Big Data
WITHOUT A BIG DATABASE

Kate Matsudaira
popforms
@katemats
## Two Kinds of Data

<table>
<thead>
<tr>
<th>nicknames</th>
<th>“user”, “transactional”</th>
<th>“reference”, “non-transactional”</th>
</tr>
</thead>
<tbody>
<tr>
<td>examples:</td>
<td>• user accounts</td>
<td>• product/offer catalogs</td>
</tr>
<tr>
<td></td>
<td>• shopping cart/orders</td>
<td>• service catalogs</td>
</tr>
<tr>
<td></td>
<td>• user messages</td>
<td>• static geolocation data</td>
</tr>
<tr>
<td></td>
<td>• …</td>
<td>• dictionaries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• …</td>
</tr>
<tr>
<td>created/modified by:</td>
<td>users</td>
<td>business (you)</td>
</tr>
<tr>
<td>sensitivity to staleness:</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>plan for growth:</td>
<td>hard</td>
<td>easy</td>
</tr>
<tr>
<td>access optimization:</td>
<td>read/write</td>
<td>mostly read</td>
</tr>
</tbody>
</table>
## Two Kinds of Data

<table>
<thead>
<tr>
<th>nicknames</th>
<th>“user”, “transactional”</th>
<th>“reference”, “non-transactional”</th>
</tr>
</thead>
<tbody>
<tr>
<td>examples:</td>
<td></td>
<td>• product/offer catalogs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• service catalogs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• static geolocation data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• dictionaries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ...</td>
</tr>
<tr>
<td>created/modified by:</td>
<td>users</td>
<td>business (you)</td>
</tr>
<tr>
<td>sensitivity to staleness:</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>plan for growth:</td>
<td>hard</td>
<td>easy</td>
</tr>
<tr>
<td>access optimization:</td>
<td>read/write</td>
<td>mostly read</td>
</tr>
</tbody>
</table>
Portland Hotels

Wed, 7/18 - Fri, 7/20  ·  1 adult · 1 room  ·  Change search

Rate  ·  ?Under $100  ·  $101 - $200  ·  $201 - $300  ·  $301+  ·  More filters
Class  ·  1 star · 2 star · 3 star · 4 star · 5 star

1 - 25 of 182 RESULTS
Sort by · Popularity  ·  Class  ·  Distance  ·  Price

1. Hotel Deluxe
   (503) 219-2094 · 4 star · 1.02 mi · 5 stars  ·  $279
   [Photos · Amenities · Details · Website · 31 reviews]
   Book with: $279 www.hoteldeluxe... · Get rate Expedia

2. Hotel Lucia
   (503) 225-1717 · 4 star · 0.39 mi · 5 stars  ·  Call for info
   [Photos · Amenities · Details · Website · 44 reviews]

3. The Heathman Hotel
   (503) 241-4100 · 4 star · 0.87 mi · 5 stars  ·  $259
   [Booking for $259 Orbitz · $259 CheapTickets · More]
   [Book with: $259 Kayak · $259 Orbitz · $259 CheapTickets · More]
   [Website · 115 reviews]

4. Governor Hotel
   (503) 224-3400 · 4 star · 0.69 mi · 4 stars  ·  $238
   [Average rate]
   [Book with: $238 Kayak · $238 Orbitz · $238 CheapTickets · More]
   [Website · 24 reviews]

5. The Nines, Portland
   (503) 222-9996 · 5 star · 0.56 mi · 5 stars  ·  Call for info
   [Website · 32 reviews]

6. Hilton Vancouver Washington
   (360) 993-4500 · 3 star · 11.37 mi · 4 stars  ·  $152
   [Booking for $152 Kayak · $152 Hotels.com · $152 Getaroom · More]
   [Website · 16 reviews]

7. Hotel Vintage Plaza, a Kimpton Hotel
   (503) 228-1212 · 4 star · 0.41 mi · 4 stars  ·  $329
   [Website · 37 reviews]
### Portland Hotels

