Scaling Your Cache

Alex Miller
@puredanger
Mission

• Why does caching work?
• What’s hard about caching?
• How do we make choices as we design a caching architecture?
What is caching?
Lots of data
Memory Hierarchy

Clock cycles to access

- Register: 1
- L1 cache: 3
- L2 cache: 15
- RAM: 200
- Disk: 100,000,000
- Remote disk: 100,000,000,000
Facts of Life

- Register
- L1 Cache
- L2 Cache
- Main Memory
- Local Disk
- Remote Disk

- Fast
- Small
- Expensive

- Slow
- Big
- Cheap
Caching to the rescue!
Temporal Locality

Hits: 

Cache: 

Stream: 

0%
Temporal Locality

Stream:

Cache:

Hits: 0%

Stream:

Cache:

Hits: 65%
Non-uniform distribution

Web page hits, ordered by rank

Page views, ordered by rank

- Pageviews per rank
- % of total hits per rank
Temporal locality
+
Non-uniform distribution
17000 pageviews
assume avg load = 250 ms

cache 17 pages / 80% of views
cached page load = 10 ms
new avg load = 58 ms

trade memory for latency reduction
The hidden benefit: reduces database load

Database

Memory

line of over provisioning
A brief aside...

- What is Ehcache?
- What is Terracotta?
CacheManager manager = new CacheManager();
Ehcache cache = manager.getEhcache("employees");
cache.put(new Element(employee.getId(), employee));
Element element = cache.get(employee.getId());

<cache name="employees"
maxElementsInMemory="1000"
memoryStoreEvictionPolicy="LRU"
eternal="false"
timeToIdleSeconds="600"
timeToLiveSeconds="3600"
overflowToDisk="false"/>
Terracotta
But things are not always so simple...
Pain of Large Data Sets

- How do I choose which elements stay in memory and which go to disk?
- How do I choose which elements to evict when I have too many?
- How do I balance cache size against other memory uses?
Eviction

When cache memory is full, what do I do?

- Delete - Evict elements
- Overflow to disk - Move to slower, bigger storage
- Delete local - But keep remote data
Eviction in Ehcache

Evict with “Least Recently Used” policy:

```xml
<cache name="employees"
    maxElementsInMemory="1000"
    memoryStoreEvictionPolicy="LRU"
    eternal="false"
    timeToIdleSeconds="600"
    timeToLiveSeconds="3600"
    overflowToDisk="false"/>
```
Spill to Disk in Ehcache

Spill to disk:

```xml
<dimStore path="java.io.tmpdir"/>
<cache name="employees"
  maxElementsInMemory="1000"
  memoryStoreEvictionPolicy="LRU"
  eternal="false"
  timeToIdleSeconds="600"
  timeToLiveSeconds="3600"
  overflowToDisk="true"
  maxElementsOnDisk="1000000"
  diskExpireThreadIntervalSeconds="120"
  diskSpoolBufferSizeMB="30" />
```
Terracotta Clustering

Terracotta configuration:

```xml
<terracottaConfig url="server1:9510,server2:9510"/>
<cache name="employees"
    maxElementsInMemory="1000"
    memoryStoreEvictionPolicy="LRU"
    eternal="false"
    timeToIdleSeconds="600"
    timeToLiveSeconds="3600"
    overflowToDisk="false">
  ...
</cache>
```
Pain of Stale Data

• How tolerant am I of seeing values changed on the underlying data source?
• How tolerant am I of seeing values changed by another node?
Expiration

TTL=4

TTL=4
TTI and TTL in Ehcache

```xml
<cache
    name="employees"
    maxElementsInMemory="1000"
    memoryStoreEvictionPolicy="LRU"
    eternal="false"
    timeToIdleSeconds="600"
    timeToLiveSeconds="3600"
    overflowToDisk="false"/>
```
Replication in Ehcache

```xml
<cacheManagerPeerProviderFactory
    class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
    properties="hostName=fully_qualified_hostname_or_ip,
                peerDiscovery=automatic,
                multicastGroupAddress=230.0.0.1,
                multicastGroupPort=4446, timeToLive=32"/>

<cache name="employees" ...>
    <cacheEventListenerFactory
        class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"
        properties="replicateAsynchronously=true,
                    replicatePuts=true,
                    replicatePutsViaCopy=false,
                    replicateUpdates=true,
                    replicateUpdatesViaCopy=true,
                    replicateRemovals=true
                    asynchronousReplicationIntervalMillis=1000"/>
</cache>
```
Terracotta Clustering

Still use TTI and TTL to manage stale data between cache and data source

Coherent by default but can relax with coherentReads=’false’
Write-Through Caching

Ehcache 2.0

- Keep database in sync with database
- Cache write --> database write
- Ehcache 2.0 adds new API:
  - `Cache.putWithWriter(...)`
  - `CacheWriter`
    - `write(Element)`
    - `writeAll(Collection<Element>)`
    - `delete(Object key)`
    - `deleteAll(Collection<Object> keys)`
Write-Behind Caching

