Mjolnirr: A Hybrid Approach to Distributed Computing
Architecture and Implementation

Dmitry Savchenko and Gleb Radchenko
System Programming Department, South Ural State University, Chelyabinsk, Russia

Keywords: Cloud Computing, Cloud Middleware, PaaS, Private Cloud, Mjolnirr.

Abstract: A private PaaS enables enterprise developers to leverage all the benefits of a public PaaS to deploy, manage, and monitor applications, while meeting the security and privacy requirements your enterprise demands. In this paper we propose the design and implementation of a Mjolnirr private cloud platform for development of the private PaaS cloud infrastructure. It provides means for custom applications development which uses resources of distributed computing environment.

1 INTRODUCTION

Cloud computing became very popular during the last few decades (Mell and Grance, 2011). This type of resource provisioning offers convenient access to computing resources, which makes it easy to use them for custom services development.

Before the introduction of the cloud computing concept, there was another distributed computing concept called "grid computing" (Foster and Kesselman, 2003). It is used by the scientific community while solving extra-large and resource-intensive scientific tasks. The name "grid" originated from the metaphor of the power grid. However, unlike electricity, integration of new resources to the grid computing environment is not trivial.

Meanwhile, PaaS (Platform as a Service) solutions provide the ability to create custom cloud applications much easier. But the problem is that the most PaaS solutions are deployed on remote hosting company servers. The owner of the application may not know exactly where his information is stored, as well as he cannot be absolutely sure about the safety and security of the information. These issues might be solved by the private PaaS platform (Fortis, Munteanu and Negru, 2012).

In this paper, we propose the design and implementation of the private cloud platform called "Mjolnirr". It provides a service which is comparable with existing cloud platforms. There are several solutions that provide private PaaS (Yandex Cocaine, AppFog, Stackato) levels of service. Comparing to these solutions, main features of our approach are integrated messaging subsystem, flexible workload management and UNICORE (Streit, 2009) grid environment integration module.

The article contains the following sections. In section 2 we present the main goals of the Mjolnirr cloud platform development. In section 3 we describe the results of the analysis of existing cloud platforms and compare them with our platform. In section 4 we describe the structure of the Mjolnirr platform and the component communication protocol. In section 5 we describe the implementation of the Mjolnirr platform. In section 6 we describe the platform performance evaluation. In section 7 we summarize the results of our research and further research directions.

2 MJOLNIRR PLATFORM CONCEPT

The main goal of the project is to create the cloud platform for development of a private PaaS cloud infrastructure. *Mjölnir* is the name of the hammer of Thor, a major Norse god associated with thunder. Therefore, we have chosen the "Mjolnirr" name as metaphor of a powerful tool linked with thunder and clouds.

Java-based application or library can be implemented as a Mjolnirr-based service. The Mjolnirr platform provides infrastructure for cloud

Savchenko D. and Radchenko G..
Mjolnirr: A Hybrid Approach to Distributed Computing - Architecture and Implementation.
DOI: 10.5220/0004958704450450
In *Proceedings of the 4th International Conference on Cloud Computing and Services Science* (CLOSER-2014), pages 445-450
ISBN: 978-989-758-019-2

applications development, including software developer kit, message brokering system and browser support. For a developer, a Mjolnirr application is represented as a collection of independent components communicating by message passing. This approach allows to develop flexible and scalable cloud applications.

Mjolnirr also provides integration with the UNICORE grid environment (Streit, 2009) through the DiVTB (Radchenko and Hudyakova, 2013) platform. The DiVTB (Distributed Virtual Test Bed) platform provides a task-oriented approach for solving specific classes of problems in computer-aided engineering through resources supplied by grid computing environments. Thus, Mjolnirr can be used both to provide infrastructure for scientific projects with the grid systems and in a business infrastructure.

During our research, we should:

1. analyse technologies of private cloud platforms development;
2. develop Mjolnirr platform architecture;
3. implement Mjolnirr platform;
4. evaluate the performance and scalability of the Mjolnirr platform.