**Portland, OR**

**Wed, 7/18 - Fri, 7/20**
1 adult · 1 room

<table>
<thead>
<tr>
<th>Rate Range</th>
<th>Number of Results</th>
<th>More Filters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Under $100</td>
<td>20可能会已更新</td>
<td></td>
</tr>
<tr>
<td>$101 - $200</td>
<td>20可能会已更新</td>
<td></td>
</tr>
<tr>
<td>$201 - $300</td>
<td>20可能会已更新</td>
<td></td>
</tr>
<tr>
<td>$301+</td>
<td>15可能会已更新</td>
<td></td>
</tr>
<tr>
<td>1 star</td>
<td>5可能会已更新</td>
<td></td>
</tr>
<tr>
<td>2 star</td>
<td>10可能会已更新</td>
<td></td>
</tr>
<tr>
<td>3 star</td>
<td>10可能会已更新</td>
<td></td>
</tr>
<tr>
<td>4 star</td>
<td>5可能会已更新</td>
<td></td>
</tr>
<tr>
<td>5 star</td>
<td>2可能会已更新</td>
<td></td>
</tr>
</tbody>
</table>

### Portland Hotel List

<table>
<thead>
<tr>
<th>Hotel Name</th>
<th>Rating</th>
<th>Distance</th>
<th>Price</th>
<th>Book with:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hotel Deluxe</strong></td>
<td>4 star</td>
<td>1.02 mi</td>
<td>$279</td>
<td>[<a href="http://www.hotelde">www.hotelde</a> Luxe.com](<a href="http://www.hotelde">http://www.hotelde</a> Luxe.com)</td>
</tr>
<tr>
<td><strong>Hotel Lucia</strong></td>
<td>4 star</td>
<td>0.39 mi</td>
<td>$235</td>
<td>Call for info <a href="https://www.expedia.com">Expedia</a></td>
</tr>
<tr>
<td><strong>The Heathman Hotel</strong></td>
<td>4 star</td>
<td>0.69 mi</td>
<td>$238</td>
<td><a href="https://www.orbitz.com">Orbitz</a></td>
</tr>
<tr>
<td><strong>Governor Hotel</strong></td>
<td>4 star</td>
<td>0.69 mi</td>
<td>$238</td>
<td><a href="https://www.kayak.com">Kayak</a></td>
</tr>
<tr>
<td><strong>The Nines, Portland</strong></td>
<td>3 star</td>
<td>1.31 mi</td>
<td>$329</td>
<td><a href="https://www.hotwire.com">Hotwire</a></td>
</tr>
<tr>
<td><strong>Hilton Vancouver Washington</strong></td>
<td>3 star</td>
<td>1.17 mi</td>
<td>$238</td>
<td><a href="https://www.cheaptickets.com">CheapTickets</a></td>
</tr>
</tbody>
</table>

---

**Reference data**
# Performance Reminder

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>main memory read</td>
<td>0.0001 ms (100 ns)</td>
</tr>
<tr>
<td>network round trip</td>
<td>0.5 ms (500,000 ns)</td>
</tr>
<tr>
<td>disk seek</td>
<td>10 ms (10,000,000 ns)</td>
</tr>
</tbody>
</table>

THE BEGINNING
THE BEGINNING

availability problems

load balancer

webapp

webapp

data loader

BIG DATABASE

service

service

service
THE BEGINNING

availability problems

performance problems

load balancer

BIG DATABASE

data loader

webapp

service

service

service
REPLICATION

load balancer

webapp

webapp

load balancer

service

service

service

BIG DATABASE

data loader
REPLICATION
REPLICATION

load balancer

webapp

webapp

load balancer

service

service

service

BIG DATABASE

data loader

scalability problems
REPLICATION

load balancer

webapp

load balancer

service

service

service

BIG DATABASE

data loader

REPLICA

scalability problems

performance problems
Replication

- Webapp
- Load balancer
- Service
- Big database
- Data loader

Scalability problems
Performance problems
Operational overhead
LOCAL CACHING

load balancer
  ▶
  ▶
  ▶
  ▶
  webapp
  webapp
  service
  service
  service

BIG DATABASE

REPLICA

data loader
LOCAL CACHING
LOCAL CACHING

load balancer
webapp
webapp
load balancer
service
service
service
cache
cache
cache
BIG DATABASE
REPLICA
data loader

