Mjolnirr: private PaaS as distributed computing evolution

D.I. Savchenko* and G.I. Radchenko*
*South Ural State University, Chelyabinsk, Russia
dmitry@mjolnirr.com, radchenkog@susu.ac.ru

Abstract - Cloud computing became very popular during the last few decades. It provides convenient access to remote computing resources for individual users and organizations. However, there are still security issues arise if the private data is transmitted to the public cloud for processing. This issue can be resolved with private cloud systems. In this paper, we propose the design and implementation of a Mjolnirr private cloud platform for development of the private PaaS cloud systems. It provides infrastructure for cloud applications development, including software developer kit and message brokering system. Proposed cloud platform can be deployed on resources of a distributed computing system, including idling resources of personal computers. For a developer, a Mjolnirr application is represented as a collection of independent components communicating by a message exchange communication protocol.

Keywords - PaaS, Mjolnirr, Cloud computing, distributed system, message passing

I. INTRODUCTION

Cloud computing became very popular computing paradigm during the last decade [15], [25]. This approach to 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" [11]. It is still used by the scientific community to solve extra-large and resource-intensive scientific tasks. The name "grid" originated from the metaphor of the power grid. However, unlike electricity devices, integration of new resources to the grid-computing environment is not trivial.

Meanwhile, PaaS (Platform as a Service) [9] solutions provide the ability to create custom applications [6]. However, the problem is that the most PaaS solutions are deployed on remote hosting platforms. The owner of the application may not know exactly where his information is stored, as well as he cannot be sure about the safety and security of his information [20]. These issues might be solved by the private PaaS platform [10].

In this paper, we propose the design and implementation of the private cloud platform called “Mjolnirr”. There are several solutions available now that provide private IaaS (VMWare Private Cloud, OpenStack) and 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 native UNICORE [24] grid environment integration module.

This paper is organized as follows. In section II we present the concept of the Mjolnirr cloud platform. In section III we describe the results of the analysis of existing cloud platforms and compare them with Mjolnirr platform. In section IV we describe the architecture of the Mjolnirr platform and the component communication protocol. In section V we describe the implementation of the Mjolnirr platform and demonstrate the process of custom applications development. In section VI we describe the platform performance evaluation. In section VII we summarize the results of our research and further research directions.

II. MJOLNIRR PLATFORM CONCEPT

The Mjolnirr private cloud platform provides development of a Java-based private PaaS cloud infrastructure. A Java application or library can be implemented as a Mjolnirr-based service. The Mjolnirr platform provides infrastructure for cloud applications development, including software developer kit, message-brokering system. For a developer, an Mjolnirr application is represented as a collection of independent components communicating by message exchange. This approach allows development of flexible and scalable cloud applications.

Mjolnirr also provide integration with the UNICORE grid environment through the DiVTB [18] 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 to provide infrastructure both for scientific projects with the grid systems and in a business infrastructure.

During our research, we should:

• analyze technologies of cloud systems and private cloud platforms development;
• develop Mjolnirr platform architecture;
• implement Mjolnirr platform;
• evaluate the performance and scalability of the Mjolnirr platform.

III. ANALYSIS OF EXISTING SOLUTIONS

An investigation shows that C-level executives and IT managers in enterprise companies have issues with integration of cloud computing in their data processing [5]. 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 [14], [21].

For the prevention of such privacy threats, it is possible to use the encryption of data that is stored in the cloud [19], [22]. However, this approach is effective only when cloud is used just for storage. If the data is processed in the cloud, it becomes available decrypted in the memory of the host, where the processing occurs [4]. 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 may cause the ban of the virtual machine or forfeiture of a 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 [7]. Nevertheless, your data cannot be completely safe. For example, in present IaaS clouds, the virtual machine can be dynamically 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 TCCP (Trusted Cloud Computing Platform) [12] solves this problem by creating the safe environment for running virtual machines. In fact, 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 issues is to deploy the cloud infrastructure on the private hardware. In this case, the owners themselves will protect all the data.

However, buying and maintaining of the hardware is more expensive than rent of computing resources [9], [10].

The simplest way is to create a private cloud system, and there are several different products existing in this area. They can be divided into two categories according to the level of abstraction – infrastructure (IaaS) solutions (e.g. OpenStack and VMWare Private Cloud), and platform (PaaS) solutions (e.g. Yandex Cocaine, AppFog and Stackato).

OpenStack [17] is a popular open source package for private cloud systems provision. It provides infrastructure layer based on virtual machines abstraction. Software developers should install web servers, load balancers and other specific services manually. It offers great opportunities, but IaaS maintenance is expensive. VMWare Private Cloud [16] also provides IaaS-level solution, but on a commercial basis. VMWare Private Cloud offers advanced virtualization capabilities, as well as, the commercial support.

Yandex Cocaine and AppFog platforms provide the ability to create private PaaS solutions based on application containers [2], [3]. These platforms allow development of a “Heroku-like” [13] application hosting. They provide a number of built-in modules and a server infrastructure. Stackato [1] provides all advantages of the mentioned solutions and provide local application store. Some of them use Docker [8] containers to provide isolated environment to the applications, but Docker can be used only on Linux hosts. However, all of the above solutions consider custom applications as a monolith and ignore its internal structure that is 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:

• 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

![Mjolnirr Platform Architecture](image)

Figure 1: Mjolnirr platform architecture
personal computers, using idling resources of the computer system, will provide cost reduction.

- Using popular programming languages, developer tools and SDK will provide ease of application development.
- Integration of new resources ease will be provided by the application architecture modularity and custom components reusability.

IV. ARCHITECTURE

Mjolnir platform architecture is presented at the figure 1. The Mjolnir platform consists of the following components.

