Achieving Read/Write Scale Without A Complete Rewrite

A Customer Case Study: Avanza Bank

March 2012

Introduction

Many applications are built with layers upon of layers of development, with relational databases at the heart of the system. Scaling these systems is extremely challenging, leading many organizations to take the easy route – simply paying more for higher-end hardware and databases. Today, we have reached the point where this approach often does not work, and is simply too expensive to maintain with the advent of cheaper, more scalable alternatives.

The case of Avanza Bank presents an excellent example of how it is possible to turn an existing online banking application into a new site that is designed for read/write scaling.

Snapshot

Industry: Banking

Primary Challenge:

Scalable transaction processing for trading platform; increase performance and efficiency of banking processes compared to Oracle's solution

Solution:

GigaSpaces provides infrastructure for Avanza's core services, including trading, customer and data processing, and storage.

Results :

- Zero system cost for data storage (vs. half system capacity before)
- Massive cost reduction Major performance improvement

Existing System Description

Avanza Bank is a leading Swedish bank, which runs the most trades on the Stockholm Stock Exchange. Avanza prides itself on providing advanced tools for its investors through its online banking system, which is designed to make it easy for investors to carry out equity transactions and fund switches.

The existing Avanza online system was a typical web site based on Java/JSP and Spring.

Scaling Architecture of the Existing Site

Most of the interactions with the site were read-mostly, where the main scaling challenge was scaling concurrent read operations. Read scaling was addressed through a side-cache architecture, which is common with many of the existing LAMP + memcached deployments where the first query hits the database and the following queries hit the cache.

New System Description

The new site was designed to fit into the real-time and social era. This would mean a great increase in traffic and activities generated by users and not just by the site owner. These activities must be presented to bank users in real-time.

Challenges

The changes to the new site lead to a significant change in the traffic and load behavior that drives a new class of scaling challenges.

Write scaling

When multiple updates are involved, the existing side-cache architecture causes diminishing returns as the cache becomes obsolete rather quickly – meaning that synchronizing the cache only adds overhead.

Using Oracle RAC with a high end hardware platform didn't prove itself either and yielded a fairly expensive solution that didn't meet the scaling requirements.

Unlike "greenfield" applications, Avanza has an existing online application ("brownfield") that serves its current customers. That brings the following list of additional challenges:

- **Existing Data Model**

The entire data model of the application was designed for use with relational data store. Changing the data model or moving it to a new NoSQL architecture, as Avanza had considered, would involve a huge change that could turn into a potentially years-long effort.

- **Legacy System**

The online bank application consists of large set of legacy applications and third-party services. Rewriting the existing infrastructure was either impossible (due to the dependency on third-party tools) or impractical.

- **Complex Environment**

As it often the case, a large portion of the legacy applications were not designed for scale, and were not built with a clear holistic architecture, because they were built in layers over the years. This vastly increases the complexity of scaling.

- **Existing Skill-Sets**

The existing development team already had fairly good knowledge of Java and Java EE. Changing the team and/or developing a completely new skill-set is a huge barrier, because the ramp-up time required to bring new developers up to speed with the complexity of the system can take years.

The Solution: Read/Write Scale Without A Complete Rewrite

It was clear that meeting the new scaling challenges would involve changes to the existing application. The main question was how to scope that change so that it wouldn't require a complete rewrite. The second goal was to build the changes in a way that would reduce the TCO of the system.

To implement a system without a rewrite, Avanza did the following:

- **Minimize the change by clearly Identifying the scalability hotspots**

The application areas requiring intensive write access are often small parts of the overall system. The first step is, therefore, limiting the change to only those hot-spots of the application, leaving the rest of the application untouched. In Avanza's case, the hot spots were identified in specific tables used by the online web application. Most of the back-end systems were still accessing the database for reporting, synchronization, and batch processing, and could therefore remain unchanged.

- **Keep the database as-is**

One of the key pieces in the existing design was the ability to address the read/write scalability outside of the database context (see the next bullet). This makes it possible to keep the existing database and the schema of the data unchanged. This way, the rest of the systems continue to work with the database as if nothing changed.