scalability problems
LOCAL CACHING

load balancer

webapp

load balancer

service cache

load balancer

service cache

load balancer

service cache

BIG DATABASE

REPLICA

data loader

scalability problems

operational overhead

Local Caching
LOCAL CACHING

- REPLICA
- webapp
- load balancer
- service
- cache
- BIG DATABASE
- service
- cache
- data loader
- consistency problems
- scalability problems
- operational overhead
- performance problems
LOCAL CACHING

- Scalability problems
- Performance problems
- Operational overhead

- Long tail performance problems
- Consistency problems

Load balancer → Webapp → Service → Cache → Big Database

Data loader → Replica

Database: BIG DATABASE

Cache: Replica
The Long Tail Problem

80% of requests query 10% of entries (head)

20% of requests query remaining 90% of entries (tail)
BIG CACHE

Load balancer

webapp

Load balancer

service

service

service

BIG CACHE

Database

replica

data loader

Preload
BIG CACHE
BIG CACHE

- load balancer
  - webapp
  - webapp

- service
- service
- service

- BIG CACHE

- replica
- database
  - data loader
  - preload

- operational overhead
- performance problems
- scalability problems
BIG CACHE

long tail performance problems

operational overhead

performance problems

scalability problems

load balancer

webapp

service

service

service

BIG CACHE

data loader

preload

replica

database

consistency problems
memcached(b)
BIG CACHE TECHNOLOGIES

memcached(b)

ElastiCache (AWS)

Do I look like I need a cache?
BIG CACHE TECHNOLOGIES

memcached(b)

ElastiCache (AWS)

Do I look like I need a cache?

Oracle Coherence
New
Ikea Kitteh Storage Solutions
New
Ikea Kitteh Storage Solutions

Targeted generic data/use cases.
Dynamically assign keys to the “nodes”

Targeted generic data/use cases.

New
Ikea Kitteh Storage Solutions
Dynamically assign keys to the “nodes”

Targeted generic data/use cases.

Scales horizontally

New Ikea Kitteh Storage Solutions
Targeted generic data/use cases.

Dynamically assign keys to the “nodes”

Scales horizontally

Dynamically rebalances data

New Ikea Kitteh Storage Solutions
New
Ikea Kitteh Storage Solutions

- Dynamically assign keys to the "nodes"
- Targeted generic data/use cases.
- Dynamically rebalances data
- Poor performance on cold starts
- Scales horizontally
Targeted generic data/use cases.

Dynamically rebalances data

Scales horizontally

Dynamically assign keys to the “nodes”

No assumptions about loading/updating data

Poor performance on cold starts
BIG CACHE TECHNOLOGIES

- Additional hardware
- Additional configuration
- Additional monitoring
- Extra network hop
- Slow scanning
- Additional deserialization
BIG CACHE TECHNOLOGIES

• Additional hardware
• Additional configuration
• Additional monitoring

→ operational overhead

• Extra network hop
• Slow scanning
• Additional deserialization
BIG CACHE TECHNOLOGIES

- Additional hardware
- Additional configuration
- Additional monitoring

- Extra network hop
- Slow scanning
- Additional deserialization
NOSQL TO THE RESCUE?
The diagram illustrates a NoSQL database setup with a webapp and services. The text 'some performance problems' suggests that there are issues that need to be addressed. The diagram is titled 'NOSQL TO THE RESCUE?'
NOSQL TO THE RESCUE?

The diagram shows a web application (webapp) and a load balancer connected to a service. There is also a NoSQL database with a replica and a data loader. The diagram indicates some performance problems and some scalability problems.
NoSQL to the Rescue?