3 ANALYSIS OF EXISTING SOLUTIONS

An investigation shows that C-level executives and IT managers in enterprise companies have concerns with integration of cloud computing in their data processing (Avanade.com, 2014). One of the most serious concerns is the possibility of data breaches. A cloud provider can give access to the company's private data (accidentally or intentionally) or may bring harm to the data owner.

It is possible to use the encryption of data stored in the cloud, but this is effective only when cloud is used only for storage. If the data is processed in the cloud, it become available decrypted in the memory of the host, where the processing occurs. In addition to this drawback, the owner of the data does not control the location of his virtual machine, so it can be moved to the physical computer with the virtual machine that contains malware. It means that they will have the same IP address. It may cause the block of the virtual machine or forfeiture of computer containing these virtual machines.

Nowadays, there are two ways to ensure the data security in the cloud. The first way is called "trusted computing". It ensures security of virtual machines in the cloud (Christodorescu et al., 2009). But user data cannot be completely safe. In IaaS clouds, the virtual machine can be moved to the other host, but the concept of trusted computing provides security only for virtual machines running on the same host. Otherwise, the concept of Trusted Cloud Computing Platform (Garfinkel et al., 2003) solves this problem by creating the safe environment for running virtual machines. But neither of these approaches solves the problem of VMs placement on the same host with a malicious VM.

Another way to deal with the security issue is to deploy the cloud infrastructure on the private hardware. But buying and maintaining of the hardware is more expensive than rent of computing resources.

The simplest way is to create a private cloud system, and there are several different platform (PaaS) solutions existing in this area.

Yandex Cocaine and AppFog platforms provide the ability to create private PaaS solutions based on application containers (Api.yandex.com, 2014; AppFog, 2013). These platforms allow creation Heroku-like application hosting. They provide a number of built-in modules and a server infrastructure. Stackato (Activestate.com, 2014) shows all advantages of the already mentioned solutions and provide local application store. However, all of the above solutions consider custom applications as a monolith and ignore its internal structure, that's why it is not possible to automatically and effectively balance the load on the individual subsystems applications. In addition to this drawback, none of these solutions considers end-user workstations as computing resources providers.

We decided to create a Mjolnirr cloud platform, which solves these problems as follows:

1. *Cost reduction* will be provided by using popular open source libraries and programming languages, as well as the opportunity to work not only on the server platform, but also on the personal computers, using idling resources of the computer system.
2. *Ease of application development* will be provided by using popular programming languages, developer tools and SDK.
3. *Integration of new resources ease* will be provided by the application architecture modularity and custom components reusability.

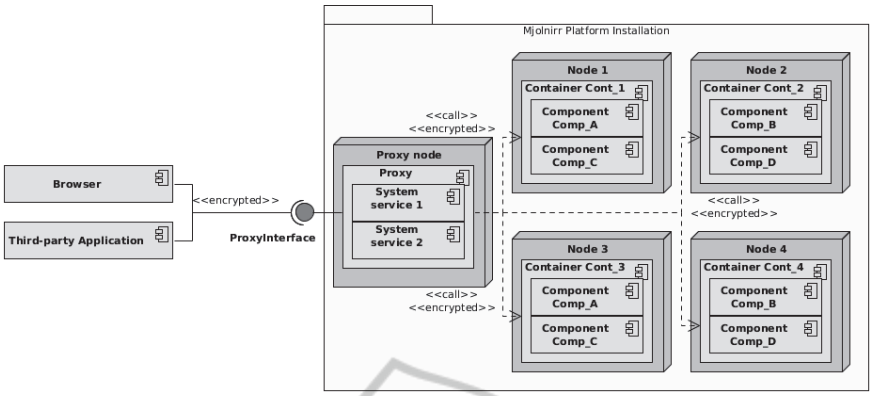

Figure 1: Mjolnirr platform architecture.

4 MJOLNIRR ARCHITECTURE

Mjolnirr platform architecture is shown at the figure 1.

