Automated Scalability System for Distributed Applications in Heterogeneous Environment

Elizaveta Kuzenkova[0000−0002−4490−4023], Iurii Korenkov[0000−0003−0373−669X], Ivan Loginov[0000−0002−6254−6098], Aglaya Ilina[0000−0003−1866−7914], and Andrey Dergachev[0000−0002−1754−7120]

ITMO University, 49 Kronverksky Pr., 197101, St. Petersburg, Russia,
e1izabeth.k@outlook.com, ged.yuko@gmail.com, ivan.p.loginov@gmail.com,
agilina@itmo.ru, amd@itmo.ru

Abstract. The project presented describes an approach for horizontal scaling management system for applications in heterogeneous environment. It incorporates interconnected supervisor services for end-user devices, making it possible to dynamically distribute utilization of available computation resources by the particular applications. Horizontal scaling achieved by connecting lots of end-user devices having supervisor services that allows resource management. Load balancing represented if form of application component migration between different compute nodes. Dynamic migration of components of running application based on shared registry of RPC endpoints and transparent routing of RPC interconnections between application components. It is also possible to not freeze the whole application during migration operations due to the way, how it is treated by the proposed system. Application component temporarily suspended during the migration of an application component and then resumed when migration completed. Described way of RPC organization allows to make parts of application independent in a way that component pausing should not cause the entire application to stop. Brief description of contract between management system and application is presented, as well as description of the migration procedure. Proposed solution can be applied to scale applications targeted for end-user devices, that can't be scaled using traditional server-side technologies of clusterization or can be scaled only using specific technical skills, that are not mastered by non-technician user.

Keywords: Scalability · Heterogeneous environment · Distributed application · Clusterization · Modularity · Live migration

1 Introduction

1.1 Problem

The problem of auto-scalability is very actual nowadays. There are more and more different devices that can be connected by the network and lots of resource-intensive applications. So, we have heterogeneous environment of end-user devices and applications without clusterization technologies in user-friendly way.

1.2 Known popular solutions

There are many solutions, but most of them limited by platform fragmentation and heterogeneous environment, that leads to limitations scalability applications. Some solutions could be used in heterogeneous environment, but they are limited by special cases typically. They don't aim for end-user. There is no common possibility to get advantages of scalability and clusterization for the end-user applications, that run on end-user devices. There is no well-known methodology of how to support it transparently in the software and no user-friendly ways of dynamic clusterization functionality exposure. But let's consider some of similar solutions.

QNX is a real-time operating system that allows you to effectively organize distributed computing by combining the entire network into a single homogeneous set of resources. This operating system is well suited for building distributed systems. However, distributed computing can only be performed in a homogeneous environment. In other words, it is required that all your devices run the QNX operating system. In addition, if you want to run your application on this system, you will need to finalize or completely rework the existing solution.

Docker is an open-source platform for running and shipping applications using relatively lightweight isolated environments – containers. Containers run simultaneously on a given host machine's kernel and often managed by Swarm. It's well suited for continuous integration and continuous delivery workflows, high load and working with heterogeneous environment. But we cannot move a running Docker container from one host to another. Scaling and managing workflows possible just in near real time. Now there are many orchestrators and containers managers for Docker, but they still don't allow ship application or its part in real time.

Kubernetes is open source software for automating the deployment, scaling and management of containerized applications. It works in heterogeneous environments and helps to manage balancing load and auto-scalability. Kubernetes deploys containers based on OS-level virtualization instead of hardware virtualization. But it still doesn't bring us non-stop working.

Mesos does the same as Kubernetes, but it has directly different principle of working. It chose from offered existing resources most suitable and then runs task on chosen agent. When task ends the same retried again. Mesos support

auto scaling of cluster and heterogeneous resources with the Dominant Resource Fairness algorithm (DRF) as default resource allocation policy.

All of them designed to run in mostly trusted environment with explicit administration procedures, completely excluding them from being applied on the end-user side of a typical application.