Ehcache 2.0

- Allow database writes to lag cache updates
  - Improves write latency
  - Possibly reduces overall writes
- Use with read-through cache
- API same as write-through
- Other features:
  - Batching, coalescing, rate limiting, retry
Pain of Loading

• How do I pre-load the cache on startup?
• How do I avoid re-loading the data on every node?
Persistent Disk Store

```xml
<diskStore path="java.io.tmpdir"/>
<cache name="employees"
  maxElementsInMemory="1000"
  memoryStoreEvictionPolicy="LRU"
  eternal="false"
  timeToIdleSeconds="600"
  timeToLiveSeconds="3600"
  overflowToDisk="true"
  maxElementsOnDisk="1000000"
  diskExpirationThreadIntervalSeconds="120"
  diskSpoolBufferSizeMB="30"
  diskPersistent="true" />
```
Bootstrap Cache Loader

Bootstrap a new cache node from a peer:

```xml
<bootstrapCacheLoaderFactory
class="net.sf.ehcache.distribution.RMIBootstrapCacheLoaderFactory"
properties="bootstrapAsynchronously=true,
maximumChunkSizeBytes=5000000"
propertySeparator=""," />
```

On startup, create background thread to pull the existing cache data from another peer.
Terracotta Persistence

Nothing needed beyond setting up Terracotta clustering.

Terracotta will automatically bootstrap:
- the cache key set on startup
- cache values on demand
Cluster Bulk Load

*Ehcache 2.0*

- Terracotta clustered caches only
- Performance
  - 6 nodes, 8 threads, 3M entries
  - Coherent: 211.9 sec (1416 TPS)
  - Bulk Load: 19.0 sec (15790 TPS) - 11.2x
Cluster Bulk Load

Ehcache 2.0

CacheManager manager = new CacheManager();
Ehcache cache = manager.getEhcachel("hotItems");

// Enter bulk loading mode for this node
    cache.setCoherent(false);

// Bulk load
    for(Item item : getItemHotList()) {
        cache.put(new Element(item.getId(), item));
    }

// End bulk loading mode
    cache.setCoherent(true);
Pain of Concurrency

• Locking
• Transactions
Thread safety

- Cache-level operations are thread-safe
- No public API for multi-key or multi-cache composite operations
- Provide external locking
BlockingCache

- Cache decorator
- On cache miss, get()s block until someone writes the key to the cache
- Optional timeout

```java
CacheManager manager = new CacheManager();
Ehcache cache = manager.getEhcache("items");
manager.replaceCacheWithDecoratedCache(
    cache,
    new BlockingCache(cache));
```
SelfPopulatingCache

- Cache decorator, extends BlockingCache
- get() of unknown key will construct the entry using a supplied factory
- “Read through” caching

```java
CacheManager manager = new CacheManager();
Ehcache cache = manager.getEhcache("items");
manager.replaceCacheWithDecoratedCache(
     cache,
     new SelfPopulatingCache(cache, cacheEntryFactory));
```
JTA

Ehcache 2.0

• Cache acts as XAResource
• Works with any JTA Transaction Manager
• Autodetects JBossTM, Bitronix, Atomikos
• Transactionally move data between database, cache, queue, etc
Pain of Duplication

- How do I get failover capability while avoiding excessive duplication of data?
Partitioning + Terracotta Virtual Memory

- Each node (mostly) holds data it has seen
- Use load balancer to get app-level partitioning
- Use fine-grained locking to get concurrency
- Use memory flush/fault to handle memory overflow and availability
- Use causal ordering to guarantee coherency
Pain of Ignorance

• Is my cache being used? How much?
• Is caching improving latency?
• How much memory is my cache using?
Dev Console

- Ehcache console
  - See caches, configuration, statistics
  - Dynamic configuration changes
- Hibernate console
  - Hibernate-specific view of caches
  - Hibernate stats
Scaling Your Cache
Scalability Continuum

<table>
<thead>
<tr>
<th>causal ordering</th>
<th>YES</th>
<th>NO</th>
<th>YES</th>
<th>YES</th>
</tr>
</thead>
<tbody>
<tr>
<td># JVMs</td>
<td>JVM</td>
<td>JVM</td>
<td>JVM</td>
<td>JVM</td>
</tr>
<tr>
<td>runtime</td>
<td>Ehcache</td>
<td>Ehcache RMI</td>
<td>Ehcache disk store</td>
<td>Terracotta OSS</td>
</tr>
<tr>
<td></td>
<td>2 or more JVMs</td>
<td>2 or more big JVMs</td>
<td>2 or more JVMs</td>
<td>lots of JVMs</td>
</tr>
<tr>
<td></td>
<td>Ehcache DX management and control</td>
<td>Ehcache EX and FX management and control</td>
<td>more scale</td>
<td></td>
</tr>
</tbody>
</table>
Thanks!

- Twitter - @puredanger
- Blog - http://tech.puredanger.com
- Terracotta - http://terracotta.org
- Ehcache - http://ehcache.org