- **Put an in-memory data grid as a front end to the database**

Scaling the application is done by front-ending the application with an In-Memory-Data-Grid (IMDG). The IMDG contains all the hot tables or rows of the original database. The online web application accesses the IMDG instead of the database. The IMDG is inherently distributed, enabling scalability by distributing the load over a cluster of machines for both read and write operations.

- **Use write-behind to reduce the synchronization overhead**

Updates from the IMDG to the underlying database are done asynchronously in batches through an internal queuing mechanism (redo log).

- **Use O/R mapping to map the data back into its original format**

In many cases, to achieve the best scalability, the data must be partitioned. This often involves changes to the data schema. Changing the data schema can break the entire system, including the areas that do not suffer from the scalability bottleneck. To resolve this mismatch, the data schema is scoped for changes to the IMDG only. The data is mapped from the IMDG schema to the original schema through standard O/R mapping tools, such as Hibernate or OpenJPA.

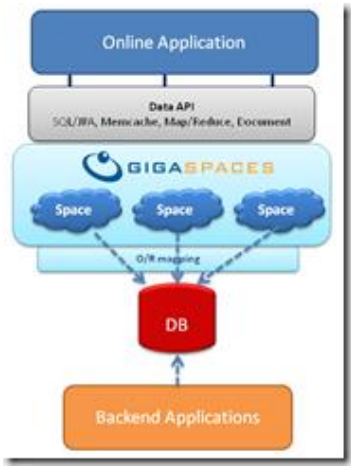
- **Use standard Java APIs and framework to leverage existing skill-set**

One of the challenges with many of the new NoSQL database alternatives is that they often force a complete re-architecture. This comes with the fairly high cost of rebuilding the skill-sets within the organization across the board for developing against new APIs as well as maintaining capacity and sizing of those new databases.

IMDGs (such as GigaSpaces) expose standard APIs, such as JPA. They enable organizations to extend the use of the existing database while removing a large part of read/write load – both the use of standard APIs and existing database enables organizations to leverage their existing skillset and still meet their scalability requirements. It also enables them to take a smoother transition (through baby steps) into a completely new scale-out architecture by enabling a new scalable database to be plugged in at a later stage.

- **Use two parallel sites (old/new) to enable gradual transition**

Switching all customers into a new system at once is risky. A better approach is to enable gradual transition of selected customers into the new sites. A common way to do this is to run two parallel sites. The challenge is synchronizing the two parallel sites. Avanza used the GigaSpaces Mirror Service to synch all the changes from one site to the other, keeping both sites up to date.



Visual summary of this approach

The TCO Angle

The second goal in the project was to reduce the system TCO.

To reduce system TCO, Avanza did the following:

- **Use RAM for high-performance access, and disk for long-term storage**

A RAM-based solution can be 10x-100x cheaper than a disk-based solution for high- performance applications (see *Memory is the New Disk for the Enterprise*). In addition, the price for RAM is constantly going down.

The optimal solution, therefore, is to use RAM to manage data that needs high speed write/read access and disk-based storage for the long-term data that is accessed less frequently.

- **Use commodity database and HW**

A single instance of an Oracle RAC deployment can cost up to $500K. Putting a data grid in front of the database eliminates the need for many of the high-end features available in the Oracle RAC database. It also eliminates the need for high-end hardware such as storage devices, InfiniBand networks, and so on.

Avanza was able to move the data into MySQL and use commodity Dell machines to run the entire relational data system.

Conclusion

In a world where the impact of software accretion is no longer tolerable, it is clear that a change is necessary and inevitable if we expect to meet the new demand for scalability. The real question is how to make this change. The approach taken by Avanza Bank in their use of GigaSpaces is an excellent example: it shows that they were not just able to quickly meet the new scaling requirements through measurable and easy-to-implement small changes, but they also reduced their cost of ownership significantly as noted recently by Avanza CTO Ronnie Bodinger:

"Throwing out Oracle's sluggish databases has increased performance of Avanza's business-critical systems, while reducing license costs significantly… The Oracle cluster we have today costs a whole lot. The new system costs a fraction of that."

- *See the full article (in Swedish)*
- *View a recording of a presentation by Ronnie Bodinger*