The Mjolnirr platform includes the following components:

1. *Proxy* provides access to the cloud system for the external clients and manages the communication between cloud application components. It also hosts all of the system services (built-in modules for user authentication, distributed file system, database access etc.). Proxy is the only component that accessible from the external network.
2. *Container* is responsible for hosting of cloud applications components and message transmission. It can be deployed both on personal computers and on the computing server nodes.
3. *Components* are custom applications, developed to run in Mjolnirr cloud environment. Each component has a unique name. UI components (applications) are multi-page applications.
4. *Clients*. All client applications use encrypted channel to communicate with the proxy. Each client should use certificate for authentication.

4.1 Proxy

The Proxy (see fig. 2) provides access to Mjolnirr system from the external network. The Proxy performs the following actions:

1) it stores and provides static resources (page layout descriptions, images etc.) of the deployed cloud applications in the *Static Storage*;
2) it provides a *Component Repository* to all of the containers;
3) it provides a *Message Server* for the components of cloud applications;
4) it handles client's requests to the *Client Interface*;
5) it performs the authorization and authentication of users;
6) it hosts all of the system services (users management, distributed file system, database access, etc.). System services act like a custom component and can be called in a standard way (see section 4.3).

The external Proxy interface exposes the following RPC methods:

1) *String getUI (component, page)* - returns the layout description of the page *"page"* of the component *"component"* as a plain text;
2) *byte[] getResourceFile (applicationName, resourceName)* - return the required resource file (image, script file) in binary format;
3) *String processRequest (componentName, methodName, args)* processes a client request, redirecting it to the first free suitable component instance. This method can be called directly from the application page, from script scope. Returns serialized plain object as an execution result;
4) *void uploadComponent (content)* – upload the custom component to the proxy. Component will be added to the *Component Repository* and immediately distributed to the running containers.

4.2 Container

The container provides cloud application component hosting and API for remote components instances method invocation. The Mjolnirr installation can have any number of containers.

Any Mjolnirr-based application consists of independent components, which use a built-in messaging system, implemented in the basis of the Publisher-Subscriber pattern. The Proxy is responsible for message queue maintenance. The Message Server of the Proxy provides publisher-subscriber messaging service for cloud application components. Mjolnirr Containers subscribe to Message Channels that operate as a broadcast delivery – any message sent to the Message Channel will be transmitted to the subscribers of this channel. Each cloud application instance is subscribed on two types of Message Channels:

1) Component Public Channel: every instance of the cloud application component is subscribed on this public channel. This a listener channel – when any message come to this channel, the appropriate component instance will be invoked.
2) Instance Private Channel: provides direct communication between instances.

When container starts, it performs several initialization steps. The order of the container initialization:

1) Container registers on the proxy and receives a list of components to load in response;
2) For each component from the list:
   a. The container checks it's local cache, for each missing package container and downloads it from the proxy's Component Repository;
   b. Container initializes the component instance;
   c. Container subscribes the component instance on the Component Public Chanel and Instance Private Channel.

Container provides messaging API to all the hosted component instances. Typical message transmission sequence looks like shown below:

1) When a component instance calls another component, it sends the call to a Component Public Channel.
2) The first available instance of the component in the cloud system processes the request.
3) The response is returned to an Instance Private Channel of the instance that sent a message.

In addition, a container has an opportunity to work in stand-alone mode. In this mode, the container does not support communication with other containers and acts as a stand-alone computing system (container and proxy at the same time).

4.3 Components

From the developer's perspective, Mjolnirr cloud application is a collection of independent components communicating by message exchange. Components are represented as a package that contains the following information:

1) manifest, that provides the interface of the component, including description of provided methods and their parameters;
2) executable to handle incoming requests;
3) static files, used in pages rendering (images, page layout descriptions and scripts) for UI provision.

Each component can be:

1) *Application Component*: provides the user interface definition, scripts, styles and UI-specific actions. Optionally contains domain logic.
2) *Module Component*: represents a single entity in the domain logic of the application. The Module Component provides data processing and storage, but doesn't provide interface and static files.

4.4 Clients

Mjonirr platform uses HTML and JavaScript to represent end-user interface in common internet browser. But it's possible to use Mjolnirr resources in third-party application using commonly used and standardized protocols.

