Автоматическое создание виртуальных кластеров Apache Spark в облачной среде Openstack[1]

О. Д. Борисенко <al@somestuff.ru>
Д. Ю. Турдаков <turdakov@ispras.ru>
С. Д. Кузнецов <kuzloc@ispras.ru>
ИСП РАН, 109004, Россия, г. Москва, ул. А. Солженицына, дом 25

Аннотация. В работе описывается процесс создания системы автоматического создания виртуальных кластеров Apache Spark в среде Openstack. Также в работе приводится краткий обзор различий между предоставляемыми метаданными в средах Openstack и Amazon EC2.

Ключевые слова: Apache Spark, Openstack, Amazon EC2, Map-Reduce, HDFS, виртуальные кластеры, облачные вычисления, Big Data.

1. Введение

На текущий момент технологии вычислений в программной модели Map-Reduce становятся все более востребованными. Существует множество реализаций данного подхода [1], [2], [3], [4], и проект Apache Spark [5] является одной из таких реализаций. Проект Apache Spark изначально разрабатывался институтом Беркли и после открытия исходных кодов в 2010 году активно развивается множеством независимых разработчиков. Apache Spark является одной из самых быстрых реализаций Map-Reduce [6], [7].

В то же время все существующие проекты, реализующие Map-Reduce, обладают сложной архитектурой, и процесс настройки кластерного окружения для их использования является сложным и запутанным. Умение программировать в парадигме Map-Reduce совершенно не обязывает разработчика и исследователя уметь настраивать сложные кластерные окружения.

[1] Работа выполнена при поддержке гранта РФФИ №14-07-00602 А «Исследование и разработка методов автоматизации масштабирования и разворачивания виртуальных кластеров для обработки сверхбольших объёмов данных в облачной среде Openstack»

На момент начала работы над данным проектом существовало лишь несколько проектов для автоматизации развертывания окружения для распределенных вычислений в парадигме Map-Reduce, и ни один из них не предполагал использование открытых облачных сред в качестве платформы для развертывания вычислительного кластера [8], [9]; более того, все эти проекты подразумевают использование лишь классического стека технологий Apache Hadoop или Cloudera Hadoop. Единственный проект, ориентированный на открытые облачные среды (Openstack Sahara [10]) был в самом начале цикла разработки, и он также подразумевал использование Apache Hadoop в качестве реализации Map-Reduce. Использование Apache Spark появилось в планах лишь недавно, и проект очень далек от завершения.

Таким образом, задача автоматического построения виртуальных кластеров с готовым окружением для выполнения задач в Apache Spark кажется очень актуальной. Кроме того, такой подход позволяет очень точно делить аппаратные ресурсы между разработчиками и выделять им ровно столько ресурсов, сколько им требуется, без привязки к нижележащему аппаратному обеспечению. Целью данной работы является предоставление решения для автоматизации создания виртуальных кластеров Apache Spark в облачной среде Openstack.

2. Компоненты Apache Spark

В этом разделе приводится краткий обзор основных компонентов Apache Spark.

2.1 Уровень хранения

В качестве системы хранения данных Apache Spark использует распределенную файловую систему HDFS [11]. Разработчики фреймворка предполагают единовременное использование двух файловых систем HDFS: одну предполагается использовать для долговременного хранения результатов работы программ, вторую же предполагается использовать для оперативного хранения промежуточных данных вычислений и для распределения заданий между вычислительными узлами кластера. Apache Spark поддерживает все известные реализации и версии файловой системы HDFS.

2.2 Вычислительные узлы

В Apache Spark используется разделение ролей вычислительных узлов. Один из узлов назначается управляющим и называется «мастером». Остальные узлы предназначаются для выполнения вычислений и называются «рабочими» (workers). Программа запускается на «мастер»-узле, и он автоматически формирует задания для «рабочих» узлов. После выполнения заданий «рабочие» узлы уведомляют «мастер» узел о результате исполнения заданий.

