Automation of business processes using a scalable resource management information system

Tilivaldi Amrayev, Rassul Yunusov

Sabayev University, Kazakhstan

**Corresponding author's e-mail: tilvaldi@mail.ru*

Abstract

The organization of a fault-tolerant and productive resource management system using models of distributed computing systems is an important task in the conditions of high competition and growing data volumes and intensity of operations performed on them. This paper discusses various approaches to solving the problem of providing enterprise resource management services. A comparison of widely known architectures with the "Monolithic" organization of the application and service-oriented architecture as "Microservices" is made. The disadvantages and advantages of those and other approaches are considered. The purpose of this article is to determine the optimal approach in solving the problem of building scalable information systems.

Keywords: Orleans, Monolith system, Redis, Microservices

1 Introduction

Paper covers well-known problems of high-load. The main goal is to determine the optimal solution for scalable software, that can be used in very high load conditions and maintain availability and resilience. For current businesses it is a very crucial question to provide such a service, that can handle millions of requests in reasonable time. As an example – the ability to have flexible hardware architecture, that allows to use only adequate amount of processing powers for existing demand, and being able to scale up horizontally if needed. Such a situation is common for nowadays – and well known as Black Friday issue, when millions of customers try to buy merchandize under discount from all around the world in short period of time. And it is very important for a business owner not to lose a single transaction. The business process of providing services to individuals, a company with a large number of service points and a high intensity of operations performed is considered. Selection of the optimal architecture for the organization of such a business process is critical to the success of the enterprise. The wrong choice of information system architecture can have negative factors, such as: financial, reputational, production risks of the company.

Therefore, this article reviews the existing approaches and technologies for solving typical problems of automating business processes. At the same time, modern problems of organizing high-performance services in the context of dynamically changing resource use intensity are considered.

The paper proposes to consider two approaches in solving such problems:

- "Monolith";
- "Microservices" - as a Service Oriented Architecture.

Monolith system - a control system or application implemented as a single unit, including the application interface, database, and systems for interaction between them. A lot of software has been made using such an approach. And this approach proved itself as a great solution, that helps to determine quickly business needs and implement the solution as ready to use solution with full coverage of required functions.

Distributed system - a system in which there is no main data processing center, and also differs in ease of scaling and scanning the system in "combat" production conditions. The main difference of the system is the location on different physical servers of parts of the application (database, interface, and other software components).

Each type of architectural solution has its pros and cons. Advantages of the Monolith concept: low cost and high speed of development, an application and database that are clearly interconnected, a large choice of developers. The disadvantages of the concept are: the difficulty in adding a new functional, limited physical characteristics, poor fault tolerance, the inability to reuse a component.

Advantages of the "Distributed Systems" architecture: scalability, fault tolerance, high speed of adding new functionality. The main disadvantages include the high cost of initial development, as well as a small number of developers who have experience with this architecture. Microservice architecture is oriented to use distributed architecture from the start. And it is concentrated to split a complex task into small subtasks with its own business context and business data. Such an approach provides and ability to select appropriate storage system for each kind of a business process. Provide an ability to chose a better technology stack to solve a problem, including programming language, storage system, existing libraries and specialized software.

2 Existing solutions

One of the projects implemented on the basis of the Monolith architecture is the project "Optimization of evacuation plans for people based on wireless sensor networks". In the project, a relational model was chosen as a database management model. The relational model allows for easy integration of information systems and applications. When implementing the project separately from the database, a file storage system is implemented, where command signals are stored. The language used is C# and ASP.NET technology, for a server application for managing a dynamic evacuation process, and C++ for signal processing [1]. The biggest problem of such a monolithic approach – is difficulty in scaling of a solution if it would have to deal with billions of sensors.