We've decided to use most common client software – web browser – as a platform default client. Therefore, there is no need to install specialized client software on the client PCs, and it gives the opportunity to use thin clients.

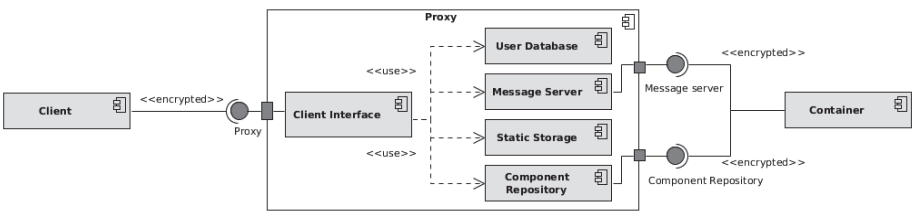

Figure 2: Mjolnirr proxy architecture.

5 IMPLEMENTATION

5.1 Components

Each custom component must have a manifest: an XML file that contains the component's name, the name of the main class, list of methods and a list of parameters for each method. Manifest can be generated automatically using special Maven plugin. Sample manifest for the Calculator service is shown below.

```
<hive>
<application
  name="Calculator"
  classname="org.exmpl.Calculator">
  <methods>
   <method
     name="calculate"
     type="java.lang.String">
    <parameters>
     <parameter
       type="java.lang.String"/>
    </parameters>
   </method>
  </methods>
</application>
</hive>
```

The container parses the manifest for each loaded component.

As stated above, the manifest must have the name of the main class as a fully qualified name of the facade class of described components. Each facade class must have an implementation of the method "*initialize*". The container calls this method immediately after instantiating the component and passes the component *context there*. This context contains information about the working directory of the current application instance and the directory containing the configuration files.

5.2 Messaging Subsystem Implementation

Mjolnirr platform messaging subsystem is implemented on the basis of the queue system called HornetQ (HornetQ project homepage, 2014), which works in accordance with the Publisher-Subscriber pattern. HornetQ can work in embedded mode, it is open source and it provides high performance (Spec.org, 2014). HornetQ server is built into the Proxy. The container provides messaging API to the components. API contains synchronous and asynchronous methods for sending messages.

5.3 UNICORE Integration

The Mjolnirr platform provides a UNICORE 6 integration module. This module uses DiVTB Server for communications with grid environment.

The concept of Distributed Virtual Test Bed (DiVTB) (Radchenko and Hudyakova, 2013) provides distributions of supercomputing simulation in two phases – development of *experiment* and launching *the test stand*. The main advantage of the DiVTB concept is provision of problem-oriented interface to the supercomputer resources.

Mjolnirr UNICORE integration module provides the following methods:

1. *void uploadTestBed (content)* – used for test bed archive upload;
2. *String createExperiment (bedID)* – create experiment from test bed with ID *"bedID"*, returns experiment UID;
3. *void startExperiment (experimentID)* – start experiment with specific parameters;
4. *int getStatus (experimentID)* – get experiment status (started, finished or failed);
5. *byte[] getResults (experimentID)* – get experiment results.

6 PERFORMANCE EVALUATION

Mjolnirr Platform was tested in two different experiments on the node with Intel Core i5 M560 and 6 GB of RAM.

6.1 Parallel Execution

We researched the behaviour of a platform during transferring a large number of concurrent messages. The results are shown on the figure 3, left.

6.2 PI Calculation

We also researched the behaviour of the platform during the long calculations. This experiments were conducted on 8 containers running for the N simultaneous requests for N=1..24. The results are shown on figure 3, right.

Experiments have shown that the platform is stable and able to pass the average number of messages; additional transmission costs are about 25 milliseconds per request.

