Scalability and Horizontal Partitioning

Aleksander M. Stensby
Integrasco A/S
2009-03-19

INTEGRASCO
WORD OF MOUTH

Introduction

- **Who am I?**
  - Aleksander M. Stensby
  - Master of ICT (University of Agder and Carleton University)
  - Senior Software Developer, *Integrasco A/S*
  - *Pattern Recognition, Machine Learning, AI, Scalability, Search, DB, etc.*
  - *Lucene, Solr, Hibernate, Spring, MySql, Java, ...*

- **What's Integrasco?**
  - Word of Mouth on the Internet
  - Trend Analysis
  - *Web Mining, Pattern Recognition, Portal Solutions*

Outline

- Scalability
- Scalability stories
- Horizontal Partitioning – Sharding
- Hibernate Shards

INTEGRASCO
WORD OF MOUTH

Scalability

- Once upon a time....
- ...We scaled databases by buying:
  - Bigger
  - Faster
  - More expensive machines

- Great! ...for big iron profit margins

- ....Not so good for the bank accounts of heroic system builders like us!

- Need to scale well past what we can afford to spend on giant database servers!

INTEGRASCO
WORD OF MOUTH

Scalability

- Scalability is a metric
  - **How many clients can you service?**

- What is scalability?
- A service is said to be scalable if;
  «when we increase the resources in a system it results in increased performance in a manner proportional to resources added!»

- Increasing performance in general means serving more units of work, or to be able to handle larger units of work, such as when datasets grow.

INTEGRASCO
WORD OF MOUTH

Scalability

Three properties of a system:

- **Consistency**
- **Availability**
- **Partitionability** - tolerance to network partitions

You can have at most two of these three properties for any shared-data system!

To scale you have to partition!

Scale up or scale out?

- **Scale vertically** - scale up – The traditional way
- Add resources to a single node in a system
  - Add CPUs or memory to a single computer.

- ...or?

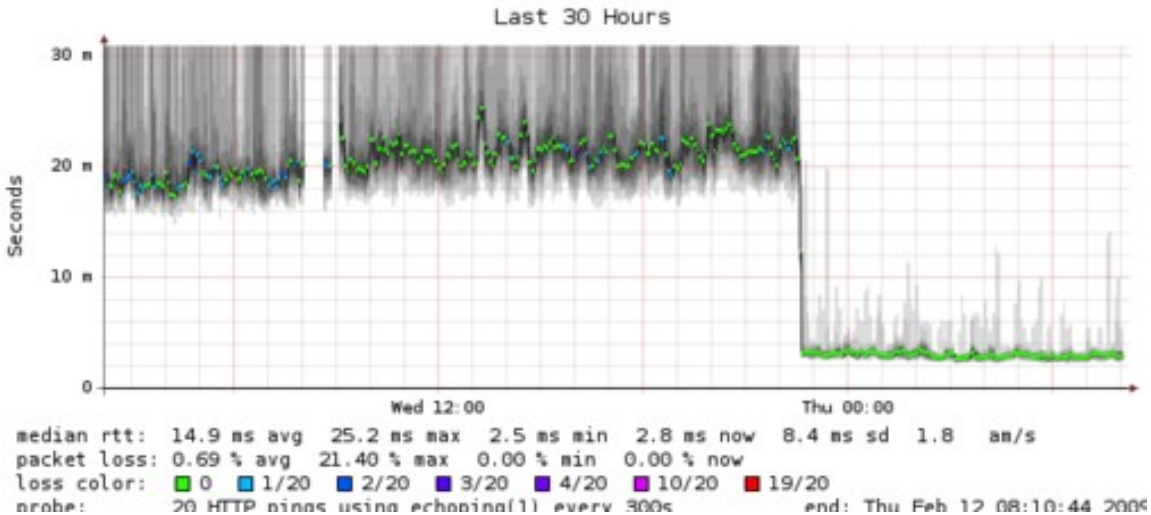
- **Scale horizontally** – Scale out!
- Add more nodes to a system
  - Add a new computer to a distributed software application.

