ACCELERATING PERFORMANCE IN CRITICAL TOPOLOGY ANALYSIS OF DISTRIBUTION MANAGEMENT SYSTEM PROCESS BY SWITCHING FROM MONOLITHIC TO MICROSERVICES

SEBASTIJAN STOJA, SRDJAN VUKMIROVIC, NIKOLA DALCEKOVIC, DARKO CAPKO, BOJAN JELACIC

Key words: Microservices, Monolithic, Architecture, Distribution management system (DMS), Topology analysis, Electricity network.

A distribution management system increases system efficiency optimizes power flows, and prevents overloads. Such a system whose responsive time is significant, using microservices in this paper is studied. This paper proposes an application for the distributed topology analysis of an electricity network model. The application relies on microservice architecture; one service being divided into multiple microservices, with Microsoft Azure Service Fabric used for deploying microservices. Testing is performed on different test cases and the benefits of obtained results for using such an application in microservice environment is demonstrated. The results obtained are compared with monolithic results of such an application. The results demonstrated required quicker execution times for calculations deployed as microservices.

1. INTRODUCTION

Microservice [1, 2] is one of the recent terms in software architecture. It refers to the development of applications in the form of small package services, each package running in a separate process, and communication of resources is manifested through the exchange across the protocols. Services are built independently, therefore they are spaced apart on the machine. In other words, microservices are small autonomous services that work together.

To understand the concept of microservice, it is important to make a comparison with monolithic applications that are built as a single unit. Enterprise applications are usually made of three parts: the client interface, database and server-side applications. Server-side application is a single service, a monolithic service. Any changes to the system involve building and deploying a new version of the server-side application. Because of the cycle of new deploying applications when service is changed, using the monolithic applications is more complicated and time-consuming, causing more frustrations.

All the above disadvantages of monolithic applications have led to the construction of microservice architecture: building applications as a set of services. On one hand, in monolithic applications, functionality is placed in one service, and the monolithic service can be scaled increasing the service, while on the other hand, in microservice architecture each individual functionality is placed in a single service and these services are scaled to the servers where necessary.

Microservices are solving one of the main problems of IT companies, which is deploying the whole system after changing one service. By using microservices, that problem is solved by decentralization and faster deployment, where changing one service does not affect the whole system. Microservice provides customers with new functionality more quickly, and changing functionality in one service does not require the need to change the others. The main purpose using a microservice is to make changes in one service and then deploy it, without changing the rest of the system.

When discussing the application of a microservice, there is a need to make comparison between service oriented architecture (SOA) and microservice. SOA is an architectural pattern in which application services provide services to other components. Service in SOA can belong to the same application, while microservice each microservice is suite of independently deployable service. SOA is regularly used in monolithic application.

Having in mind all these advantages that microservices offer, in this paper we use microservice architecture in a distribution management system (DMS) [3, 4]. DMS systems are used to increase system efficiency, optimize power flows [5] and prevent overloads. They are working in real-time where time for reading and writing is essential, so that the client gets a quick interaction and reacts to the problems. Also, the system should be scalable without major failures when it manages electricity networks. For all these features, using microservices can provide a solution. From those systems, one function topology analysis is chosen and is analyzed in this paper.

Architecture of an application, which is calculating topology analysis, is proposed, and tested using Microsoft Azure Service Fabric. A comparison is also made between monolithic and microservice architecture, and advantages and disadvantages of using such systems in microservice environment are shown, relying on obtained results.

Microservices can leverage publisher and subscriber messaging middleware [6] while each microservice should have its own database [7]. Although we researched how microservices should be composed jointly to implement the required solution, in this paper we focus only on the comparison of microservices versus monolithic architectures.

This paper is structured as follows: in Section 2, related work is described. Section 3 introduces distribution management system, and Section 4 illustrates proposed solution. Section 5 is dedicated to experiment and results. Finally, Section 6 summarizes the contributions of this paper.

2. RELATED WORK

In recent years, microservices have been used in applications, transforming monolithic application to microservice, testing, and illustrating the performances and benefits of using microservices.
Stubb et al. [8] offer an extensible solution to the service discovery problem for microservices running in Docker containers, aimed to be easily accessible. It does not alter the original containers and no special infrastructure is needed to start using it - all that is required is Docker and the Serfnode image. The new images form a decentralized cluster of Serfnode, where they advertise themselves providing service discovery. Additionally, they showcased Serfnode extensibility by providing a solution to the file system synchronization problem. But, their study does not include any tests compared to this paper. Also, Docker technology is used in previous literature [9] where the authors discuss some concepts about microservices architecture and how Docker can help to successfully apply the beneficial architectural style with a well working case study.