- **Proxy** provides access to the cloud system for the external clients and manages the communication between cloud application components. Proxy is the only component that accessible from the external network;
- **Containers** are deployed on computing nodes and responsible for hosting of cloud applications components and message transmission. It can be deployed both on the personal computer and on the nodes of dedicated server;
- **Components** are custom applications, created to run in cloud environment. Each component has a unique name. UI components (applications) are multi-page applications, it means that applications has different representations for different actions, like different web-pages of one web-application;
- **Client** (browser or third-party application) is an application, which render user interfaces and provide user interaction with cloud applications, hosted in Mjolnir platform. All clients should use encrypted channel and client certificates to communicate with the proxy.

A. Proxy

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

- it stores and provides static resources (page layout descriptions, images etc.) of the deployed cloud applications in the Static Storage;
- it acts as a messaging server for the components of cloud applications;
- it handles client’s requests to the Client Interface;
- it performs the authorization and authentication of users.

The external Proxy interface provides the following methods as RESTful API:

- `getUI(applicationName, pageName)` – returns the page layout description;
- `getResourceFile(applicationName, resourceName)` – gets the static file;
- `processRequest(componentName, methodName, args)` processes a client request, redirecting it to the first free suitable component. This method can be called directly from the application page (e. g. with JavaScript).

B. Container

The container (Fig. 3) provides cloud application component hosting. The container provides an API for remote components instances method invocation. The Mjolnirr installation can have any number of containers.

Any Mjolnir-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 operates 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:

- Component Public Channel: every instance of the

![Figure 2: Mjolnirr proxy architecture](image)

![Figure 3: Mjolnirr container architecture](image)
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.

- Instance Private Channel: provides direct communication between instances.

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

- Container registers in the proxy database and receives the list of components to load in response;
- For each component from the list:
  - The container checks it’s local cache, for each missing package container downloads it from the proxy;
  - Container runs the component;
  - Container subscribes on the Component Public Channel and Instance Private Channel for the loaded component.

Container provides messaging API to the hosted component. Typical message transmission sequence looks like shown below (Fig. 4):

- when one component calls another component remotely, the container sends the call to a Component Public Channel (for example, when the Comp_A component calls the Comp_B component, the application Comp_A, will send the request to the channel CPC_B);
- The first available instance of the component in the cloud system processes the request;
- The response is returned to a private IPC channel of the instance that sent a message.

In addition, 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).

C. Components

From the developer’s point of view, Mjolnir cloud application is a collection of independent components communicating by message exchange. Components represented as a package that contains the following information:

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

Each component can be:

- Application Component: provides the user interface definition, scripts, styles and UI-specific actions. Optionally contains complex logic.
- 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.

D. Clients

Mjolnir platform uses HTML and JavaScript to represent end-user interface in common Internet browser. But it’s possible to consume Mjolnir resources in third-party application because of using commonly used and standardized protocols.

V. IMPLEMENTATION

A. Components

Each custom component must have a manifest: an XML file that contains all the necessary information about the component’s name, the name of the main class, list of methods and a list of parameters for each method. Sample manifest for the Calculator service is shown below.

```xml
<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 the 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.

B. Messaging subsystem implementation

Mjolnirr platform messaging subsystem is implemented on the basis of the open source HornetQ queue, which is built into the Proxy component. HornetQ was chosen for the following reasons:

• it can work in embedded mode;
• it provides high efficiency in queue management performance [23].

The container provides messaging API to the components. Example of using messaging interface is shown below.

```
Communicator communicator = new HornetCommunicator();

YourResultClass result = Communicator .sendSync(
    componentContext,
    "componentName",
    "methodName",
    argsList,
    YourResultClass.class);

communicator.send(
    componentContext,
    "componentName",
    "methodName",
    argsList,
    new Callback<YourResultClass>() {
        public void run(YourResultClass res) {}
    });
```

C. UNICORE integration

The Mjolnirr platform provides a UNICORE 6 integration module. This module uses DiVTB Server for communications with grid environment. It enables Mjolnirr applications to use UNICORE grid resources easily.

The concept of DiVTB allows users to split the supercomputing simulation in two phases - building experiment and launching the test stand. The main advantage of the DiVTB concept is the ability to use several times once set up the experiment.

UNICORE integration module provides the following methods:

• uploadTestBed – used for test bed archive uploading;
• indexTestBed – indexes specific test bed;
• indexAllTestBeds – indexes all available test beds;
• createExperiment – create experiment from one of the test beds;
• startExperiment – start experiment with specific parameters;
• getStatus – get experiment status (started, finished or failed);
• getResults – get experiment results.

VI. PERFORMANCE EVALUATION

Mjolnirr platform was evaluated using text-processing experiment. 1 gigabyte of text data was divided on 100 parts and distributed between available worker components for processing. Each worker divided text on words and counted frequency for each unique word. Pieces of work were distributed automatically – each worker polled Message Bus to receive new task. The results of the experiments are shown on the figure 5.

We used 1 virtual node with 2 CPU cores and 2048 MB RAM for proxy node and 10 virtual nodes with 1 CPU core, 512 MB RAM as worker nodes. Virtual machines were installed on the computing server node.
with the following characteristics: 2 Intel Xeon X5680 CPU (6 Cores, 3.33 GHz) 12 GB DDR3 RAM.

Experiments have shown that the platform is stable. Average execution time on 10 containers was 219 seconds, comparing to 1107 seconds using 1 worker. Thus, acceleration of parallel word frequency counter task reached 5.3 on 10 worker nodes.

VII. CONCLUSION

In this article, we described the design and implementation of a private cloud platform called Mjolnirr, which allows creating 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 of the Mjolnirr platform 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.

REFERENCES