- Motivation? - Moore's law!
- Computer prices drop, performance continues to increase!
  - Use low cost «commodity» systems!
  - 100 small computers in a cluster vs. a billion dollar super computer?

First steps?

- **Replication**
- **Caching**
- **Load-Balancing**

- *Varnish, Squid, Nginx, Apache HTTP Server (extended with mod_proxy), Lighttpd, Perlbal, Pound, Zeus*



INTEGRASCO
WORD OF MOUTH

Scalability Stories

- Google Architecture
- Flickr Architecture

INTEGRASCO
WORD OF MOUTH

Google Architecture

«The King of scalability»

Platform

- Linux
- A large diversity of languages: Python, Java, C++

Stats

In 2005 Google indexed 8 billion web pages...

Estimated 450,000 low-cost commodity servers in 2006

- **Google File System (GFS)** – Core storage platform
- **MapReduce** is a programming model and an associated implementation for processing and generating large data sets.
- **BigTable** is a large scale, fault tolerant, self managing system that includes terabytes of memory and petabytes of storage. It can handle millions of reads/writes per second.

Hardware

- Use ultra cheap commodity hardware and built software on top to handle their death.

INTEGRASCO
WORD OF MOUTH

Flickr Architecture

> *«Flickr now handles more than 1 billion transactions per day, responding in less then a few seconds and can scale linearly at a low cost.»*

Platform

- Linux (RedHat)
- PHP, Perl, Java, Apache

Stats

4 billion queries per day.

Over 400,000 photos added every day

- MySql, Memcached, Shards
- Squid – reverse-proxy for HTML and images.
- Everything (except photos) are stored in the database.
- Scaled at first by replication, but that only helps with reads.
- Use horizontal scaling so they just need to add more machines.
- Each Shard holds 400K+ users data.
- Use dedicated servers for static content.