2 Project description

We can't name even one known system that has all the listed characteristics such as scaling in a heterogeneous environment, non-stop working of an application during the migration of its component, orientation to end-user, clear and simple solution using. Well-known similar solutions either aimed to homogeneous environment are unable to sustain working application during scaling management operations and can't be applied in particular software, when that software does not have builtin clusterization technologies. There are Kubernetes, Mesos, Docker, QNX – solutions that are related to different classes (of destination), but with similar features in the scalability context. This project presents an approach that based on this functionality and makes an accent on application in heterogeneous software or hardware environments.

2.1 Aspects of research

To solve these problems, the principles of dynamic clustering and accessibility management of distributed resources were investigated. As a result, the goal was determined to create a prototype system that can automatically control the distribution of application components without pausing it in a heterogeneous environment. Desired system should also have enough security to provide safety data transferring.

To ensure safety of application component live migration to another computational resource possible attack vectors have to be addressed. The essential security requirements listed as follows:

- Appropriate access control. The source and destination hosts should mutually authenticate each other.
- Confidentiality. The contents of data passed between computational nodes under the authority of the user should be protected from the second-person inspection, so that no one can read or misuse an information about user's applications or tasks executed on top of distributed infrastructure.
- Infrastructure integrity. No one could interfere with dynamic resource managment coordination and distributed state of the application managed by the system, so it should be consistent at any moment.
- Availability. No one should be able to disrupt the services provided by distributed application, except infrastructure fragmentation.

This goal was decomposed into a few tasks:

- Implement software prototypes with listed functionality as follows:
  - dynamic resources management from the point of their usage by parts of the distributed application;
  - fog-like computation environment integrity;
  - relocatable resources accessibility support;
  - essential security requirements should be considered.
- Design and research software infrastructure that allows dynamic transfer of an application parts between compute nodes in a heterogeneous environment.

2.2 Solution overview

Presented software system was designed in conjunction with research tasks. The architecture based on the interaction of services. These services located on the particular user devices, which can have different hardware architecture and capabilities. This provides an opportunity of working in unified heterogeneous environment.

Given services collect information about available resources and parameters of user's devices at each moment of time. They monitor changes in utilization of pro-cessors, memory and network. Services can interchange this information with each other and interact, distributing components of running application to perform load balancing. Balancing operations guided by an information about utilization require-ments and thresholds, so that when they met, it leads to the decision about particular application component migration from one device to another, keeping the whole application state intact.

In order to specify and avoid situations when migration of the component of run-ning application is impossible or unwanted, services also should be configured and keep information about quotas of resources dedicated for user's tasks and accom-panying constraints.

The application can be considered as running in the managed environment and consisting of modules (so-called Application Domains - AppDomains). Modules are interacting parts of an application, that are the components mentioned above. The application can be migrated from one user's device to another, connected by the network, by services that supervise and manage different computational nodes. For example, on the Fig. 1 there are green modules of one application, and blue modules of another.

An application integrity reached by remote procedure call (RPC) abstraction. Supervisor services maintain safety connections between modules and route their communications transparently for the RPC layer. So it does not matter, on which device the particular module of the application is running now. It is important, because RPC end-points are not bound to devices due to migrations. Thus, all devices are sharing their resources between each other and form a self-organizing peer-to-peer overlay network. Information about RPC end-points is available through the distributed registry.

It becomes possible to not freeze the whole distributed application during its modules migration, because of the connection information availability between modules independently on their states.

