MJOLNIRR: ГИБРИДНЫЙ ПОДХОД К ОРГАНИЗАЦИИ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ

Д.И. Савченко, Г.И. Радченко

Кафедра Системного Программирования, Южно-Уральский Государственный Университет, Челябинск, Россия

1 Введение

В последние несколько лет большое распространение получила концепция облачных вычислений [7]. Облачные вычисления предоставляют повсеместный и удобный доступ к вычислительным ресурсам, что сильно облегчает использование их для построения собственных сервисов.

До появления облачных вычислений, научным сообществом была предложена концепция грид [5], которая и поныне активно используется для решения сверхбольших и сверхресурсоемких научных задач. Название “грид” возникло как метафора электросети (power grid), однако интеграция новых ресурсов и разработка приложений, использующих грид-ресурсы, является значительно более сложной задачей, чем включение электроприбора в сеть.

С другой стороны, существует модель предоставления облачных ресурсов PaaS (Platform as a Service, платформа-как-сервис) [9], предоставляющая возможность простого создания приложений на основе облачной инфраструктуры. Однако большинство PaaS-решений на рынке развернуты на серверах компаний-провайдеров, и владелец данных не может обеспечить сохранность и защищенность этих данных [3]. Эти проблемы может решить только частная PaaS-платформа [9]. Таким образом, является актуальной проблема защищенности данных в облачных системах.

В данной работе рассматривается проектирование и реализация платформы Mjolnirr, обеспечивающей создание частных облачных PaaS-систем. Название Mjolnirr происходит от названия молота Тора (Mjölnir), скандинавского бога грома и молнии. Было решено выбрать это название в качестве метафоры мощного инструмента, связанного с облаками. Любую библиотеку или программу на Java возможно реализовать на основе Mjolnirr как сервис. С точки зрения разработчика, приложение на платформе Mjolnirr представляет собой набор независимых компонентов, осуществляющих коммуникацию посредством передачи сообщений. Такой подход позволяет легко создавать гибкие и масштабируемые системы.

Платформа Mjolnirr также предоставляет интеграцию с грид-средой UNICORE при помощи платформы DiVTB [14]. DiVTB (Distributed Virtual Test Bench) предоставляет проблемно-ориентированный подход для решения задач в области CAE (computer-aided engineering) с использованием грид-ресурсов.

Структура дальнейшего изложения построена следующим образом. В разделе 2 рассматриваются существующие решения в области облачный систем. В разделе 3 рассматривается структура платформы Mjolnirr. В разделе 4 содержится информация об особенностях реализации платформы Mjolnirr, а также иллюстрируется использование платформы для создания собственных приложений. В разделе 5 описываются проведенные эксперименты и их результаты. В конце подводятся итоги проведенной работы и обозначаются направления дальнейших исследований.

2 Анализ существующих решений

Многие руководители крупных компаний не готовы доверить свои данные общественным облачным системам [12]. Одним из наиболее серьезных опасений является возможность нарушения конфиденциальности данных - компания-провайдер облачных сервисов может получить доступ к закрытым данным (случайно или по злому умыслу), что может нанести ущерб владельцу данных.

Для предотвращения угроз конфиденциальности можно использовать шифрование [6], но такой подход эффективен только при использовании облака исключительно для хранения. Если же в облаке производится обработка данных, то они становятся доступными в открытом виде в оперативной памяти вычислительного узла, на котором происходит обработка [8]. Кроме того, владелец данных не контролирует местоположение виртуальной машины - она может быть перемещена на один физический узел с машиной, содержащей вредоносное ПО, при этом они будут иметь один IP-адрес. Это может повлечь блокировку сервера интернет-провайдером либо конфискацию физического носителя, содержащего обе виртуальных машины.

Концепция TCCP (Trusted Cloud Computing Platform) [10] создает доверенную среду для работы виртуальной машины. Однако, ни один из обозначенных подходов не решает проблему нахождения виртуальной машины на одном узле с машиной злоумышленника.

Простейший способ решения этой проблемы — использование частного облака. При этом, для обеспечения легкости разработки новых приложений, желательно, чтобы частная облачная платформа предлагала набор готовых API для разработки и прозрачного развертывания. На данный момент существует несколько решений в этой области — *Yandex Cocaine* [13], *Stackato* [11] и *AppFog* [8]. Эти платформы позволяют

создавать хостинги для приложений наподобие Heroku [4], имеют ряд встроенных модулей и серверную инфраструктуру. *Stackato* предоставляет также локальный магазин приложений. Однако эти решения не учитывают внутреннюю структуру приложения, и это не позволяет эффективно балансировать нагрузку автоматически; а также, ни одно из этих решений не ориентированно на возможное использование персональных компьютеров конечных пользователей в качестве рабочих узлов.