Villamizar et al. [10] have made comparisons between monolithic and microservice applications. They analyzed microservice architecture pattern to deploy large applications in the cloud as a set of small services that can be developed, tested, deployed, scaled, operated and upgraded independently, allowing companies to gain agility, reduce complexity and scale their applications in the cloud in a more efficient way. In the results, it is seen when using microservices, executed time is reduced for such applications, similar findings are seen in this paper. The only difference between this paper and the aforementioned is that a different approach was tested and different deployment services where used where microservices are deployed. Also, Kecskemeti et al. [11] are making comparisons between monolithic and microservices without testing results and can’t make the comparison.

Testing is one of the most important parts of developing software. In recent years, the behavior-driven development (BDD) has become one of the most popular agile software development processes, and frequently used in microservices development. Rahman et al. [12] explained BDD acceptance tests across multiple repositories and they showed how to reduce the conflict between developers and testers by allowing them to independently iterate on separate repositories while achieving the same application goals. Such an architecture of BDD tests in microservice environment is useful but it should be used in one real system to see the benefits of it.

Balalaie et al. [13] shows us how microservice architecture enables DevOps and they are showing five things that they have learned. First, they solved the problem for developers of deploying every dependent service to run the isolated service on their machine, while simultaneously introducing dynamic service collaboration. Second, the problem for service contracts – since changing so many services that expose their contracts could be error-prone. It can be solved in the way of versioning services, but it results in complex deployment. Another way is the tolerant reader service design pattern, where the team is responsible for service and can be confident that the customers are satisfied with the service. Third, distributed-system development needs skilled developers. Fourth, creating service development templates are important. Finally, system flexibility, which made development much easier. These are helpful insights when making applications in microservice environment.

Dalcokovic et al. [14] achieved a horizontal scalability based on an actor model deployed as a set of microservices for massive data processing. The key is in the data nature. Often processing data with big data characteristics has unrelated or loosely referenced data model that can be processed in parallel and differently. It can be concluded from the paper that such scenarios are implemented more effectively with microservices rather than monolithic as horizontal scalability is one of the major issues in such solutions.

Existing researches mostly focus on microservice architecture, how to build microservices, how to use them and make the parallel between monolithic and microservice applications. According to our research we believe that this is the first work that shows performance of critical topology analysis DMS process by switching from monolithic to microservices.

3. DISTRIBUTION MANAGEMENT SYSTEM

DMS [3,4] is a utility IT system capable of collecting, organizing, displaying and analyzing real-time or near real-time electric distribution system information. A DMS can also allow operators to plan and execute complex distribution system operations in order to increase system efficiency, optimize power flows, and prevent overloads.

One of the functions of DMS system, which is used in this paper, is topology analysis [15]. Topology analysis aims to describe electricity network, that one may have all the functions, which are carried over the network, executed quickly and efficiently. In addition, topology analysis of the electricity network aims to give a complete picture of the network so that there is a clear insight into what is energized, de-energized, regional affiliation, etc.

For execution of topology analysis, an internal model data [16] is used, which creates elements of electricity distribution network in a form that is suitable for the effective performance of the electrical calculations.

The internal model data consists the following: static data with information about the factory (catalog) characteristics of elements and their correlation; dynamic data with information about the data in real time, taken from the SCADA [17] system; and, data topology as the result of the topology analysis.

The relevant elements of the internal model for electricity network based on common information model (CIM) [18] needed in this architecture are the following [19–21]: branch which represents any two-terminal element at the end of which are nodes, node which is the point of connection between two or more branches, root where represents a set of elements on which power calculations are made and, switch which is an electrical component that can break an electrical circuit, interrupting or diverting the current from one conductor to another.

4. PROPOSED SOLUTION DESIGN

Figure 1 shows the application architecture for distributed topology model process of the electricity network in microservices environment. Azure Service Fabric [22] is used for microservice environment. The architecture is designed and implemented by relying on microservice architecture [23] and centralized architecture for messages, having in mind that it is genuinely used in large business applications that require sophisticated control between the transport layer and the service components. The benefits of this topology are advanced
service for topology analysis for each microservice solution. This service is introduced relying on communication foundation (WCF). The address for and the messaging service is done via Windows messaging service. The initialization of internal model is performed in this service, and data is prepared for topology analysis, such as creating IDs based on electricity network on which the client sent a request, via asynchronous call to the service for requests. By inserting this requests from the client and each request is being propagated to the service for requests. Messaging service is a bottleneck communication required to transfer data from the client to service does not perform any orchestration, transformation, error handling, and better overall load balancing and scalability. It uses centralized architecture for message transmission, where one service is declared for message broker that does not perform any orchestration, just to serve for forwarding requests.

