...</td>
<td>line</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set n</td>
<td>line</td>
<td>line</td>
<td>...</td>
<td>line</td>
</tr>
</tbody>
</table>

CAT allows the host to restrict the store only to a subset of ways, depending on the guest that issued the memory access.
How should we allocate the LLC in a public cloud?

- What is the benefit of each guest from the cache?
- How can the cloud provider know which guest will benefit from LLC the most?
Some applications can benefit from more cache (cache-utilizers)

Figure: Benchmarks from Phoronix Test Suite: http://www.phoronix-test-suite.com/
But not all applications can exploit the cache to increase performance (cache-neutral)

Figure: Benchmarks from Phoronix Test Suite: http://www.phoronix-test-suite.com/
Some cache-neutral applications will pollute the cache (cache-polluters)

- E.g. an application that reads or writes a stream of data will pollute the cache with this data but will not use it again in the near future

Figure: Composite-Scimark (cache-utilizer) and Monte-Carlo (cache-neutral)
How should we allocate the LLC in a public cloud?

✔ What is the benefit of each guest from the cache?

❓ How can the cloud provider know which guest will benefit from LLC the most?
White box approaches cannot work in a real commercial cloud
- What is the guest doing? What should be measured? How?
- How much is the performance worth to the client?
- Whose fault is it that the guest’s performance is low? Maybe the software is inefficient?
White box approaches cannot work in a real commercial cloud
- What is the guest doing? What should be measured? How?
- How much is the performance worth to the client?
- Whose fault is it that the guest’s performance is low? Maybe the software is inefficient?

Black box approaches cannot work in a real commercial cloud
- Guest measurements: results can be mis-reported
- Host measurements: High miss ratio can be faked to induce the host to allocate more cache
The *Ginseng* system uses an economic mechanism (VCG) that incentivizes even black-box guests to reveal how much cache is worth to them.

- **VCG**: auction mechanism designed by Vickrey (1961), Clarke (1971), Groves (1973)

Using this knowledge, *Ginseng* can find the allocation that maximizes the social welfare: sum of guest valuations.
The host announces an auction every 10 seconds.
The host announces an auction every 10 seconds

Each guest bids with a valuation for each quantity of cache ways — how much it is worth, subjectively
Clients should be able to evaluate, in economic terms, their benefit from the cache.

(a) Performance profiling

(b) Valuation

Figure: Composite-Scimark profiling and valuation function
The host announces an auction every 10 seconds

Each guest bids with a valuation for each quantity of cache ways — how much it is worth, subjectively

The host finds the allocation that maximizes the social welfare: the allocation that all the guests together value the most
Ginseng Protocol (VCG)

The host announces an auction every 10 seconds.

Each guest bids with a valuation for each quantity of cache ways — how much it is worth, subjectively.

The host finds the allocation that maximizes the social welfare: the allocation that all the guests together value the most.

The host informs the guests of their allocation and charges them according to the exclusion-compensation principle.
The exclusion-compensation principle:

- Each guest pays for the damage it inflicted on the other guests in the system

As a result:

- The guests cannot improve their status by bidding a higher or a lower value
- Prices are not uniform
- They may drop to a minimal price (possibly zero) if there is no demand for the LLC
How should we allocate the LLC in a public cloud?

- What is the benefit of each guest from the cache?
- How can the cloud provider know which guest will benefit from LLC the most?
Requirements for LLC allocation with the RaaS model:

- Fine allocation granularity
- Fine time granularity
- Market-driven pricing
Requirements for LLC allocation with the RaaS model:

- Fine allocation granularity
- Fine time granularity
- Market-driven pricing
Requirements for LLC allocation with the RaaS model:

- ✅ Fine allocation granularity
- ▶️ Fine time granularity
- ✅ Market-driven pricing
Requirements for LLC allocation with the RaaS model:

- Fine allocation granularity
- Fine time granularity
- Market-driven pricing
Reallocation of the cache should be **fast** and therefore **efficient**.
- Reallocation of the cache should be **fast** and therefore **efficient**
- The **cache leakage** effect might reduce the **efficiency** of reallocation
  - However, it does not have security implications
Dynamic, Fast Cache Reallocation