Нами было принято решение создать облачную платформу, отвечающую следующим требованиям:

1. *Снижение стоимости* должно обеспечиваться работой не только на серверных платформах, но и на пользовательских компьютерах, используя свободные ресурсы вычислительной системы.

2. Простота разработки приложений должна обеспечиваться использованием популярных ЯП и средств разработки.

3. *Легкость интеграции новых ресурсов* должна обеспечиваться модульностью архитектуры приложений и возможностью повторного использования компонентов.

3 АРХИТЕКТУРА MJOLNIRR

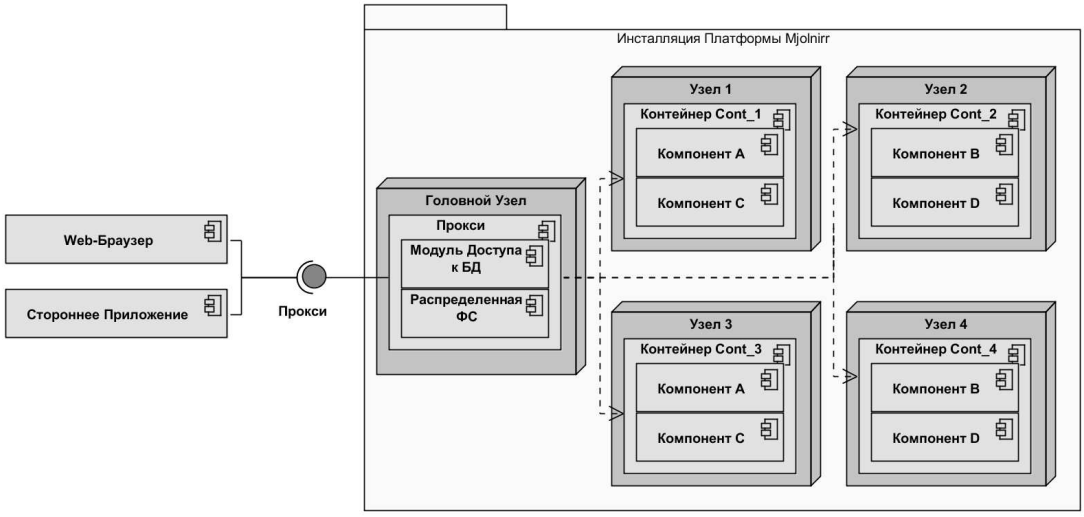


Рис. 1. Архитектура платформы Mjolnirr

На рисунке 1 представления архитектура платформы Mjolnirr.

Любое Mjolnirr-приложение состоит из нескольких независимых компонентов, взаимодействующих между собой при помощи встроенной системы обмена сообщениями, реализованной на базе паттерна «Издатель-Подписчик». Сервер обмена сообщениями, запущенный внутри Прокси, позволяет подписываться на различные каналы, которые представляют широковещательную доставку сообщений.

В состав платформы входят следующее подсистемы:

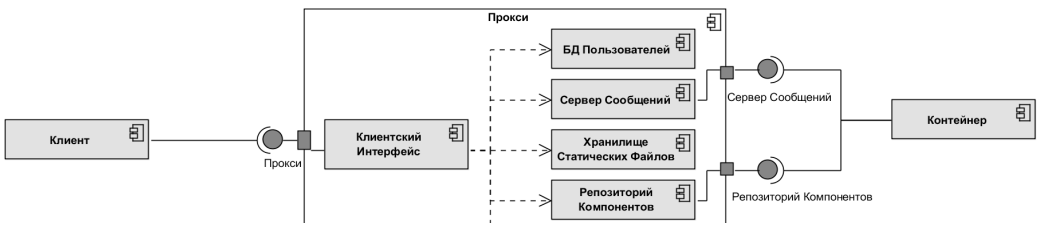
1. *Прокси* — компонент, который редоставляет доступ к системе клиентам, хранит и возвращает по запросу статические файлы, поддерживает очередь сообщений, занимается распределением компонентов по контейнерам, выполняет авторизацию и аутентификацию пользователей. Все системные сервисы (модули аутентификации, доступа к БД) запущены внутри Прокси. Это единственный компонент инсталляции платформы, доступный из внешней сети.