The Rescue?

- Some performance problems
- Some scalability problems
- Some operational overhead

NoSQL Replica

NoSQL Database

Data loader

Service

Webapp

Load balancer
REMOTE STORE RETRIEVAL LATENCY

network | remote store | network | client
Remote Store Retrieval Latency

TCP request: 0.5 ms

Lookup/write response: 0.5 ms

TCP response: 0.5 ms

read/parse response: 0.25 ms
Remote Store Retrieval Latency

TCP request: 0.5 ms

Lookup/write response: 0.5 ms

TCP response: 0.5 ms

read/parse response: 0.25 ms

Total time to retrieve single value: 1.75 ms
TOTAL TIME TO RETRIEVE A SINGLE VALUE

from remote store: 1.75 ms
from memory: 0.001 ms
(10 main memory reads)
TOTAL TIME TO RETRIEVE A SINGLE VALUE

from remote store: 1.75 ms
from memory: 0.001 ms
(10 main memory reads)

SEQUENTIAL ACCESS OF 1 MILLION RANDOM KEYS

from remote store: 30 minutes
from memory: 1 second
THE TRUTH ABOUT DATABASES
“What I'm going to call as the *hot data cliff*: As the size of your hot data set (data frequently read at sustained rates above disk I/O capacity) approaches available memory, write operation bursts that exceed disk write I/O capacity can create a *trashing death spiral* where hot disk pages that MongoDB desperately needs are evicted from disk cache by the OS as it consumes more buffer space to hold the writes in memory.”

“Redis is an in-memory but persistent on disk database, so it represents a different trade off where very high write and read speed is achieved with the limitation of data sets that can't be larger than memory.”
They are fast if everything fits into memory.
Ur computr needs
More rams
CAN YOU KEEP IT IN MEMORY YOURSELF?

- webapp
- load balancer
- service
- full cache
- data loader
- service
- full cache
- data loader
- service
- full cache
- data loader

BIG DATABASE
Can you keep it in memory yourself?
CAN YOU KEEP IT IN MEMORY YOURSELF?

load balancer

webapp

webapp

service

full cache
data loader

service

full cache
data loader

service

full cache
data loader

operational relief

DATABASE

scales infinitely
Can you keep it in memory yourself?
CAN YOU KEEP IT IN MEMORY YOURSELF?

webapp
load balancer
webapp
load balancer
service full cache data loader
service full cache data loader
service full cache data loader
operational relief
BIG DATABASE
scales infinitely

consistency problems
performance gain
**Fixing Consistency**

Diagram showing a load balancer connecting to multiple webapp services, each with a full cache and data loader.
Fixing Consistency

1. Deployment “Cells”
2. Sticky user sessions

Deployment Cell

- Load balancer
- Webapp
- Service
- Full cache
- Data loader
HOW DO YOU FIT ALL OF THAT DATA INTO MEMORY?
"Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered.

We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.

Donald Knuth
HOW DO YOU FIT ALL THAT DATA IN MEMORY?
THE ANSWER
“Domain Layer (or Model Layer):

Responsible for representing concepts of the business, information about the business situation, and business rules. State that reflects the business situation is controlled and used here, even though the technical details of storing it are delegated to the infrastructure. This layer is the heart of business software.”

Eric Evans, Domain-Driven Design, 2003
DOMAIN MODEL DESIGN GUIDELINES

#1 Keep it immutable
# Domain Model Design Guidelines

1. Keep it immutable
2. Use independent hierarchies

http://alloveralbany.com/images/bumper_gawking_dbgeek.jpg
# Domain Model Design Guidelines

1. Keep it immutable
2. Use independent hierarchies
3. Optimize Data

Help! I am in the trunk!

