# Scaling Your Cache

TERRACOTTA

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

# Facts of Life

# Caching to the rescue!

# Temporal Locality

# Temporal Locality

# Non-uniform distribution

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

# A brief aside...

- What is Ehcache?
- What is Terracotta?

# Ehcache Example

```
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

App Node
App Node
App Node
App Node
Terracotta Server
Terracotta Server
App Node
App Node
App Node
App Node

# 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:

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

# Spill to Disk in Ehcache

Spill to disk:

```
    <diskStore path="java.io.tmpdir"/>

    <cache   name="employees"
             maxElementsInMemory="1000"
             memoryStoreEvictionPolicy="LRU"
             eternal="false"
             timeToIdleSeconds="600"
             timeToLiveSeconds="3600"

             overflowToDisk="true"
             maxElementsOnDisk="1000000"
             diskExpiryThreadIntervalSeconds="120"
             diskSpoolBufferSizeMB="30" />
```

# Terracotta Clustering

Terracotta configuration:

```
<terracottaConfig url="server1:9510,server2:9510"/>

<cache  name="employees"
        maxElementsInMemory="1000"
        memoryStoreEvictionPolicy="LRU"
        eternal="false"
        timeToIdleSeconds="600"
        timeToLiveSeconds="3600"
        overflowToDisk="false">

    <terracotta/>
</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

# TTI and TTL in Ehcache

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

# Replication in Ehcache

```
<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

```
<diskStore path="java.io.tmpdir"/>

<cache  name="employees"
        maxElementsInMemory="1000"
        memoryStoreEvictionPolicy="LRU"
        eternal="false"
        timeToIdleSeconds="600"
        timeToLiveSeconds="3600"
        overflowToDisk="true"
        maxElementsOnDisk="1000000"
        diskExpiryThreadIntervalSeconds="120"
        diskSpoolBufferSizeMB="30"

        diskPersistent="true" />
```

# Bootstrap Cache Loader

Bootstrap a new cache node from a peer:

```
<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.getEhcache("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

```
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

```
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

# Thanks!

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