One of the projects implemented on the basis of microservice architecture is the project "Kolesa-Krisha-Market", a local company in Kazakhstan, that provides an online service of real estate market and automobile market and now also the merchandize online store. One of the main factors showing the need for implementation through microservices is customer focus, namely, fast and high-quality implementation of recording ad views. "Kolesa-Krisha-Market", while working on their own project, faced the main problem, lack of speed, as well as possible duplication of information. Work in one database is limited by the physical characteristics of the media, thereby limiting the scalability of the project. And also during the work of any project the failure of the database is possible. It is in connection with these restrictions on the project that the Go language bundle and the Redis data warehouse are used. Redis is a data warehouse, the main advantage of which is the fast processing of information due to the delayed recording of information in the database. Redis is used as a cache that creates a queue of tasks. Creating a queue of tasks, using Redis, writing to the database (Mongo DB) was carried out asynchronously, which allows the use of fast processing of information loaded on the service. One of the project's solutions was an approach to information processing, namely the understanding of the shortcomings of Mongo DB, the periodic delays in recording information on physical media. Nevertheless, there was a risk of information loss, not only because of the disconnection of the physical media on which Redis is deployed, but also in connection with the transfer of the counter from the Redis cache to the Mongo DB database. The solution to this problem is carried out using the "secure queue" template. Using this solution allows you to reduce losses due to the creation of copies of processed items in the queue, separate from the main one. A queue of copies will exist until they are permanently stored in the database. As a result of this solution, microservice based on the Go, Redis and Mongo DB bundles successfully working under load, and is ready for periodic unavailability of one of the data stores.

In addition to fast data processing due to the lack of writing to physical media, Redis has several advantages in contrast to traditional DBMS. The flexibility of the data structure allows you to maintain multiple data structures, such as: strings, lists, sets, hash tables, and others. Redis also allows you to simplify writing code using fewer lines for storage, due to the built-in structures for data processing. Improving the read operation, by distributing requests between servers, is achieved using a master-slave architecture and supporting asynchronous replication [2].

Ceph is the open source software used as a distributed file system. The CRUSH algorithm used in the Ceph framework allows the storage clusters to be freed from the limitations of centralized data storage tables. The algorithm is used in addition to traditional file systems, allowing you not to change the entire infrastructure and does not require the operation of additional equipment. The algorithm makes it possible to uniquely determine the location of an object based on the hash of the name of the object and a specific map, which is formed based on the physical and logical structures of the cluster [3].

RabbitMQ is a lightweight message broker that allows queuing between applications for data exchange [4]. In turn, this brings benefits in the form of load balancing and work sharing among applications. RabbitMQ comes with several interfaces, so that applications between which the exchange is carried out can be written in different program languages. RabbitMQ is implemented on the Actor model conceptual model, where each entity is an actor. The actors are separated from each other and do not share the memory, so that one actor cannot directly change the status of another actor [5].

Hadoop Apache is an open source project supported by the Apache Software Foundation. Hadoop is used to manage scalable, distributed computing, or as a general-purpose file storage. The distributed file system (HDFS) and the MapReduce system are two key components of Hadoop [6]. HDFS is the main data storage system that repeatedly copies blocks of data and distributes these copies among the computing nodes of the cluster, ensuring high reliability and speed of calculations. MapReduce is a framework for writing applications for high-speed processing of large amounts of data on parallel clusters of computational nodes.

Apache Spark is a platform for distributed data processing that uses distributed memory to efficiently process large amounts of data. Apache Spark performs calculations in RAM, bypassing the disadvantage of Apache Hadoop, namely, the expectation of building a map [7]. Thanks to RDD (resilent distributed dataset) technology, Apache Spark performs calculations only when it is necessary to output the finished result. In this case, the use of RAM, limits the amount of data processed. Limiting the amount of data performed does not allow Apache Spark to be considered a more advanced system than Apache Hadoop.

Storm is a fault-tolerant computing system focused on distributed processing of large data streams, similar in functionality to Apache Hadoop, but operating in real time [8]. The advantage of using Storm is solutions focused on several languages: Java, C #, Python, thereby simplifying the development of the system. Storm works by using network topologies, rather than special tasks like in Apache Hadoop. Using Storm ensures that each incoming message will be fully processed even if the analysis is distributed among hundreds of nodes.