http://alloveralbany.com/images/bumper_gawking_dbgeek.jpg
intern() YOUR IMMUTABLES
private final Map<Class<?>, Map<Object, WeakReference<Object>>> cache =
    new ConcurrentHashMap<Class<?>, Map<Object, WeakReference<Object>>>();

public <T> T intern(T o) {
    if (o == null)
        return null;
    Class<?> c = o.getClass();
    Map<Object, WeakReference<Object>> m = cache.get(c);
    if (m == null)
        cache.put(c, m = synchronizedMap(new WeakHashMap<Object, WeakReference<Object>>()));
    WeakReference<Object> r = m.get(o);
    @SuppressWarnings("unchecked")
    T v = (r == null) ? null : (T) r.get();
    if (v == null) {
        v = o;
        m.put(v, new WeakReference<Object>(v));
    }
    return v;
}
USE INDEPENDENT HIERARCHIES

Product
  id = ...
  title = ...

Offers
Specifications
Description
Reviews
Rumors
Model History

Product Summary
  productId = ...

Offers
  productId = ...

Specifications
  productId = ...

Reviews
  productId = ...

Model History
  productId = ...

Description
  productId = ...

Rumors
  productId = ...

Product Info
COLLECTION
OPTIMIZATION
Leverage PRIMITIVE KEYS/VALUES

<table>
<thead>
<tr>
<th>Collection with 10,000 elements [0 .. 9,999]</th>
<th>Size in memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.util.ArrayList&lt;Integer&gt;</td>
<td>200K</td>
</tr>
<tr>
<td>java.util.HashSet&lt;Integer&gt;</td>
<td>546K</td>
</tr>
<tr>
<td>gnu.trove.list.array.TIntArrayList</td>
<td>40K</td>
</tr>
<tr>
<td>gnu.trove.set.hash.TIntHashSet</td>
<td>102K</td>
</tr>
</tbody>
</table>

[Trove](“High Performance Collections for Java”)
Collections with small number of entries (up to ~20):

class ImmutableMap<K, V> implements Map<K, V>, Serializable {
  ...
}

class MapN<K, V> extends ImmutableMap<K, V> {
  final K k1, k2, ..., kN;
  final V v1, v2, ..., vN;
  @Override public boolean containsKey(Object key) {
    if (eq(key, k1)) return true;
    if (eq(key, k2)) return true;
    ...
    return false;
  }
  ...
}
SPACE SAVINGS

java.util.HashMap:
128 bytes + 32 bytes per entry

compact immutable map:
24 bytes + 8 bytes per entry
Numeric Data Optimization
Prices will hold steady 77% Confidence

<table>
<thead>
<tr>
<th>Current low</th>
<th>Price History</th>
<th>Lowest</th>
<th>Average</th>
<th>Highest</th>
<th>Volatility</th>
</tr>
</thead>
<tbody>
<tr>
<td>$327</td>
<td>164 days</td>
<td>$327</td>
<td>$357</td>
<td>$400</td>
<td>Low</td>
</tr>
</tbody>
</table>
Example: Price History

Problem:
- Store daily prices for 1M products, 2 offers per product
- Average price history length per product ~2 years

Total price points:
\[(1M + 2M) \times 730 = \sim 2 \text{ billion}\]
PRICE HISTORY
FIRST ATTEMPT

TreeMap<Date, Double>

88 bytes per entry * 2 billion = ~180 GB
Typical Shopping Price History
RUN LENGTH ENCODING

6 a 3 b 6 c
Price History Optimization

- **positive:** price (adjusted to scale)
- **negative:** run length (precedes price)
- **zero:** unavailable

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-20</td>
<td>100</td>
<td>-40</td>
<td>150</td>
<td>-10</td>
<td>140</td>
<td>-20</td>
<td>100</td>
<td>-10</td>
<td>0</td>
<td>-20</td>
<td>100</td>
<td>90</td>
</tr>
<tr>
<td>-9</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Drop pennies
- Store prices in primitive short (use scale factor to represent prices greater than `Short.MAX_VALUE`)

**Memory:**

\[
15 \times 2 + 16 \text{ (array)} + 24 \text{ (start date)} + 4 \text{ (scale factor)} = 74 \text{ bytes}
\]
Space Savings

