Mykyta Ruzhevskyi, Yuri Gordienko.

IMPROVING SYSTEM FAIL-PROOF THROUGH THE USE OF A MESSAGE BROKER WITH EVENT-DRIVEN ARCHITECTURE.

This article discusses ways to improve system resiliency in the presence of outdated components that are weakly scaled or do not scale at all, and have a high complexity of modernization. A way to increase fault tolerance is to isolate the problem component from other elements of the infrastructure and change the way other components interact with it to the message broker. After the implementation of this method of interaction, the development of new components is simplified through the use of the same tools for interaction, and the original component as a reference source of test data.

Key words: microservices, apache kafka, failure-proof, resiliency, message broker

Target setting. In the modern world, there are certain trends in software development:

- Monolithic applications are being splitted into the microservices
- Non-core functionality is being delegated to other services
- On-premise hosting in being replaced to the cloud solutions
- Old-fashioned SQL databases are being replaced by the variety of NoSQL-databases, with different architecture and data storage approaches

Alongside with freshly-built modern applications there are the legacy ones. Those legacy applications are very hard and/or expensive to modernize because of:

- It is critical to have bug-free-prooven software (e.g. healthcare or military software, where bugs can cause casualties)
- High certification price for newly developed software (e.g. in aircraft industry certificates can cost millions of dollars each)
- Lack of domain experts to identify weaknesses and nuances
- Lack of resources to develop updated component options

Often, such software products are poorly scalable horizontally or not scalable at all, while the load on these systems regularly increases. As of Cisco Annual Internet Report Trends, the active internet user count increases every year, from 3.9 billion users in 2018 to 4.5 billion users in 2020 and expected to be over 5.3 billion in 2023.

Therefore, with growth of internet user count, traffic also has grown. The cheaper the network connection became, the wider and easier the internet spread. Also, there is migration from wired internet connection to mobile and wireless networks, that are active almost 24/7. And the wider the internet spread, the more often our services can be accessed, therefore the more load on infrastructure is being done.

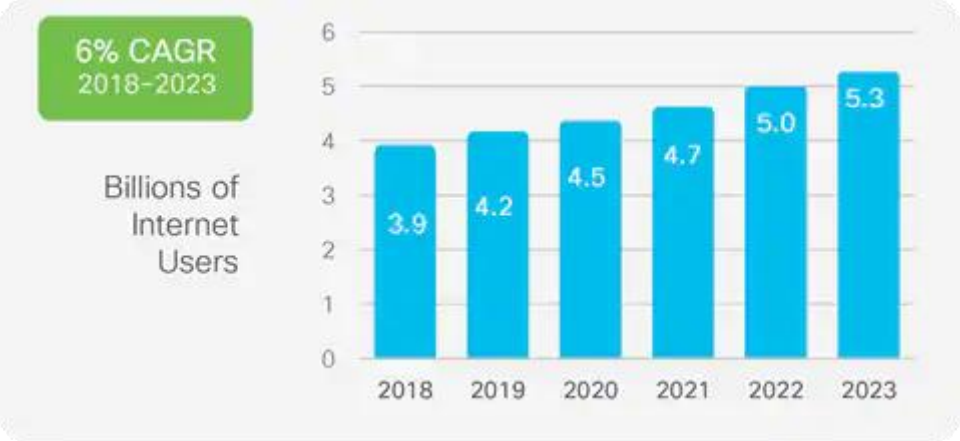

Fig. 1. Ciso active internet user trend [1]

Actual scientific research and issues analysis. In normal conditions these systems are still able to handle the workload assigned to them, but situations of "peak" load can cause the system to fail. Regularly, these incidents jeopardize the entire infrastructure and cause the entire architecture to "fall down". As a result, businesses have troubles with work processes, disappointed clients or financial losses.

As an example, let's look at several examples of the sources of such peak loads:

– The launch of a new advertising campaign, due to the novelty factor, causes an increased number of active users in the system (on the site) for several hours

– The so-called "slashdot effect", which occurs when a popular website links to a smaller website, causing a massive increase in traffic. This overloads the smaller site, causing it to slow down or even temporarily become unavailable.

– In the banking sector, the "pay-day" is associated with an increased load on the processing of banking systems, as users are much more likely to update balance sheet information

– Problems on competitors' sites force users to look for an alternative and also increase site traffic, until competitors fix unavailability of service.