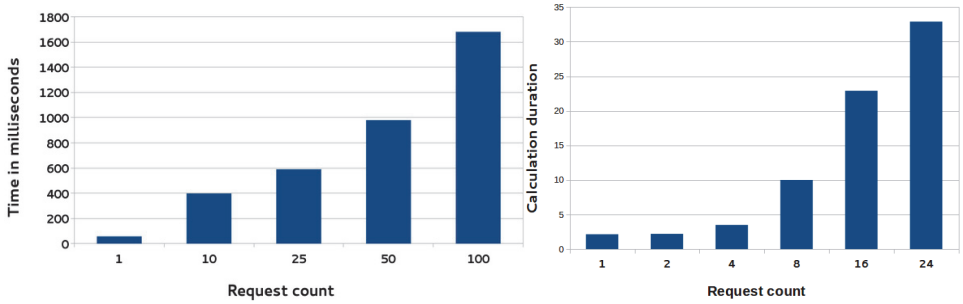

Figure 3: Performance evaluation.

7 CONCLUSIONS

In this article, we described the design and implementation of a private cloud platform called Mjolnirr, which allows development of distributed cloud applications on private computing resources. Main features of the described platform are advanced messaging system and distributed computing support.

As a further development, we will investigate and implement application-level migration support, integration with the advanced resource monitoring systems, flexible adaptation to load changes, advanced system security and application store. The application store will reduce the number of duplicate software products and simplify the creation of individual business infrastructure to meet the needs of a particular company. Also, we will implement a set of standard system services (unified database access, advanced users management and distributed file system access) and improve system security, providing isolation of custom components from the containers to protect the system from accidental or intentional denial of service attacks.

ACKNOWLEDGEMENTS

The reported study was partially supported by Grant Fund for Assistance to Small Innovative Enterprises in Science and Technology research project No. №0000829 and by RFBR, research project No. 14-07-00420-a.

REFERENCES

Activestate.com. 2014. *Stackato: The Platform for the Agile Enterprise | ActiveState*. [online] Available at: http://www.activestate.com/stackato [Accessed: 11 Jan 2014].

Api.yandex.com. 2014. *Cocaine. Cocaine technology description*. [online] Available at: http://api.yandex.com/cocaine/ [Accessed: 11 Jan 2014].

Appfog.com. 2014. *AppFog - PaaS for public and private clouds*. [online] Available at: https://www.appfog.com/ [Accessed: 11 Jan 2014].

Avanade.com. 2014. *Global Study: Cloud computing provides real business benefit | Avanade*. [online] Available at: http://www.avanade.com/us/about/avanade-news/press-releases/Pages/Global-Study-Cloud-Computing-Provides-Real-Business-Benefits-But-Fear-of-Security-and-Control-Slowing-Adoption-page.aspx [Accessed: 11 Jan 2014].

Christodorescu, M., Sailer, R., Schales, D. L., Sg, Urra, D. and Zamboni, D. 2009. Cloud security is not (just) virtualization security: a short paper. pp. 97--102.

Foster, I. and Kesselman, C. 2003. *The Grid 2*. Burlington: Elsevier.

Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M. and Boneh, D. 2003. Terra: A virtual machine-based platform for trusted computing. 37 (5), pp. 193--206.

Mell, P. and Grance, T. 2011. The NIST definition of cloud computing (draft). *NIST special publication*, 800 (145), p. 7.

Peng, J., Zhang, X., Lei, Z., Zhang, B., Zhang, W. and Li, Q. 2009. Comparison of several cloud computing platforms. pp. 23--27.

Pepple, K. 2011. *Deploying OpenStack*. O'Reilly Media.

Radchenko, G. and Hudyakova, E. 2013. Distributed Virtual Test Bed: an Approach to Integration of CAE Systems in UNICORE Grid Environment. *MIPRO 2013 Proceedings of the 36th International Convention*, pp. 183-188.

Santos, N., Gummadi, K. P. and Rodrigues, R. 2009. Towards trusted cloud computing. pp. 3--3.

Streit, A. 2009. UNICORE: Getting to the heart of Grid technologies. *eStrategies*, 3 pp. 8-9.

Spec.org. 2014. *All Published SPEC SPECjms2007 Results*. [online] Available at: http://www.spec.org/jms2007/results/jms2007.html [Accessed: 12 Jan 2014].

HornetQ project page. 2014. *HornetQ – putting buzz in messaging*. [online] Available at: http://www.jboss.org/hornetq [Accessed 12 Jan 2014]