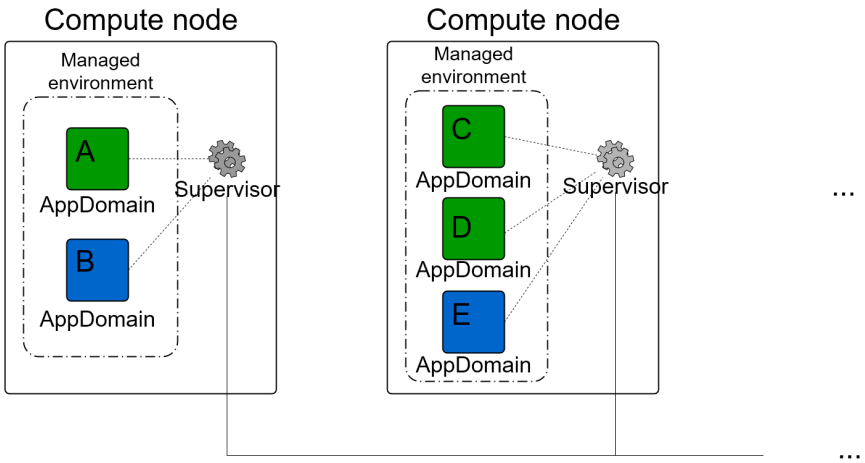

Fig. 1. Applications distributed across different computing nodes

In fact, the main subject of this research is management and migration of application's modules. It is required to be able to get meta-information about application module to manage the state of particular application domain and its RPC end points, used for inter-domain communications. The CLR runtime, chosen for our implementation, presents such capabilities by rich reflection services and builtin application domains functionality.

All of this leads to the only one limitation. The application should consist of more than one module. If the application consists of only one module, it can't be distributed.

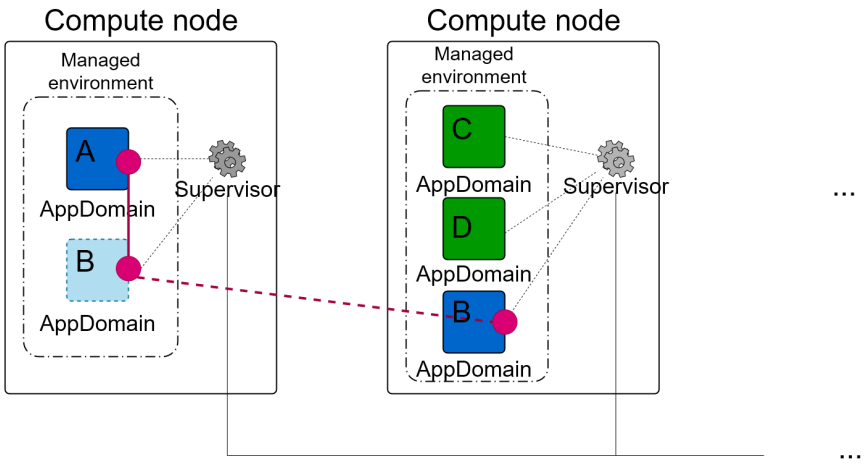

Fig. 2. The process of AppDomain migration

The information required to connect application to the service is stored in an environment variable. A communication channel creates between each application domain and the service under discussion during application domain initialization. Each AppDomain has a specific contract implementation. This contract allows the application and service to interact and makes possible all of the functionality described above. The following code is an example of the programming interface for managing an application module.

```
interface IAppManager
{
        void Start (IRpcHost);
        AppState Freeze();
        void Restore(AppState, IRpcHost);
}
```

This contract allows the service to manage the lifecycle of each module of distributed application. Any AppDomain can be migrated from one compute node to another until it has bounded resources, such as specific device requirements. To accomplish this, the execution of the particular AppDomain should be frozen, then its state should be serialized and transferred, and then it should be resumed on the other device.

The supervising service collects information about other available devices and monitors performance indications during the execution of the application of the compute node, where it is running, as well as other available compute nodes. If there are no restrictions about application state or environment state and it is appropriate to transfer particular application module, then the supervising service can initiate the migration procedure.

Here's how the state of an application can be seen in conjunction with migration procedures:

- Initialization of application
- Initialization of application modules
- Application module hibernation (during transportation)
- Restoration of the application module
- Completion of the application module
- Shutdown the application

Process of application instance initialization looks like following sequence of operations:

- RPC endpoint information retrieving from the environment variable
- Establishing RPC channel connection
- Selection of startup mode
- Registration of new application instance or restore of the module of an existing application instance