When using Storm, only three abstractions are applied: spout, bolts, topologies. Spout is the source of computing threads (figure 1). Spout read messages from queue brokers such as RabbitMQ, Kafka, but can also create its stream, or read from the streaming API. Spout implementations already exist for most queue systems. Bolt processes any number of

inputs streams and creates a series of new streams. Most of the computational logic comes in bolts, such as functions, filters, streaming connections, streaming aggregations, connecting to databases, and so on. Topology is a network of spouts and bolts, with each edge in the network representing a bolt subscribed to the output stream of another spout or bolt. Topology is an arbitrarily complex multi-stage flow computation. Topologies run indefinitely when deployed.

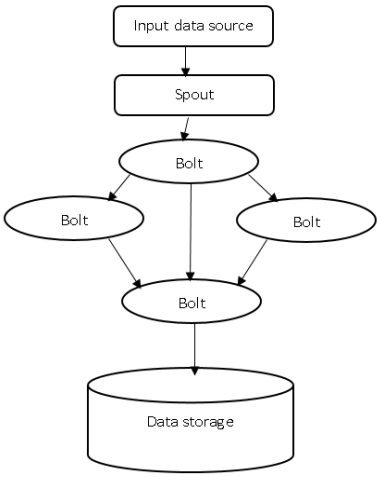

FIGURE 1 Basic concepts of Apache Storm

As a conclusion of different solutions that are aimed to solve a different problems, according to demand for the business process, it is much more easy to build a scalable solution now than ever before. But usage of them requires additional experience and competences in questions of building the resilient business processes within an integration of small Microservices, taking in consideration their asynchronous nature.

3 Orleans frameworks

Orleans is a software environment for creating scalable and flexible cloud applications. It is based on .Net Framework and now available for Dotnet Core plaftorm as an open source solution. Thus, it has all the benefits of multi-platform software. Initially created by Sergey Bykov and evolved at Microsoft into a mature solution, that have proved its efficiency in many active projects including online gaming platform Halo. One of the main concepts of the framework is the use of simple templates of parallelism. Its basis is the Actor model. The actor-like component is called Grain (grain), which is an isolated unit of computational state, the interaction of which is carried out through asynchronous messages [9]. Each Grain has a unique identifier, which is composed of its type and primary key (128-bit GUID)

The isolated state of the grain and the limited execution model allows Orleans to save, transfer, replicate, and reconcile the state of the grain. One of the main aspects of Grain, is a permanent existence, since the "grain" is a purely logical entity, and exists virtually. Grain cannot be explicitly created or destroyed. Grain will carry out its work and influence the application only from the moment of its activation, until its deactivation [10] (Figure 1).

Grain activation is always done one move at a time. The step-by-step model of asynchronous execution allows you to create queues for multiple activation requests. Deactivation of Grain, occurs not only when the deactivation method is called directly, but also in the state when the system does not address this Grain for a long time [11]. Thus, the system, even when there are many Grain, in working condition uses only a small part of the total. Figure 2 shows the structure of the work of Orleans, in which only a part of the “grains” (highlighted in red) are in the active state.

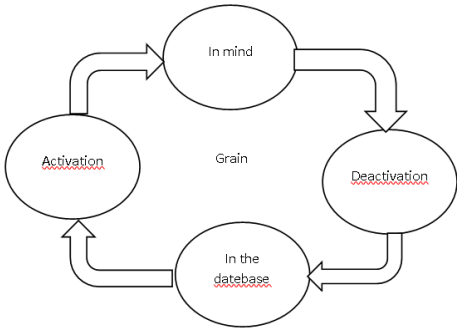

FIGURE 2 Grain operation diagram

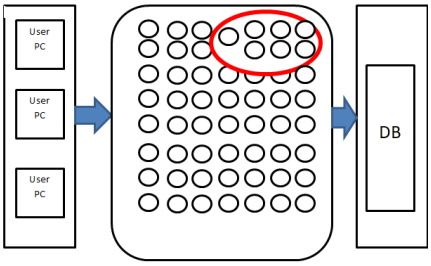

FIGURE 3 Structure of the Orleans and Grain

Orleans is currently being used as a platform for building and launching cloud services, thanks to the scaling capabilities of the framework.