4.1. COMPONENTS IN MICROSERVICE AND MONOLITHIC SOLUTION

The application (Fig. 1) consists of the following components: client, messaging service, service for requests and service for topology analysis (TA).

The client specifies which part of the network they are interested in using to calculate the topology of the electricity network. The communication between the client and the messaging service is done via Windows communication foundation (WCF). The address for messaging service on which client should connect is given on the client's side.

Messaging service is the service used only in microservice solution. This service is introduced relying on microservice architecture pattern described above. This service does not perform any orchestration, transformation, calculation or complex routing; it is rather used only for communication required to transfer data from the client to the service for requests. Messaging service is a bottleneck and therefore it is inserted into the architecture so the service for requests would not be overloaded with requests from the client. In other words, messaging service receives requests from the client and each request is being propagated to the service for requests. By inserting this service in the architecture, the application yields better results during topology analysis. Testing has shown that it provides 30 times faster execution, in comparison to the situation in which the messaging service would not exist in the architecture. The communication with the service for requests is asynchronous, that is, each client creates a new asynchronous call to the service for requests.

Service for requests is used in both cases. The service loads an electricity network schema for a given part of electricity network on which the client sent a request, via messaging service. The initialization of internal model is performed in this service, and data is prepared for topology analysis, such as creating IDs based on nodes, creating a model for topology, and creating asynchronous request to the service for topology analysis for each root. The communication between this service and service for topology analysis is asynchronous and it takes place over WCF. The main difference between monolithic and microservices is that, in monolithic it creates one request to services for topology analysis, while on the other hand, in microservice it creates multiple requests.

According to the elements which are needed for internal model, the following XML files are generated: Branches.xml file - includes all branches of electricity network. Each branch has a unique ID with initial and final node and its phase; Nodes.xml file - includes all nodes of the electricity network. Each node has unique node and its own phase; Roots.xml file - includes all roots of the electricity network. Each root has a unique ID, its node, and phase; and, Switches.xml file - includes all switches of electricity network. This scheme is generated via a specific tool - the scheme generator electricity network; thus, it greatly simplifies testing. This tool can modify the following parameters: the number of roots and the number of nodes of electricity network. The above-mentioned XML files are generated based on these parameters.

Service for topology analysis has the same functionality in both solutions, it is based on internal models and on the number of passed roots of the service for requests. It then calls the algorithm for calculating the topology in that way that first the outdated roots will be processed. When a root is calculated for a given topology, the results are returned. The difference is that in monolithic it is one service, while in microservice there are multiple services.

Microservice architecture is implemented with the support of the service fabric reliable service. Reliable services in service fabric provide reliability, availability, consistency and scalability. These services can be stateless and stateful. Stateful reliable services maintain the state of the service. For example: the service that retrieves, processes, and stores information in an external store (such as an Azure blob or table storage) is stateful. It keeps its state in the external state store. A stateless reliable service does not keep the state of service - for example, the calculator. Relying on service fabric, these services are modeled as follows: Messaging service is a stateless service which only forwards the request from the client to the service for requests and, the service for requests is a stateful service. Since stateful service keeps the state, it can be concluded that the service for requests is the most critical service, where the most important parts of the business logic are done. This service keeps the following states: a scheme that is read from the generated XML file and an internal model made based on the scheme which is sent to the service for topology analysis. The service for topology analysis is a stateless service since it is not required to keep the state but only to employ the request algorithm to calculate the topology and return the results back. This service is divided into multiple services (microservices).

Monolithic application is standalone application which does not rely on service fabric.

4.2. APPLICATION DATA FLOW

Figure 2 illustrates the data flow of the architecture explained above. First, the electricity network scheme is required to be generated through specific tools for generating, that was developed for this study. The client, based on the familiarity of the scheme, can choose the root needed to calculate the topology analysis and is sending the
request via WCF to the messaging service. This service receives the request and forwards it to continue to the service for requests, forming an asynchronous call via WCF. As explained above, the only function of this service is to be a mediator. When the request arrives to the service for topology analysis, the XML file previously generated is read with the tool for electric network scheme generation. Based on these, XML files are filled in internal model to get everything ready for the calculation of the topology. Based on this, the tool for generating electricity scheme fills in the internal model, furthermore to send the execution to the service for requests to the messaging service. This service sends the results back to the service for topology analysis. These results are further propagated through the service for requests to the messaging service, and at the end it reaches the clients desired result topology for a given part of the electric network scheme.