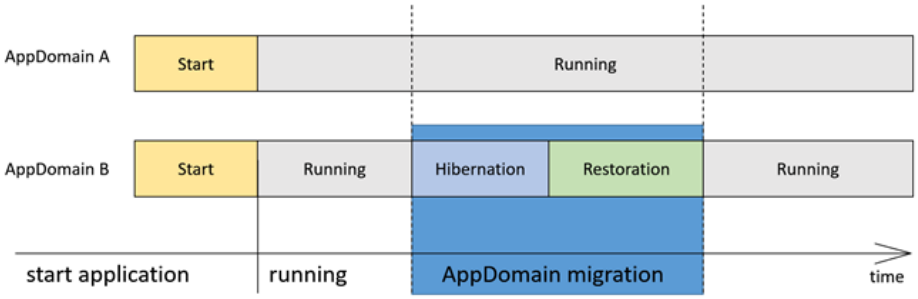

Fig. 3. Migrated AppDomain states

3 Conclusion

Proposed solution allows to create "seamless" workspaces, where the boundary between different computing devices can be eliminated, if they connected to the same network, thereby potentially increases the productivity of scientific and technical work.

It removes obstacles about extra efforts and pains required for the interaction of user with the applications deployed on the different devices, connected by the network, when such applications are not designed to be accessible remotely.

It is the first steps to ubiquitous computing - a model of human interaction with a computing system in which the user is surrounded by computing devices permeating the environment, integrated into everyday things.

The software system architecture was designed with the contract between managing service and manageable application module. The logic of mentioned service was formulated.

Further research planned on implicit implementation of AppDomain state serialization and migration. It should decrease the complexity of proposed technology integration for the applications. These plans include prototyping and testing in the emulated network environments with various limitations on bandwidth and latency.

Further areas of application of the proposed solution are Internet of Things domain and fog computing. The nature of the computer systems that used in this areas supposes limited computational resources from the one side, and the requirement of reliability and small time to get the results. Such aspects lead to adaptivity of software system as possible answer to needs of integrated end-user application development.

References

1. lorian Katenbrink, Ludwig Mittermeier, Andreas Seitz, Harald Mueller, and Bernd Bruegge. Dynamic Scheduling for Seamless Computing. In 2018 IEEE 8th Inter-

national Symposium on Cloud and Service Computing (SC2), pages 41–48, Nov 2018.
2. Tanenbaum, Andrew S., and Herbert Bos. "Modern operating systems.", Pearson, (2015).
3. Richter, J. "CLR via C#. Programming in .NET Framework 4.5 platform on in C#", Saint-Petersburg: Piter. (2013).
4. Hildebrand, Dan. "An Architectural Overview of QNX." USENIX Workshop on Microkernels and Other Kernel Architectures. (1992).
5. Sam Newman. Building Microservices. O'Reilly Media, Inc., 1st edition, (2015).
6. J. Oueis, E. C. Strinati, and S. Barbarossa. The Fog Balancing: Load Distribution for Small Cell Cloud Computing. In IEEE 1st Vehicular Technology Conference (VTC Spring), 2015.
7. Florian Katenbrink, Ludwig Mittermeier, Andreas Seitz, Harald Mueller, and Bernd Bruegge. Dynamic Scheduling for Seamless Computing. In 2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2), pages 41–48, Nov 2018.
8. Eva Marín-Tordera, Xavier Masip-Bruin, Jordi Garcia Almiñana, Admela Jukan, Guang-Jie Ren, Jiafeng Zhu, and Josep Farre. What is a Fog Node A Tutorial on Current Concepts towards a Common Definition. CoRR, 2016
9. Harald Mueller, Spyridon V. Gogouvitis, Houssam Haitof, Andreas Seitz, and Bernd Bruegge. Poster Abstract: Continuous Computing from Cloud to Edge. In IEEE/ACM Symposium on Edge Computing (SEC), pages 97–98, 2016.
10. Kakadia, Dharmesh. Apache Mesos Essentials. Packt Publishing Ltd, 2015.
11. Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. Linux journal, 2014(239), 2.
12. Buchanan, Steve, Janaka Rangama, and Ned Bellavance. "Inside Kubernetes." Introducing Azure Kubernetes Service. Apress, Berkeley, CA, 2020. 35-50.