GigaSpaces Technologies

Real Time Analytics for Big Data
Lessons from Twitter..
The Real Time Boom..

Facebook Real Time Social Analytics

SaaS Real Time User Tracking

Google Real Time Web Analytics

Twitter paid tweet analytics

New Real Time Analytics Startups..

Google Real Time Search
Analytics @ Twitter

Counting
- How many request/day?
- What’s the average latency?
- How many signups, sms, tweets?

Correlating
- Desktop vs Mobile user?
- What devices fail at the same time?
- What features get user hooked?

Research
- What features get re-tweeted
- Duplicate detection
- Sentiment analysis
Note the Time dimension

- Real time (msec/sec)
- Near real time (Min/Hours)
- Batch (Days..)
The data resolution & processing models

**Counting**
- Mostly Event Driven
- High resolution – every tweet counts

**Correlating**
- Ad-hoc queries
- Mid resolution - Aggregated counters

**Research**
- Pre generated reports
- Cross grain resolution – trends,..
Twitter Real-time Analytics System
Twitter by the numbers

- It takes 1 week for users to send a billion Tweets.
- The average number of Tweets people sent per day: 140 million.
- Tweets sent on March 11, 2011: 177 million.
- Current TPS record: 6,939.
- Average number of new accounts per day over the last month: 460,000.
- 5% of twitter users create 75% of the content.
Reach is the number of unique people exposed to a URL on Twitter
Computing Reach

- Tweets
- Followers
- Distinct Followers

Count
Reach

© Copyright 2011 Gigaspaces Ltd. All Rights Reserved
Challenge 1: Collect twitter feeds

• Collect feeds from a @<twitter id>
  – Write scalability 10k tweets/sec
  – Reliability – no message loss
  – Message size: 140 char
  – Latency – x msec
  – Store at least an hour
Challenge 2: Parse tweets

• Parse every tweets into word/count token
  – Which technology to use
    • Database
    • Hadoop, Batch
    • Event processing

  – Models for processing reliability
    • Ensure once and only once processing
    • Replay, handling replay of workflow
  – Message ordering
  – Message locality
  – Avoiding backlog
Challenge 3: Global indexing

• Collect the word/count into global index
  – Scalability
  – Consistency
  – Backlog
Challenge: 4 – Storing the data

- Sizing? Yearly storage
- Performance?
- Compression?
Challenge 5: Query the data

- Collecting specific word count
- Word count trend
- Collecting word count per user/region
- Collecting real-time stream
- Monthly/Yearly trend analysis
Challenge 6: Managing the system

- Deploy the cluster in a cloud
- Elasticity – increase instances based on load without breaking the system and without manual intervention
- Design the system for continues development
- Monitoring – provide consistent monitoring for all the various parts
- Trouble shooting – through Log analysis
Real-time Analytics System With GigaSpaces
Instead of treating memory as a cache, why not treat it as a primary data store?

- Facebook keeps 80% of its data in Memory (Stanford research)

- RAM is 100-1000x faster than Disk (Random seek)
  - Disk - 5 -10ms
  - RAM – x0.001msec
ONE Data any API’s:

The right API for the JOB

- **Document**
  - Storing tweets

- **Key/Value**
  - Atomic-counters

- **JPA**
  - Complex query

- **Executors**
  - Real Time Map/Reduce
  - Aggregated join query
Availability

• Backup node per partition
• Synchronous replication to ensure consistency and no data loss
• On-demand backup to minimizing over provisioning overhead (cost, performance, capacity)
• Partition incoming tweets based on tweet id

• Partition word/count index based on word hash key – all updates to the same index are routed to the same partition.
Global index update - optimization

• Use batch write for updating the word/count index
• Use atomic update to ensure consistency with minimum performance overhead on concurrent updates
Processing the data

• Use event handlers to process the data as it's coming.
• Use shared state to control the flow (order)
• Use FIFO to ensure order
• Use local TX to recover from failure

1) Parse
2) Update Global index
Collocate

• Group event handler and the data into processing-units
  – Minimize latency
  – Simple scaling (less moving parts)
  – Better reliability

1) Parse
2) Update Global index
BigData Database for long term data

- **Write-behind**
  - Batch update to the DB to minimize disk performance and latency overhead
  - Logs are backed up to avoid data loss between batches
- **Plug-in to any DB**
  - Use plug-in approach to enable flexibility for choosing the right DB for the JOB
Automation & Cloud enablement

My Recipe
2 Tomcats
10 Data PU’s
10 Cassandra
Recipe DSL
Lifecycle scripts
Custom plug-ins (optional)
Service binaries (optional)

```groovy
application {
    name="simple app"
}

service {
    lifecycle{
        init "mysql_install.groovy"
        start "mysql_start.groovy"
        stop "mysql_stop.groovy"
    }
}
```
Putting it all together

- In Memory Data Grid
- RT Processing Grid
  - Light Event Processing
  - Map-reduce
  - Event driven
  - Execute code with data
  - Transactional

Event Sources

Real Time Map/Reduce

Script

```groovy
script = new StaticScript("println hi; return 0")
```

Query

```java
q = em.createNativeQuery("execute ?");
q.setParameter(1, script);
```

Integer

```java
result = query.getSingleResult();
```
Economic Data Scaling

• Combine memory and disk
  – Memory is \( \times 10, \times 100 \) lower than disk for high data access rate (Stanford research)
  – Disk is lower at cost for high capacity lower access rate.
  – Solution:
    • Memory - short-term data,
    • Disk - long term data
  – Only \(~16G\) required to store the log in memory (500b messages at 10k/h) at a cost of \(~32\) per server.
Economic Scaling

- Automation - reduce operational cost
- Elastic Scaling – reduce over provisioning cost
- Cloud portability (JClouds) – choose the right cloud for the job
- Cloud bursting – scavenge extra capacity when needed
Streaming data processing

Over the next few years we'll see the adoption of scalable frameworks and platforms for handling streaming, or near real-time, analysis and processing. In the same way that Hadoop has been borne out of large-scale web applications, these platforms will be driven by the needs of large-scale location-aware mobile, social and sensor use. – Edd Dumbill O’REILLY
Summary

Big Data Development Made Simple:
Focus on your business logic, Use Big Data platform for dealing scalability, performance, continues availability,..

Its Open: Use Any Stack : Avoid Lockin
Any database (RDBMS or NoSQL); Any Cloud, Use common API’s & Frameworks.

All While Minimizing Cost
Use Memory & Disk for optimum cost/performance.
Built-in Automation and management - Reduces operational costs
Elasticity – reduce over provisioning cost
THANK YOU!

@natishalom
http://blog.gigaspaces.com