The messaging service is introduced during testing to get better results. If the client directly forwarded a request to the Service for requests, this service would be a bottleneck, because all requests from a client would come directly to it. In this way, it would receive all the requirements for each request, to read the electric scheme from XML files and to fill the internal model, furthermore to send the execution to the service for topology analysis. Introducing messaging service for any request forms an asynchronous call to the service for requests to the messaging service. These results are further propagated through the service for requests to the messaging service, and at the end it reaches the clients desired result topology for a given part of the electric network scheme.

As illustrated in Fig. 1, there are multiple services for topology analysis for microservice solution. In monolithic is only one, which calculates topology for all given electricity network. This is the essence of this work because one service is broken into multiple services, or microservices, to independently perform the topology analysis. This number can increase, depending on the configuration of Service Fabric cluster.

4.3. TOPOLOGY ANALYSIS ALGORITHM

The topology analysis algorithm which is used for testing proposed solution is having the task to determine connected parts of elements in a network which enables fast and efficient execution of the power calculations, and to supply each element with necessary information for a client to have an insight into the current state of the network. The topology analysis is conducted for each root separately and only outdated roots are processed. This is significant in the case of the status change of a field equipment, or if a new model is added to the network element. Therefore, it is only necessary to update the topology change of the affected root, which greatly speeds up the process of updating the network graph, as the changes occur only in the root where it exists. The only case where all the roots in the network need to be updated is during the initial analysis of the network topology.

5. EXPERIMENTS

The solution was implemented on Windows Azure service fabric cluster with virtual machines which has the following characteristics: 5 cores, 14GB, 8 data disks, 8x500 Max IOPS and 200GB Local SSD. Five nodes are configured on Service Fabric cluster.

Services for topology analysis are deployed on each node with their primary and replicas, and one instance of messaging service and service for requests with their primary and replicas are deployed on one of five partitions.

If some of the primary services cancels, its replica service takes on the role.

We tested three aspects of the proposed solution, namely: 2, 10 and 20 clients time send the request to calculate the topology analysis of electricity network. As explained above, the changes occur only in the root where it exists. The only case where all the roots in the network need to be updated is during the initial analysis of the network topology.

Each client is separated to one thread, which calls for a given root of electricity network to calculate topology. As explained above, the changes occur only in the root where it exists. The only case where all the roots in the network need to be updated is during the initial analysis of the network topology.

Any combination of the above has been tested 100 times to make the results valid and reliable, and based on these results the average (Avg.) value and deviation (Dev.) are calculated. Avg. shows the mean for all tested results and Dev. shows the degree of scattering of the results of 100 repetitions.
5.1. TWO CLIENTS SEND REQUEST FOR CALCULATING TOPOLOGY ANALYSIS

The first combination is shown in Table 1: two clients are sending request for calculating topology analysis for different types of electricity network with 20, 50, 100 and 200 roots.

### Table 1

<table>
<thead>
<tr>
<th>Number of clients: 2</th>
<th>Number of roots</th>
<th>Microservices</th>
<th>Monolithic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>Avg. [s]</td>
<td>0.126</td>
<td>1.079</td>
<td>5.581</td>
</tr>
<tr>
<td>Dev. [s]</td>
<td>0.144</td>
<td>0.120</td>
<td>0.515</td>
</tr>
<tr>
<td>Avg. [s]</td>
<td>0.339</td>
<td>5.375</td>
<td>42.566</td>
</tr>
<tr>
<td>Dev. [s]</td>
<td>0.019</td>
<td>0.085</td>
<td>0.480</td>
</tr>
</tbody>
</table>

From Table 1 and Fig. 3 it can be concluded that with the increase of the number of roots, electricity scheme is approximately exponentially increasing the execution time of the topology analysis for both microservices and monolithic architecture. To see the results for microservices in Fig. 3 better, the results are multiplied ten times.

Comparing the results of microservices and monolithic application for 20 roots, it can be concluded that the execution for monolithic is 2.6 times longer, while comparing the results for 200 roots the execution for monolithic is 35 times longer.

5.2. TEN CLIENTS SEND REQUEST FOR CALCULATING TOPOLOGY ANALYSIS

In Table 2 is shown the second combination: ten clients are sending request for calculating topology analysis for different types of electricity network with 10, 50, 100 and 200 roots.