- Reallocation of the cache should be **fast** and therefore **efficient**
- The **cache leakage** effect might reduce the **efficiency** of reallocation
  - However, it does not have security implications

<table>
<thead>
<tr>
<th></th>
<th>Way 1</th>
<th>Way 2</th>
<th>Way 3</th>
<th>Way 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2 cache ways
Reallocation of the cache should be **fast** and therefore **efficient**

- The **cache leakage** effect might reduce the **efficiency** of reallocation
  - However, it does not have security implications
Dynamic, Fast Cache Reallocation

- Reallocation of the cache should be **fast** and therefore **efficient**
- The **cache leakage** effect might reduce the **efficiency** of reallocation
  - However, it does not have security implications

1 cache way

3 cache ways

Expected Outcome

<table>
<thead>
<tr>
<th>Way 1</th>
<th>Way 2</th>
<th>Way 3</th>
<th>Way 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Dynamic, Fast Cache Reallocation

- Reallocation of the cache should be **fast** and therefore **efficient**
- The **cache leakage** effect might reduce the **efficiency** of reallocation
  - However, it does not have security implications

<table>
<thead>
<tr>
<th>1 cache way</th>
<th>Way 1</th>
<th>Way 2</th>
<th>Way 3</th>
<th>Way 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Set 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Set 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Set 4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We designed an application that takes advantage of the cache leakage by
- Ensuring its data fits perfectly in its cache ways
- Repeatedly touching all its data, in parallel

We measured how repeated reallocations affect real application performance

Performance varied by up to 4% from the baseline values
- Up to 1.1% on average for all of the workloads
- Unnoticeable cache leakage in real world scenarios
Measuring the Leakage Effect

- We designed an application that takes advantage of the cache leakage by
  - Ensuring its data fits perfectly in its cache ways
  - Repeatedly touching all its data, in parallel

- We measured how repeated reallocations affect real application performance

- Performance varied by **up to 4%** from the baseline values
  - Up to 1.1% on average for all of the workloads

- **Unnoticeable** cache leakage in real world scenarios
Allocating LLC with the Resource-as-a-Service Model

Requirements for LLC allocation with the RaaS model:

- Fine allocation granularity
- Fine time granularity
- Market-driven pricing
Requirements for LLC allocation with the RaaS model:

- Fine allocation granularity
- Fine time granularity
- Market-driven pricing
Evaluating our Solution
Experimental Methodology

- Each guest VM ran one application and served 10 customers, one at the time.
- It valued each customer differently, for example:
  - High paying customers will have a high valuation.
  - Medium paying customers will have a medium valuation.
  - Non-paying customers will have a low valuation.
Figure: All guests run *Fast Fourier Transform* with 1 high-valuation customer, 1 medium-valuation customers and 8 low-valuation customers.
Figure: All guests run Fast Fourier Transform with 1 high-valuation customer, 1 medium-valuation customers and 8 low-valuation customers.
Evaluation on a Growing Number of VMs

![Graph showing social welfare against number of VMs with three different allocation methods: Shared Cache, Perf. Maximizing, and Ginseng.]

**Figure:** All guests run *Fast Fourier Transform* with 1 high-valuation customer, 1 medium-valuation customers and 8 low-valuation customers.
Compared to Performance Maximizing

Figure: Maximum improvement factor of *Ginseng* compared to the performance-maximizing method.
Figure: Maximum improvement factor of *Ginseng* compared to the shared-cache method.
Compared to Shared Cache (ZOOM)

Figure: Maximum improvement factor of *Ginseng* compared to the shared-cache method.
Ginseng efficiently allocates LLC to selfish black-box guests while maximizing their aggregate benefit

The guests utilize their cache fast enough to allow such rapid changes in the allocation without any substantial effect on their performance
Questions?

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

Some of the figures are designed using images from freepik.com and flaticon.com and licensed by CC 3.0 BY.