At the same time, most of these load jumps are of a short-term nature, but at the same time, in the case of a well-built user experience, it can increase the long term popularity and make clients loyal to your product.

However, as already mentioned, the existence of components that are not designed for such load surges negatively affects the stability of the entire system, up to its complete fall.

The research objective. The purpose of this article is to create a method that allows developers to increase overall system stability by keeping the legacy component in bounds of it's designed load.

To solve the overload problem, there are the following standard ways to solve it:

– Horizontal scaling - deploying the application on more physical hardware

(servers). The created cluster splits the load and can process the number of servers times more requests, than a single node. This requires a special architectural design from a software engineering team and this is the industry standard way to solve this issue.

– Vertical scaling - deploying the application on more performant hardware to increase it's throughput. The main disadvantage of this method is that the maximum performance of hardware is limited.

– Caching - storing previously computed data in pre-built data structure and using it in case when previously computed data is required.

– Replication - duplicating services that heavily rely on read/write operations, such as master/slave replica set, when all data writes are made to one specific node, which transfers all the updated data to slave replicas, which have been used for read operations.

As we can see, the methods above can reduce the possibility of a component failing, however, not every system allows horizontal scaling, vertical scaling is limited and the resource savings provided by caching will not help save much resources and not every operation can be cached. Furthermore, replication approach also requires the modification of existing software.

After done a research and experiments I have figured out the following ways to improve stability over weak and non fail-redundant components:

1. Predicting server response based on mathematical induction and machine learning.

2. Smoothing load peaks by using Message Queue combined with asynchronous architecture instead of synchronous requests.

The machine learning approach can be used only in specific tasks, that does not require complete accuracy and discrete values (for example, they can be used in weather forecast API and cannot be used in ticket booking systems). Otherwise, the second approach is suitable for this kind of data but unsuitable for applications that require immediate response from the server.

In this article the Message Queuing approach will be discussed.

The research results. Modern microservice communication is usually based on HTTP and SOAP requests between two microservices. Furthermore, message queue services such as Kafka and RabbitMQ are usually used as event bus, as a broadcast tool to notify other services about changes that are being made in data.

The main common point of REST and SOAP is that they both are application-level protocols and that they both are based over the HTTP protocol, which is synchronous by default. Every non-typical increase in workload can cause performance degradation and component failure. Furthermore, the more requests are made to the server, the more load is being done, the more requests fail, the even more requests are made.

The message queue approach works different:

1. Client creates some operation (e.g. registration)
2. REST proxy accepts the request, validates it and puts this request in the inbox queue.
3. When a component is able to process a message, it takes a message from the queue. Firstly, messages from a higher priority queue are taken, then - the less priority.
4. Depending on the ability of the server to process requests simultaneously, the number of parallel processed messages is been configured.
5. After processing the message, the result is placed in the outgoing message queue marked with the message recipient.
6. REST proxy receives a response and generates a message for the client.
7. If the response from the component was received before the response timeout expired, the response will be returned.
8. In another case, a message with the result of the operation (if required) can be saved in the response database as a message, delivered via push notification or in another way.

For messages with the highest priority, you can use HTTP requests though. Thus, messages that come from the request queue will be processed later; however, the client will see the most relative data (that was previously delivered via HTTP).

Moreover, this approach allows you to gradually replace certain components with a new developed system:

- Messages can be read by two different versions of the service
- The results are written in different queues, at the end of the processing of the message by both services, the results of the work are compared
- The old system acts as a reference for the new
- In the case of a long absence of errors, the new system can be considered true and the previous system can be used only in case of execution errors

By iterating over the existing system in result we will create the tested, scalable system with a complete feature set of it’s ancestor.

Conclusion. Despite the fact that the best way to get rid of the problem of a system failure is to replace an unstable component, its stability can be improved by replacing the interaction interface with a problem component with one that can prevent the problem component from being overloaded by limiting its load.

References

1. Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper (https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html)

AUTHORS

Mykyta Ruzhevskyi - postgraduate student, Department of Computer Engineering, National Technical University of Ukraine "Igor Sykorsky Kyiv Polytechnic Institute".

Email: sleeper253@gmail.com

Yuri Gordienko (supervisor) – professor, Department of Computer Engineering, National Technical University of Ukraine "Igor Sykorsky Kyiv Polytechnic Institute".