### Table 2

<table>
<thead>
<tr>
<th>Number of clients: 10</th>
<th>Number of roots</th>
<th>Microservices</th>
<th>Monolithic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>Avg. [s]</td>
<td>0.03</td>
<td>0.209</td>
<td>1.112</td>
</tr>
<tr>
<td>Dev. [s]</td>
<td>0.001</td>
<td>0.145</td>
<td>0.085</td>
</tr>
<tr>
<td>Avg. [s]</td>
<td>2.036</td>
<td>5.615</td>
<td>28.521</td>
</tr>
<tr>
<td>Dev. [s]</td>
<td>0.181</td>
<td>0.197</td>
<td>0.301</td>
</tr>
</tbody>
</table>

From Table 2 and Fig. 4 it can be concluded that with the increase of the number of roots, execution time for topology analysis is raising up. Also, to see the results for microservices in Fig. 4 better, the results are multiplied ten times.

Analyzing the results of microservices and monolithic application for 100 roots, brings to the conclusion that the execution for monolithic is 25 times longer.

5.3. TWENTY CLIENTS SEND REQUEST FOR CALCULATING TOPOLOGY ANALYSIS

In Table 3 is shown the third combination: twenty clients are sending requests for calculating topology analysis for different types of electricity networks, with 10, 50, 100 and 200 roots.

### Table 3

<table>
<thead>
<tr>
<th>Number of clients: 10</th>
<th>Number of roots</th>
<th>Microservices</th>
<th>Monolithic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>Avg. [s]</td>
<td>0.028</td>
<td>0.162</td>
<td>0.982</td>
</tr>
<tr>
<td>Dev. [s]</td>
<td>0.001</td>
<td>0.009</td>
<td>0.208</td>
</tr>
<tr>
<td>Avg. [s]</td>
<td>7.399</td>
<td>10.445</td>
<td>33.255</td>
</tr>
<tr>
<td>Dev. [s]</td>
<td>0.076</td>
<td>0.079</td>
<td>0.831</td>
</tr>
</tbody>
</table>

From Table 3, similarly as in Table 1 and Table 2, it can be concluded that with the increase of the number of roots, the execution time for topology analysis is raising up. Once again, as in Fig. 3 and Fig. 4 in order to see the results for microservices in Fig. 5 better, the results are multiplied ten times. Analyzing results for 50 roots shows that the duration for monolithic takes 64 times more time than for microservices.

5.4. COMPARING RESULTS

Based on these results, it can be observed that as the number of clients increases, the execution of the topology analysis lasts longer (Fig. 6). Of course, for larger electricity network schemes, with more roots, the execution of the topology analysis takes longer than for electricity network with less roots. Also, it can be concluded that, for a given model of the electricity network scheme, increasing the number of clients leads to performance mean value exponential growth. For example, by comparing the results of 10 and 20 clients, for 100 roots, it can be concluded that
the results for 20 clients show 24 % more time used than for 10 clients. This is expected due to the test scenario, having in mind that for 20 clients there are 20 threads, which execute topologies, while in the first case for 10 clients there are 10 threads.

All these threads are sending requests on 5 instances of the service for topology analysis, on 5 different partitions. Increasing the number of clients (number of threads), prolongs the execution of topology in this test scenario.

Comparing the results of monolithic application and microservices brings to the conclusion that the increase of the number of roots leads to even greater difference in performance between microservices and monolithic application. Analyzing Figs. 3, 4 and 5 with the increase of the number of roots, the difference between monolithic application and microservices is larger.

6. CONCLUSIONS

In this paper, we describe the application of the architecture for topology model analysis of a DMS using microservice architecture and Windows Azure Service Fabric, in order to show how dividing one service into multiple microservices gives a more flexible architecture, and complete separation of one of the microservices from the other microservices, which can perform a completely independent function. For the purpose of testing this architecture, topology analysis of the electricity network is used.

This application architecture gives superior results than monolithic applications, since one service for topology analysis scales to multiple instances of the service for topology analysis, and the execution of the topology works in parallel; each root is processed independently.

Performance results for microservices shown in the previous section are very good in comparison to monolithic. Further the acceleration of execution of topology analysis, it is seen that having a service being divided into multiple microservices contributes to accelerating calculations of DMS functions. It also benefits the acceleration of obtaining results, such as phase, power, energizing, belonging to the elements, circuit hierarchy and others.

The advantages of using microservices are numerous: the ability to respond quickly to a constantly changing environment; easy to deploy each service; ease of testing due to the separation and isolation of business functionality into independent applications, performance, each service can be scaled, and, development becomes easier due to the smaller and isolated scope.

Received on July 22, 2017

REFERENCES

23. M. Richards, Software Architecture Patterns, O'Reilly, 2015.