Reduction compared to TreeMap<Date, Double>:

155 times

Estimated memory for 2 billion price points:

1.2 GB
Space Savings

Reduction compared to TreeMap<Date, Double>:

155 times

Estimated memory for 2 billion price points:

1.2 GB  \ll  180 GB
public class PriceHistory {

    private final Date startDate; // or use org.joda.time.LocalDate
    private final short[] encoded;
    private final int scaleFactor;

    public PriceHistory(SortedMap<Date, Double> prices) { ... } // encode
    public SortedMap<Date, Double> getPricesByDate() { ... } // decode
    public Date getStartDate() { return startDate; }

    // Below computations implemented directly against encoded data
    public Date getEndDate() { ... }
    public Double getMinPrice() { ... }
    public int getNumChanges(double minChangeAmt, double minChangePct, boolean abs) { ... }
    public PriceHistory trim(Date startDate, Date endDate) { ... }
    public PriceHistory interpolate() { ... }
}
KNOW YOUR DATA
COMPRESS TEXT
String Compression:

**BYTE ARRAYS**

- Use the minimum character set encoding

```java
static Charset UTF8 = Charset.forName("UTF-8");

String s = "The quick brown fox jumps over the lazy dog"; // 42 chars, 136 bytes
byte[] b = s.getBytes(UTF8); // 64 bytes
String s1 = "Hello"; // 5 chars, 64 bytes
byte[] b1 = s1.getBytes(UTF8); // 24 bytes

byte[] toBytes(String s) { return s == null ? null : s.getBytes(UTF8); }
String toString(byte[] b) { return b == null ? null : new String(b, UTF8); }
```
STRING COMPRESSION: SHARED PREFIX

- Great for URLs

```java
public class PrefixedString {
    private PrefixedString prefix;
    private byte[] suffix;

    ...

    @Override public int hashCode() { ... }
    @Override public boolean equals(Object o) { ... }
}
```
public abstract class AlphaNumericString {
    public static AlphaNumericString make(String s) {
        try {
            return new Numeric(Long.parseLong(s, Character.MAX_RADIX));
        } catch (NumberFormatException e) {
            return new Alpha(s.getBytes(UTF8));
        }
    }

    protected abstract String value();
    @Override public String toString() { return value(); }

    private static class Numeric extends AlphaNumericString {
        long value;
        Numeric(long value) { this.value = value; }
        @Override protected String value() { return Long.toString(value, Character.MAX_RADIX); }
        @Override public int hashCode() { ... }
        @Override public boolean equals(Object o) { ... }
    }

    private static class Alpha extends AlphaNumericString {
        byte[] value;
        Alpha(byte[] value) {this.value = value; }
        @Override protected String value() { return new String(value, UTF8); }
        @Override public int hashCode() { ... }
        @Override public boolean equals(Object o) { ... }
    }
}
STRING COMPRESSION: LARGE STRINGS
String Compression: Large Strings

Gzip

Become the master of your strings!
String Compression: Large Strings

Become the master of your strings!

bzip2

Gzip

String Compression: Large Strings

Gzip

bzip2

Just convert to byte[] first, then compress

Become the master of your strings!
JVM Tuning
make sure to use compressed pointers
(-XX:+UseCompressedOops)
make sure to use compressed pointers
(-XX:+UseCompressedOops)

use low pause GC
(Concurrent Mark Sweep, G1)

This s#!% is heavy!
JVM Tuning

- Make sure to use compressed pointers (-XX:+UseCompressedOops)
- Use low pause GC (Concurrent Mark Sweep, G1)
- Over provision heap by ~30%
- Adjust generation sizes/ratios
- This s#!% is heavy!
JVM Tuning

These books are here for an essential structural purpose. They are not for sale.
JVM Tuning

These books are here for an essential structural purpose. They are not for sale.
JVM Tuning

Print garbage collection

These books are here for an essential structural purpose. They are not for sale.
JVM Tuning

Print garbage collection

If GC pauses still prohibitive then consider partitioning
IN SUMMARY
KNOW YOUR BUSINESS
KNOW YOUR DATA
Know when to optimize.
My company: https://popforms.com
My website: http://katemats.com

And much of the credit for this talk goes to Leon Stein for developing the technology. Thank you, Leon.
How do you load the data?