2. *Контейнер* — это компонент, который отвечает за размещение пользовательских компонентов и передачу сообщений, может быть развернут на пользовательских компьютерах.

3. *Компоненты* — пользовательские приложения, созданные для работы на платформе Mjolnirr. Каждый компонент имеет уникальное имя. Приложения, имеющие интерфейс, могут иметь одну или несколько страниц, наподобие веб-портала.

4. *Клиентские приложения (Веб-браузер и сторонние приложения)* — все клиентские приложения используют защищенный канал для связи с Прокси. Каждый клиент должен предоставлять личный сертификат для аутентификации. Интерфейс приложений представлен в виде стандартной HTML-страницы и интерпретируется браузером как веб-сайт.

3.1 Прокси



На рисунке 2 представлена архитектура Прокси. Этот компонент предоставляет доступ к установке Mjolnirr из внешней сети. Он выполняет следующие функции:

1. хранит и отдает по запросу статические файлы (CSS, JS и т. д.) для развернутых компонентов;
2. предоставляет реестр компонентов всем запущенным контейнерам;
3. выполняет роль сервера обмена сообщениями;
4. обрабатывает клиентские запросы;
5. производит пользовательскую авторизацию и аутентификацию;
6. производит балансировку активных компонентов по запущенным контейнерам;
7. отвечает за хостинг системных сервисов платформы.

3.2 Контейнер

Основной задачей компонента Container является хостинг компонентов и вызов их методов по требованию. Контейнеров может быть произвольное количество. Каждый контейнер подписан на 2 типа каналов сообщений (для каждого компонента, установленного в рамках контейнера):

1. *Публичный канал компонента.* Каждый экземпляр компонента приложения подписан на этот канал. Когда сообщение, адресованное этому компоненту, приходит в систему, первый доступный экземпляр перехватывает его.

2. *Приватный канал экземпляра.* Предоставляет доступ к конкретному экземпляру компонента. Используется для возвращения результата обработки запроса.

При запуске, контейнер выполняет ряд шагов инициализации. Порядок запуска контейнера:

1. Контейнер загружает все JAR-архивы, которые находятся в его директории компонентов;
2. Для каждого загруженного компонента:
    a) Контейнер разбирает манифест компонента;
    b) Контейнер отправляет архив компонента к прокси для развертывания статических файлов;
    c) Контейнер подписывается на публичные и приватные каналы компонента.

Контейнер предоставляет API для обмена сообщениями всем экземплярам компонентов, запущенных в этом контейнере. Типичная последовательность действий при обмене сообщениями выглядит так:

1. Когда экземпляр компонента вызывает другой компонент, он посылает запрос в Публичный канал компонента;
2. Первый свободный экземпляр вызываемого компонента перехватывает и обрабатывает запрос;
3. Ответ возвращается в Приватный канал экземпляра компонента, отправившего запрос.

3.3 Компоненты

С точки зрения разработчика, приложение на платформе Mjolnirr представляет собой набор независимых компонентов, осуществляющих коммуникацию посредством передачи сообщений. Каждый из этих компонентов представляет собой архив, содержащий:

1. Манифест, в котором описан интерфейс компонента;
2. Исполняемые файлы, используемые для обработки входящих сообщений;
3. Статические файлы, используемые при отображении веб-страниц и в работе компонентов.

Каждый компонент может быть одного из двух типов:

1. Компонент-приложение предоставляет пользовательский интерфейс, скрипты и стили в виде статических файлов, упакованных в архив приложения, а также логику обработки запросов от пользователя;

2. Компонент-модуль представляет одну сущность из предметной области в рамках приложения. Предоставляет функции обработки данных и обращения к хранилищу, но не имеет графического интерфейса.

3.4 Клиенты

Платформа Mjolnirr использует HTML и JavaScript для предоставления пользовательского интерфейса, однако возможно создание сторонних приложений, использующих API платформы. В качестве клиента было решено использовать самое распространённое ПО – веб-браузер. Платформа поддерживает все современные браузера, а также современные стандарты веб-разработки (HTML5, CSS3, JavaScript 5).

