Cache on delivery

marco@sensepost.com
whoami

sensepost
Scalable applications / Cloud?

Essential characteristics
- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured service

Service models
- Cloud Software as a Service (SaaS)
- Cloud Platform as a Service (PaaS)
- Cloud Infrastructure as a Service (IaaS)

Deployment models
- Private cloud
- Community cloud
- Public cloud
- Hybrid cloud

http://csrc.nist.gov/groups/SNS/cloud-computing/
Cloud options

http://www.flickr.com/photos/eli_k_hayasaka/764416130/

Tuesday 20 July 2010
The need for caching

- Large percentage of data remains relatively constant
  - Wikipedia page contents
  - Youtube video links
  - FB Profile data
- Poorly designed solutions regenerate data on each request
- Don’t regenerate, rather regurgitate
- Caching! = buffering
80% of WikiMedia's content is served by Squid.

~80% of Wikimedia’s content is served by Squid

Caching solutions

At all layers, there are caches

- Hard disk cache: < 64MB
- CPU Cache: < 32MB
- Caching proxies: GBs-TBs
- Cached scripts/pages: MBs-GBs
- Cached database queries / computations: MBs-GBs
- Browser caches: MBs-GBs
Caching solutions

Let’s focus on the application layer (too many options)

<table>
<thead>
<tr>
<th>Redis</th>
<th>Persistent KV Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ehcache</td>
<td>Persistent Store</td>
</tr>
<tr>
<td>Memcache</td>
<td>KV Store</td>
</tr>
<tr>
<td>MemcacheDB</td>
<td>Persistent KV Store</td>
</tr>
<tr>
<td>Websphere eXtreme Scale</td>
<td>Obj Store</td>
</tr>
<tr>
<td>Oracle Coherence</td>
<td>Obj Store</td>
</tr>
<tr>
<td>Google BigTable</td>
<td>Persistent Store</td>
</tr>
</tbody>
</table>

Tuesday 20 July 2010
Caching solutions

Let's focus on the application layer (too many options)

<table>
<thead>
<tr>
<th>Memcache</th>
<th>Persistent KV Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ehcache</td>
<td>Persistent Store</td>
</tr>
<tr>
<td>Redis</td>
<td>KV Store</td>
</tr>
<tr>
<td>Persistent KV Store</td>
<td>Obj Store</td>
</tr>
<tr>
<td>MemcacheDB</td>
<td>Persistent KV Store</td>
</tr>
<tr>
<td>Websphere eXtreme Scale</td>
<td>Obj Store</td>
</tr>
<tr>
<td>Oracle Coherence</td>
<td>Persistent Store</td>
</tr>
<tr>
<td>Google BigTable</td>
<td>Persistent Store</td>
</tr>
</tbody>
</table>

Tuesday 20 July 2010
Memcache

- memcache.org
- Written for early LJ
- Non-persistent network-based KV store
- [setup+usage demo]
Basic KV

- Slabs are fixed size
- Dst slab determined by value size
- Users don’t care about slabs
- Miners care about slabs
function get_foo(foo_id)
    foo = memcached_get("foo:" . foo_id)
    return foo if defined foo

    foo = fetch_foo_from_database(foo_id)
    memcached_set("foo:" . foo_id, foo)
    return foo
end
Trivial protocol

• ASCII-based
• Long-lived
• Tiny command set

• ????
• set
• get
• stats
• ...

Binary and UDP protocols also exist, these were not touched.
Trivial protocol

- ASCII-based
- Long-lived
- Tiny command set
- ????
- set
- get
- stats
- ...

Binary and UDP protocols also exist, these were not touched.
Trivial protocol

- ASCII-based
- Long-lived
- Tiny command set
  - ???
  - set
  - get
  - stats
  - ...

Binary and UDP protocols also exist, these were not touched.
Trivial protocol

- ASCII-based
- Long-lived
- Tiny command set
- ????
- set
- get
- stats
- ...

Binary and UDP protocols also exist, these were not touched.
Trivial protocol

- ASCII-based
- Long-lived
- Tiny command set

- ???
- set
- get
- stats
- ...

Binary and UDP protocols also exist, these were not touched.
Goals

• Connect to memcached
• Find all slabs
• Retrieve keynames from each slab
• Retrieve each key
Lies, damn lies, and stats

- stats cmd has subcmds
  - items
  - slabs
  - ...

This gets us the slabs_ids

stats slabs
STAT 1:chunk_size 80
<...>
STAT 2:chunk_size 104
<...>
STAT 3:chunk_size 136
<...>
STAT 4:chunk_size 176
<...>
STAT 6:chunk_size 280
<...>
STAT 8:chunk_size 440
<...>
STAT 9:chunk_size 552
<...>
STAT 9:cas_badval 0
STAT active_slabs 7

Tuesday 20 July 2010
Retrieving key names

Rely on two poorly documented features
Retrieving key names

Feature #1:
Remote enabling of debug mode
Retrieving key names

Feature #2:

“stats cachedump”
Retrieving key names

Feature #2:

“stats cachedump”
Retrieving key names

Feature #2:

“stats cachedump”
Retrieving key names

Feature #2:

“stats cachedump”
Retrieving key names

Feature #2:

“stats cachedump”
Retrieving key names

Feature #2:

“stats cachedump”

This gets us key names
And this gets us?

- No need for complex hacks. Memcached serves up all its data for us.

- What to do in an exposed cache?
  - Mine
  - Overwrite
Mining the cache

- go-derper.rb – memcached miner
- Retrieves up to $k$ keys from each slab and their contents, store on disk
- Applies regexes and filters matches in a *hits* file
- Supports easy overwriting of cache entries
- [demo]
Finding memcaches

- Again with the simple approach
- Pick an EC2 subnet, scan for memcaches Port 11211 and mod’ed .nse
- Who’s %#^&ing cache is this?
- Where’s the good stuff?
- Is it live?
Results

- Objects found
- Serialized Java
- Pickled Python
- Ruby ActiveRecord
- .Net Object
- JSON
Results: Actual Sites

• [screenshots in the talk]
Fixes?

- FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW. FW.

- Hack code to disable stats facility (but doesn’t prevent key brute-force)

- Hack code to disable remote enabling of debug features

- Switch to SASL
  - Requires binary protocol
  - Not supported by a number of memcached libs

- Also, FW.
Places to keep looking

- Improve data detection/sifting/filtering
- Spread the search past a single EC2 subnet
- Caching providers (?!?!)
- Other cache software
Questions?

sensepost.com/blog