FAIL

PEPSI®
Cache Loading
Final datasets should be compressed and stored (i.e. S3)

Cache loading tips & tricks
Final datasets should be compressed and stored (i.e. S3)

Keep the format simple (CSV, JSON)

Help! I am in the trunk!

Cache loading tips & tricks
Cache loading tips & tricks

Final datasets should be compressed and stored (i.e. S3)

Keep the format simple (CSV, JSON)

Poll for updates

Poll frequency == data inconsistency threshold

Help! I am in the trunk!
CACHE LOADING TIME SENSITIVITY
CACHE LOADING:
LOW TIME SENSITIVITY DATA

/tax-rates

/date=2012-05-01

tax-rates.2012-05-01.csv.gz

/date=2012-06-01

tax-rates.2012-06-01.csv.gz

/date=2012-07-01

tax-rates.2012-07-01.csv.gz
Cache Loading: Medium/High Time Sensitivity

/prices

/full

/date=2012-07-01

price-obs.2012-07-01.csv.gz

/date=2012-07-02

/inc

/date=2012-07-01

2012-07-01T00-10-00.csv.gz

2012-07-01T00-20-00.csv.gz
CACHE LOADING STRATEGY SWAP
Cache is immutable, so no locking is required.
Cache is immutable, so no locking is required.

Works well for infrequently updated data sets.

And for datasets that need to be refreshed each update.
CACHE LOADING STRATEGY: CRUD
Cache Loading Strategy: CRUD

Deletions can be tricky

YARRRRRR!
Cache Loading Strategy: CRUD

Avoid full synchronization

Deletions can be tricky

YARRRRRRR!
Cache Loading Strategy: CRUD

Deletions can be tricky

Avoid full synchronization

Consider loading cache in small batches. Use one container per partition

YARRRRRR!
public class LongCache<V> {
    private TLongObjectMap<V> map = new TLongObjectHashMap<V>();
    private ReentrantReadWriteLock lock = new ReentrantReadWriteLock();
    private Lock r = lock.readLock(), w = lock.writeLock();
    public V get(long k) {
        r.lock();
        try { return map.get(k); } finally { r.unlock(); }
    }
    public V update(long k, V v) {
        w.lock();
        try { return map.put(k, v); } finally { w.unlock(); }
    }
    public V remove(long k) {
        w.lock();
        try { return map.remove(k); } finally { w.unlock(); }
    }
}
CACHE LOADING OPTIMIZATIONS
Cache loading optimizations

Periodically generate serialized data/state

I am “cooking” the data sets. Ha!

Keep local copies
Cache loading optimizations

I am “cooking” the data sets. Ha!

Periodically generate serialized data/state

Validate with CRC or hash

Keep local copies
service instance

- product summary
- offers
- predictions
- matching
- dependencies

load balancer

health check

service status aggregator (servlet)
DEPENDENT CACHES

Service instance

- Product summary
- Offers
- Predictions
- Matching
- Dependencies

Service status aggregator (servlet)

Load balancer

Health check
service instance

- service status aggregator (servlet)
- offers
- predictions
- matching
- dependencies
- product summary

load balancer

health check

DEPENDENT CACHES
DEPENDENT CACHES

service instance

- product summary
- matching
- dependencies
- offers
- predictions
- service status aggregator (servlet)

load balancer

health check
DEPLOYMENT CELL STATUS

load balancer
health check

deployment cell

cell status aggregator

webapp status aggregator
service 1 status aggregator
service 2 status aggregator

HTTP or JMX
Hierarchical Status Aggregation