4 Реализация

4.1 Компоненты

Для реализации системы был разработан механизм определения интерфейса пользовательских компонентов на основе механизма аннотаций языка Java 6. На рисунке 3 представлен пример класса-интерфейса для сервиса *Calculator*.

```
@MjolnirrComponent(
    componentName = "calculator",
    instancesMinCount = 1,
    instancesMaxCount = 255,
    memoryVolume = 128)
```

```
public class Calculator extends AbstractApplication {
    private ComponentContext context;
    @MjolnirrMethod public String calculate(String expression)
            throws Exception {
        return Helper.calculate(expression);
    }

    @Override public void initialize(ComponentContext context) {
        this.context = context;
    }
}
```

Рис. 3. Интерфейс сервиса Calculator

Аннотация *@MjolnirrComponent* обозначает класс-интерфейс компонента и содержит в себе поля:

1. *componentName* – имя компонента в системе. Под этим именем компонент будет доступен для удаленных вызовов.
2. *instancesMinCount* – минимальное количество экземпляров компонента в системе.
3. *instancesMaxCount* – максимальное количество экземпляров компонента в системе.
4. *memoryVolume* – количество памяти в мегабайтах, необходимое компоненту для нормальной работы.

Аннотация *@MjolnirrMethod* обозначает отдельный метод класса-интерфейса как экспортируемый. Аннотация имеет поле *executionTime,* значение которого показывает максимальную длительность исполнения метода в секундах. По умолчанию имеет значение 30.

Контейнер производит разбор аннотаций класса-интерфейса для каждого загружаемого компонента, считывая описание предоставляемых методов компонента.

4.2 Реализация Прокси

Подсистема реализована как REST-сервис, имеющий следующие методы:

- *GET /static/{application}/{resource}* – получить статический файл *resource* приложения *application*.
- *POST /request* – отправить запрос в облако. POST-запрос имеет поля *applicationName* (имя вызываемого компонента), *methodName* (имя метода компонента) и *args* (список аргументов, передаваемых методу). В ответ возвращается структура, имеющая два поля: *responseClass* (фактический класс возвращаемого объекта) и *responseObject* (сериализованный объект).
- *POST /tmp* – загрузить временный файл в облако. Служебный метод.
- *GET /tmp/{filename}* – выгрузить временный файл из облака, после этого он удаляется. Используется для выгрузки файлов, являющихся результатами работы облачных компонентов.
- *GET /{application}/{page}* – получить страницу *page* приложения *application*.