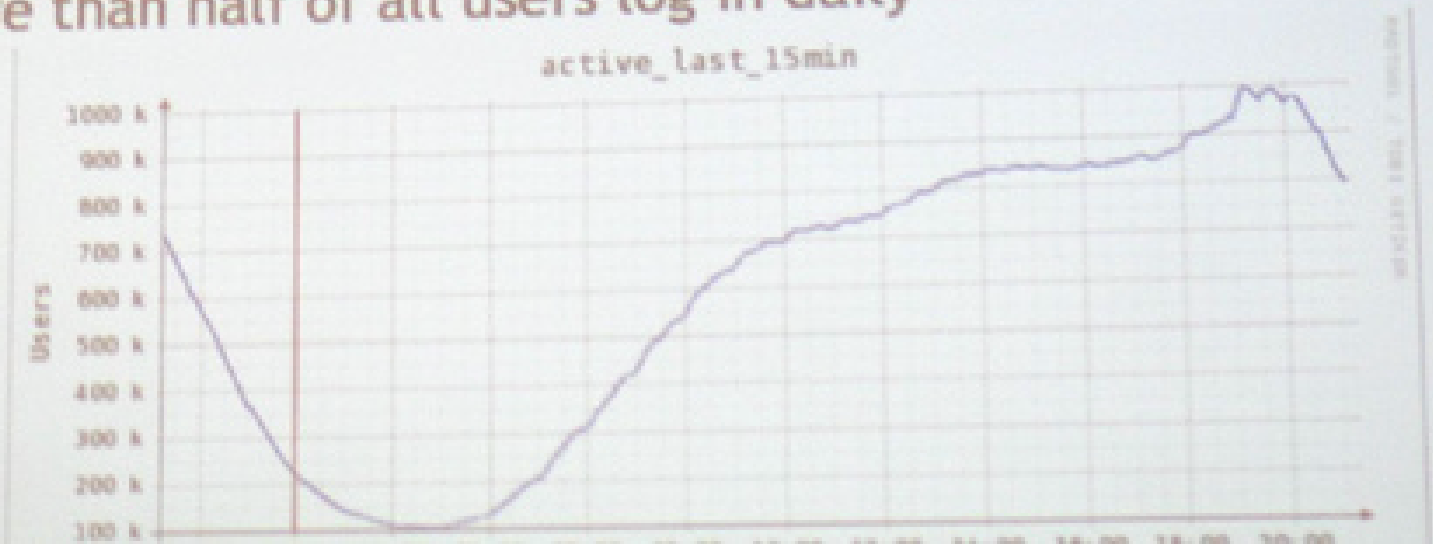
Hardware

- EMT64 w/RHEL4, 16GB RAM,
- 6-disk 15K RPM RAID-10.
- - Data size is at 12 TB of user metadata (these are not photos, this is just innodb ibdata files - the photos are a lot larger).

facebook

Scalability

- Photos - more than 1B (120TB)
- 1M concurrent sessions
- 30B pages monthly (6th most trafficked US site). 1% of all internet time spent on Facebook.
- More than half of all users log in daily



INTEGRASCO
WORD OF MOUTH

Sharding!

«Sharding is a database technique where you break up a big database into many smaller ones.»

«Instead of having 1 million customers on a single, big iron machine, you perhaps have 100,000 customers on 10 different, smaller machines.»

INTEGRASCO
WORD OF MOUTH

Horizontal Partitioning - Sharding

Shard?

- If data doesn't fit on one machine then split it up into pieces
- E.g., too much data to fit in one single relational database
- Each piece is called a shard

Sharding?

- The process of splitting up data.
- For example, putting employees 1-10,000 on shard1 and employees 10,001-20,000 on shard2.

- Structure of the data is identical from server to server.
- The same schema is used across all databases (MySQL, etc).

Advantages

- High availability.
  - If one box goes down the others still operate.
- Faster queries.
  - Smaller amounts of data in each user group mean faster querying.
- More write bandwidth - You can write in parallel!
  - Increases your write throughput.
  - Writing is major bottleneck for many websites.
- You can do more work.
  - A parallel backend - do more work simultaneously.
  - Handle higher user loads, especially when writing data, because there are parallel paths through your system.
  - Load balance web servers, which access shards over different network paths, which are processed by separate CPUs, which use separate caches of RAM and separate disk IO paths to process work - very few bottlenecks limit your work!

HIBERNATE

Hibernate Shards

- Hibernate shards – horizontal data partitioning made easy!
- If your database has a JDBC adapter that means Hibernate can talk to it
- If Hibernate can talk to it that means Hibernate Shards can talk to it!
- ... Makes everything complex?
- When you have too much data, when you fill your database up, you need another solution – most likely to shard your data across multiple relational databases.
- The complexity arises because your application has to have the smarts to access multiple databases....

Hibernate Shards

- If you know Hibernate – Shards is easy!
- The Hibernate API is the same.
- The shard implementation hasn’t violated the API.
- Sharded versions of Session, Critieria, and Factory – no need to change your code!
- Query isn't implemented yet because features like aggregation and grouping are very difficult to implement across databases...
- A sharded session is used to contain Hibernate’s sessions so Hibernate capabilities are preserved.

Sharded Schema Design

- Pick a dimension, a root level entity, that is **easily** sharded.
  - Users and customers are common examples.

- Accept the fact that those entities and all the entities that hang off those entities will be stored in separate physical spaces.

- Querying across different shards will be difficult.
- As will management and just about anything else you take for granted.

- Plan for the future – pick a strategy that will last you a long time!
- Repartitioning/resharding the data is operationally very difficult!!

Sharded Schema Design

- Build simpler models that don't contain as many relationships - you don't have cross shard relationships!
- Your objects graphs should be contained on one shard as much as possible.
- Because the shards design doesn’t modify Hibernate core, you can design using shards from the start, even though you only have one database.
- Then when you need to start scaling it will be easier to grow!