The next advantage of Orleans, is a constant view of the server cluster failures, thus, when one of the servers is disconnected, server recovery and cluster reconfiguration are provided. When shutting down the server, Orleans (Figure 4), knows which grain, did not complete its work at the time of shutdown, thus the grain that worked at the time of shutdown on the server is automatically restored on another server from this cluster.

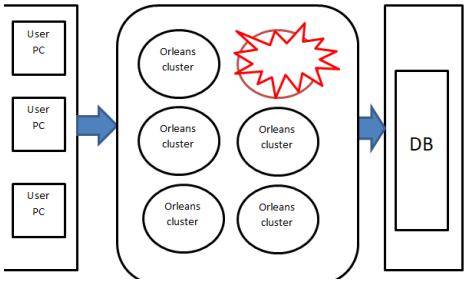

FIGURE 4 Failure of the Orleans cluster

4 Conclusion

Thus, in the course of the work were studied:

1) Advantages and disadvantages of distributed systems and systems architecture "Monolith"

2) Existing solutions based on microservice architecture are considered.

3) Orleans framework

As a result of the analysis of existing architectures, microservice architecture was chosen as the main architecture, based on the Orleans framework. The Orleans framework uses the Actor model, through isolated grains that exchange data through asynchronous messages. The isolated state and constraint in the runtime environment allow Orleans to save, transfer, and reconcile the state of the grains without programmer intervention.

It is intended to use Orleans to simplify the development of the application, due to design patterns that ensure the scalability and reliability of the project.

Usage of Orleans framework simplifies the process of building distributed software. But it is still require understanding of distributed computations concepts to build efficient solution. It is crucial to keep the balance between entities that have to be Grains and that don't have to be grains. It is also important to state that there are still open crucial problems that we have to solve – such as building resilent diestributed transactions, that are already a part of Microsoft Research Team endeavors, and distributed indexing of grains by different criterias

References

[1] Amirgaliyev Y, Yunussov R, Mamyrbayev O 2016 *Optimization of people evacuation plans on the basis of wireless sensor networks*
[2] Wilson J, Redmond E 2012 *Seven Databases in Seven Weeks*
[3] Rosenberg N A 2006 Standardized Subsets of the HGDP-CEPH Human Genome Diversity Cell Line Panel *Accounting for Atypical and Duplicated Samples and Pairs of Close Relatives*
[4] Andreassen O O, Marazita F, Miskowiec M K 2017 *Upgrade of the CERN RADE framework architecture using Rabbit MQ and MQTT*
[5] Rabiee A 2018 *Analyzing Parameter Sets for Apache Kafka and RabbitMQ On A Cloud Platform*
[6] Chansler R, Kuang H, Radia S, Shvachko K 2010 *The Hadoop Distributed File System*
[7] Alsheikh M A, Niyato D, Lin S, Tan H, Han Z 2016 *Mobile big data analytics using deep learning and apache spark*
[8] Yang M, Ma R T B 2016 *Smooth Task Migration in Apache Storm*
[9] Bykov S, Geller A, Kliot G, Larus J R, Pandya R, Thelin J 2011 *Orleans: cloud computing for everyone*
[10] Bernstein P A, Bykov S, Geller A, Kliot G, Thelin J 2014 *Orleans: Distributed Virtual Actors for Programmability and Scalability*
[11] Bykov S, Geller A, Kliot G, Larus J R, Pandya R, Thelin J 2010 *Orleans: A Framework for Cloud Computing*