4.3 Система обмена сообщениями

Система обмена сообщениями платформы Mjolnirr реализована на базе системы очередей HornetQ, которая работает в режиме «Издатель-Подписчик». Контейнер предоставляет запущенным компонентам API для обмена сообщениями. Пример использования интерфейса приведен на рисунке 4.

```
Communicator communicator = new HornetCommunicator();

YourResultClass result = communicator.sendSync(componentContext,
"component-Name", "methodName", argsList, YourResultClass.class);

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

Рис. 4. Пример использования системы обмена сообщениями

4.4 Интеграция с UNICORE

Платформа Mjolnirr предоставляет модуль интеграции с UNICORE, основанный на концепции DiVTB.

Модуль интеграции с предоставляет следующие методы:

1. *void uploadTestBed (content)* – загружает архив со стендом в базу стендов сервера DiVTB;
2. *String createExperiment (bedID)* – создает эксперимент из стенда с ID "*bedID*", возвращает UID эксперимента;
3. *void startExperiment (experimentID, params)* – запускает эксперимент с параметрами, переданными в файле *params*;
4. *int getStatus (experimentID)* – возвращает статус выполнения эксперимента (запущен, завершен или провален);
5. *byte[] getResults (experimentID)* – возвращает результаты эксперимента в виде ZIP-архива.

Модуль является стандартным модулем платформы Mjolnirr, любой компонент может использовать его для доступа к грид-среде UNICORE.

5 Тестирование производительности

Платформа Mjolnirr была протестирована на 11 узлах, один из которых был оборудован 4-ядерным процессором и 2 гигабайтами оперативной памяти и играл роль сервера, а остальные имели по одному одноядерному процессору и по 512 мегабайт оперативной памяти и играли роль пользовательских машин.

Тестовый файл размером в 1 гигабайт был разбит на 100 файлов по 10 мегабайт. Каждый фрагмент файла был отправлен в очередь для обработки. Запущенные экземпляры обработчика брали отправленные фрагменты и подсчитывали количество вхождений для каждого слова в тексте. Результаты эксперимента представлены на рисунке 5.

Вычислительные эксперименты показали, что платформа стабильна, нагрузка распределяется по доступным экземплярам компонентов равномерно. Среднее время выполнения задачи на 10 экземплярах составила 219 секунд, было достигнуто ускорение, равное 5.3 на 10 экземплярах компонентов-обработчиков.

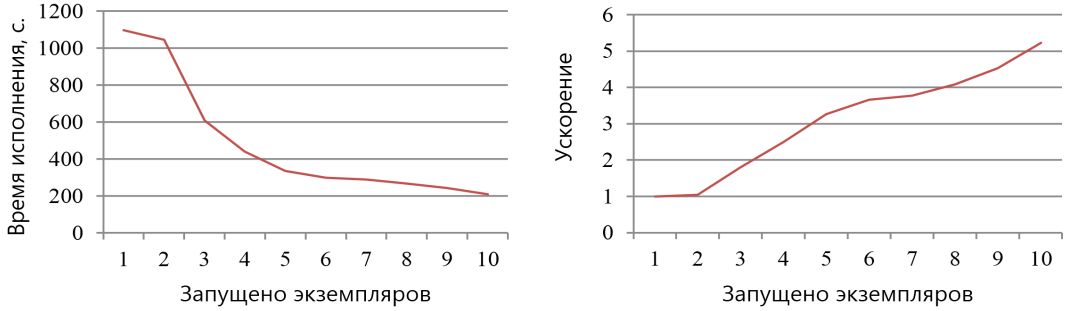


Рис. 5. Обработка большого количества текстовых данных

6 Заключение

В этой статье были описаны архитектура и реализация частной облачной платформы Mjolnirr, позволяющей создавать распределенные приложения на частных серверах. Дальнейшим развитием платформы Mjolnirr станет поддержка миграций на уровне приложений, интеграция со средствами мониторинга, гибкая адаптация под изменяющуюся нагрузку, улучшение безопасности системы.

Работа выполнена при поддержке Фонда Содействия Развитию Малых Форм предприятий в научно-технической сфере №0000829 а также Российского фонда фундаментальных исследований (грант № 14-07-00420).

ЛИТЕРАТУРА:

1. A. Streit. UNICORE: Getting to the heart of Grid technologies // eStrategies. Vol. 3. 2009. P. 8-9.
2. AppFog [Электронный ресурс]. URL: https://www.appfog.com/ (дата обращения: 11.01.2014).
3. Chang W. Y., Abu-Amara H., Sanford J., Transforming Enterprise Cloud Services // Springer Science+Business Media. Springer. 2010.
4. Heroku [Электронный ресурс]. URL: https://www.heroku.com/ (дата обращения: 11.01.2014)
5. I. Foster and C. Kesselman, The Grid 2: Blueprint for a New Computing Infrastructure. The Morgan Kaufmann Series in Computer Architecture and Design Series. Elsevier Science, 2003.
6. Kamara, S. and Lauter, K. Cryptographic cloud storage // Proceedings of Financial Cryptography: Workshop on Real-Life Cryptographic Protocols and Standardization. NEC. 2010.
7. Mell P., Grance T. The NIST Definition of Cloud Computing. National Institute of Standards and Technology (NIST), USA, 2011. 7 c.
8. Ren, L. et al, Design Space Exploration and Optimization of Path Oblivious RAM in Secure Processors // Proceedings of the 40th Annual International Symposium on Computer Architecture. ACM. C. 571-582, 2013.
9. Rhoton J., Haukioja R., Cloud Computing Architected. Recursive, 2011.
10. Santos N., Gummadi K., Rodrogues R., Towards Trusted Cloud Computing // HotCloud'09 Proceedings of the 2009 conference on Hot topics in cloud computing. USENIX Association Berkeley. article No. 3.

11. Stackato [Электронный ресурс]. URL: http://www.activestate.com/stackato (дата обращения: 11.01.2014).
12. Survey: Cloud Computing ’No Hype’, But Fear of Security and Control Slowing Adoption [Электронный ресурс]. URL: http://www.circleid.com/posts/20090226_cloud_computing_hype_security/ (дата обращения: 11.01.2014).
13. Yandex Cocaine [Электронный ресурс]. URL: http://api.yandex.com/cocaine/ (дата обращения: 11.01.2014).
14. Савченко Д.И., Радченко Г.И. DiVTB Server: среда выполнения виртуальных экспериментов // Параллельные вычислительные технологии (ПаВТ'2013): труды международной научной конференции (1–5 апреля 2013 г., г. Челябинск). Челябинск: Издательский центр ЮУрГУ, 2013. С. 532-539.