Sharding Strategies

- A Strategy dictates how data are spread across the shards.
  - It's an interface you need to implement.
- There are three Strategies:
  - **Shard Resolution Strategy** - how to retrieve objects
  - **Shard Selection Strategy** – where to save objects
  - **Shard Access Strategy** – how to access the shards (serially, 2 at a time, in parallel, etc)?
- Some out of the box implementations supplied:
  - Round Robin
  - Attribute Based
    - Look at attributes in the data to determine which shard
    - For example: Sharding users by country

Configuration

- Configuration is set by creating a prototype configuration for all shards
- Then you specify what's different from shard to shard like:
  - URL
  - user name and password
  - dialect (MySQL, Postgres, etc).
- Then they'll create a sharded session factory for Hibernate so developers use standard interfaces.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.url">jdbc:mysql://localhost:3306/raw00</property>
    <property name="connection.username">root</property>
    <property name="connection.password"></property>
    <property name="show_sql">true</property>
    <property name="dialect">org.hibernate.dialect.MySQLInnoDBDialect</property>
    <property name="hibernate.connection.shard_id">0</property>
    <property name="hibernate.shard.enable_cross_shard_relationship_checks">true</property>
  </session-factory>
</hibernate-configuration>
```

How to?

```
public SessionFactory createSessionFactory() {
  AnnotationConfiguration prototypeConfig = new AnnotationConfiguration()
  .configure("shardtest0.hibernate.cfg.xml");

  prototypeConfig.addAnnotatedClass(User.class);

  List<ShardConfiguration> shardConfigs = new ArrayList<ShardConfiguration>();
  shardConfigs.add(buildShardConfig(getClass().getResource("shardtest0.hibernate.cfg.xml")));
  shardConfigs.add(buildShardConfig(getClass().getResource("shardtest1.hibernate.cfg.xml")));

  ShardStrategyFactory shardStrategyFactory = buildShardStrategyFactory();

  ShardedConfiguration shardedConfig = new ShardedConfiguration(prototypeConfig,
  shardConfigs, shardStrategyFactory);

  return shardedConfig.buildShardedSessionFactory();
}
```

How to?

```
ShardStrategyFactory buildShardStrategyFactory() {
  return new ShardStrategyFactory() {
    public ShardStrategy newShardStrategy(List<ShardId> shardIds) {
      RoundRobinShardLoadBalancer loadBalancer = new RoundRobinShardLoadBalancer(shardIds);
      ShardSelectionStrategy pss = new RoundRobinShardSelectionStrategy(loadBalancer);
      ShardResolutionStrategy prs = new AllShardsShardResolutionStrategy(shardIds);
      ShardAccessStrategy pas = new SequentialShardAccessStrategy();
      return new ShardStrategyImpl(pss, prs, pas);
    }
  };
}
```

INTEGRASCO
WORD OF MOUTH

How to?

```
SessionFactory sessionFactory = createSessionFactory();

Session session = sessionFactory.openSession();

Criteria criteria = session.createCriteria(User.class);
criteria.add(Restrictions.in("userId", new Integer[]{3, 12, 6028, 5073, 3590}));

List<User> list = criteria.list();
(...)

session.close();
```

Hibernate Shards Limitations

- Full Hibernate HQL is not yet supported.
- Distributed queries are handled by applying a standard HQL query to each shard, merging the results, and applying the filters.
  - This all happens in the application server so using very large data sets could be a problem.
  - It's up to you (the developer) to do the right thing to manage performance.
- No handling of fail over situations, which is just like Hibernate.
  - You could handle it in your connection pool or some other layer.
  - It’s not considered part of the shard/OR mapping layer.
- It’s possible to shard across different databases as long as you keep the same schema in the same in each database.
- The number of shards you can have is somewhat limited because each shard is backed by a connection pool which is a lot of databases connections.
- ORDER_BY operations across databases must be done in memory so a lot of memory could be used on large data sets.

Bibliography

- http://www.integrasco.com/

- http://highscalability.com/
- http://en.wikipedia.org/wiki/Scalability

- http://shards.hibernate.org/
- http://hadoop.apache.org/