HIGHLY SCALABLE CONTENT RATING SERVICE: A CASE STUDY

Nikolche Mihajlovski
Tricode
Skopje, Macedonia

Abstract

A web application is scalable if it can increase its working capacity while maintaining consistency, end-to-end latency and the same code. The paper describes a case of building a basic, but highly scalable web service which provides rating functionality for arbitrary content on any web site that is using it. The architecture is based on "Shared Nothing" web cluster for the presentation layer; In-Memory Data Grid cluster for data storage and processing; and relational database for asynchronous, long-term persistence. The design and implementation are based on the Space-Based Architecture concepts and GigaSpaces XAP middleware.

Introduction

A web application is scalable if it can increase its working capacity while keeping three things constant:
- consistency of the business logic and data — otherwise when you scale, things won't work.
- end-to-end latency - otherwise when you scale, you might be supporting more clients, but degrading the service level each client receives.
- the application code — otherwise you aren't scaling the application, you're replacing it.

Web application scalability is a hot topic nowadays. More and more theory, patterns and technologies for writing scalable web applications are emerging... In the text that follows it is shown how to combine some of them to build a highly scalable web service for content rating.

Challenge

The main goal was to design and implement a basic, but highly scalable web service which provides rating functionality for arbitrary content on any web site that is using it. It can be accessed through rating widget (e.g. 5 rating stars) embedded on any web site. The minimal requirements for such service are:

- access/usage through embedded widget on any web site,
- basic authentication of users,
- adding/changing/removing content rating,
- calculation of average content rating.

Although the described system has just basic features, the main challenge is that the system can scale linearly, to support high load at any time in future.

Architecture of solution

The scalability challenge dictates the architecture of the solution, which is based on a Shared Nothing Architecture (SNA), as a common architectural pattern for scaling out web applications. It is a distributed computing architecture in which each node is independent and self-sufficient, and there is no single point of contention across the system. So, such system can scale-up simply by adding computing nodes.

The system architecture, as shown on Fig. 1, is based on the following parts:
- a cluster of independent web servers without shared state data (Shared Nothing pattern),
- load balancer which spreads the incoming requests across the web servers,
- In-Memory Data Grid (IMDG) which keeps the shared state data partitioned,
- relational database for long-term persistence of the shared state data.

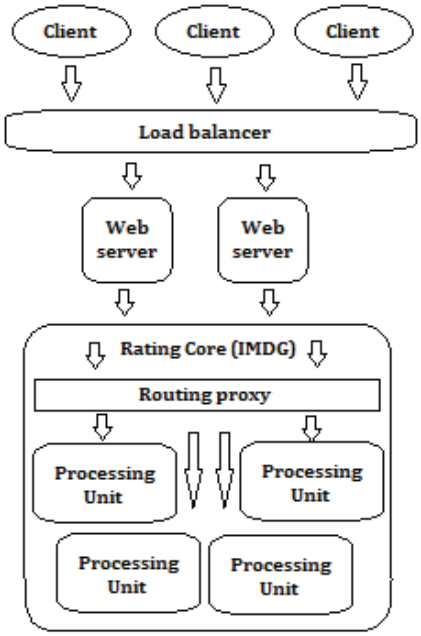

Figure 1: Architecture

A typical Shared Nothing web architecture would have the processing logic executed on the web servers, requiring communication with the IMDG to read/write the shared data. In contrast, the solution is based on Space-Based Architecture [1], an approach for high throughput / low latency implementation of linearly scalable systems, having the processing logic running on the IMDG, collocated in the same processes with the shared data. Thus, the IMDG is the core of the system (Rating Core), handling the shared data storage and processing.

The IMDG is clustered in a combined replication/partition topology. The scalability of the shared state data storage and processing is achieved by partitioning the IMDG in self-

sufficient processing units. Each of them can perform the entire processing logic on its own and keeps all data needed for it. Thus, scaling out the Rating Core can be achieved simply by adding more processing units to the IMDG.

The IMDG maintains its reliability normally in memory using replication, so the data can be written into the relational database asynchronously to avoid hitting the typical database bottleneck. In case of eventual lack of memory for keeping all the shared state data in the IMDG, the oldest data will be evicted from the memory, but not lost, since it will already be persisted in the relational database. In such scenario the IMDG will become a read/write LRU cache for the relational database.

Design & implementation of solution

The domain model of the solution is fairly simple, as shown in Fig. 2. The system was designed and implemented around two main principles that helped the achievement of high scalability:
- incremental calculation,
- eventual consistency.

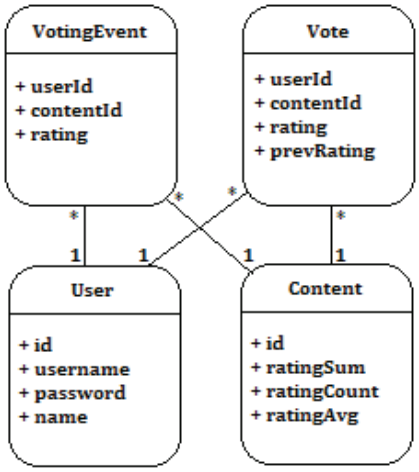

Figure 2: Domain model

In a typical use-case of voting for a content, the web server sends an event message to the Rating Core, which routes it to the appropriate processing unit. The event (VotingEvent model) is enqueued and asynchronously processed on the processing unit. The event processing logic stores the voted rating for the specified user and content (Vote model), also updating the total/average rating of the content (Content model).

The most important consideration in terms of design was the data affinity / partitioning strategy of the system. The fact that all rating operations are always executed per one single content implicates independence between the voting/rating state data for different content. And the correlation between voting/rating state data for the same content implicates the data affinity per content. Thus, the content ID was selected as a natural partitioning key, so all the data for a certain content would be on the same partition. That makes each processing unit self-sufficient and independent.

The design and implementation of the solution is based on the GigaSpaces XAP middleware. It is powerful starting point for implementation of Space-Based Architecture projects. All low-level aspects of clustering (replication, partitioning, content-based routing, etc.) are handled by the middleware, so the development effort was focused on the higher-level application logic.

Results

The system was created and tested in experimental conditions, and is not used in production environment. It takes advantage of Space-Based Architecture principles and GigaSpaces XAP features to result in several great accomplishments:
- high scalability (due to architectural partitioning of the web cluster and IMDG cluster into self-sufficient processing units),
- high performance (due to in-memory data storage and co-located processing),
- rapid development (due to GigaSpaces XAP high level of abstraction over the cluster as platform and nice integration with standard Java technologies).

References

[1] Nati Shalom, The Scalability Revolution: From Dead End to Open Road, 2007.