2.3 Слежение за жизнедеятельностью кластера

От версии к версии в проекте Apache Spark менялась модель слежения за работоспособностью вычислительных узлов. В первых версиях проекта использовалось разделение ответственности: слежение за работоспособностью узлов производилось при помощи выделенных узлов, на которых специальным образом настраивался ZooKeeper [12]. В более поздних версиях от этого подхода отказались, и был реализован собственный протокол опроса узлов о доступных им ресурсах и слежения за работоспособностью. В данной работе используются только встроенные в Apache Spark механизмы слежения за работоспособностью кластера.

Отдельно стоит отметить, что для слежения за использованием ресурсов узлами вычислительной сети, разработчки проекта Apache Spark предлагают использовать специализированный инструмент для слежения за нагрузкой под названием Ganglia [13]. В рамках данной работы также используется этот инструмент.

3. Существующие способы создания кластеров Apache Spark в облачных средах

На момент начала работы над данным проектом существовало ровно два способа создания кластеров Apache Spark.

Первый способ предполагает ручную конфигурацию всех компонентов системы. То есть для создания кластерного окружения необходимо задать конфигурацию для следующих подсистем:

- Apache Hadoop (v1/2.x) в двух экземплярах
- Окружений JVM и Scala
- Ganglia
- Apache Spark

Настройка JVM и Scala подразумевает установку всех необходимых пакетов и настройку переменных окружения для дальнейшего их использования в Apache Spark и Apache Hadoop.

Apache Hadoop и Apache Spark конфигурируются при помощи множества файлов XML, которые должны содержать роли каждого из узлов, IP-адреса и порты для соединений между узлами, уровни репликации HDFS, конфигурацию нижележащего уровня под файловой системой HDFS и так далее.

Ganglia также подразумевает настройку с разделением ролей: на каждом узле должен существовать конфигурационный файл с указанием тех ресурсов, за которыми должен следить демон, и должен существовать конфигурационный файл на «мастер»-узле для слежения за ресурсами кластера в целом.

Второй способ предназначается для настройки кластеров в облачной среде Amazon EC2 [14][15]. Настройка окружения подразумевает использование заранее сконфигурированного образа виртуальной машины с установленными компонентами без конфигурации и специального набора скриптов для настройки компонентов. Образ виртуальной машины использует два типа блочных устройств для хранения данных в HDFS: устройство для перманентного хранения данных и устройство для оперативных данных HDFS и раздела для своппинга. Второе блочное устройство называется термином «ephemeral» и уничтожается при остановке кластера.

Скрипты настройки выполняют следующую последовательность действий:

- при помощи программных интерфейсов Amazon EC2 (далее – API Amazon EC2) инициируется процесс запуска необходимого количества виртуальных машин из заранее настроенного образа на базе ОС Red Hat Enterprise Linux 5.3;
- с помощью API Amazon EC2 настраиваются правила ограничений сетевого доступа к сетевым портам каждой из виртуальных машин;
- скрипт впадает в режим ожидания ровно на 5 минут;
- после истечения пяти минут скрипт «узнает» выданные IP-адреса для каждой из виртуальных машин при помощи API Amazon EC2; в случае, если виртуальные машины не успели запуститься, скрипт выдает ошибку;
- с использованием полученных данных заполняются конфигурационные файлы для всех компонентов сиситем. «Мастер» узел назначается NameNode для обеих файловых систем HDFS;
- «мастер»-узлу передаются все конфигурационные файлы при помощи ssh-подключения;
- «мастер»-узел располагает полученные файлы в заранее определенных папках и по ssh раздает дочерним узлам соответствующие им конфигурационные файлы;
- «мастер»-узел инициирует запуск служб HDFS и Apache Spark на всех узлах.

После выполнения перечисленных действий управление возвращается на терминал пользователя, и пользователю сообщаются необходимые данные для подключения к свежесозданному кластеру.

Ключевые особенности второго способа:

- используется API закрытой облачной системы Amazon EC2;
- используется заранее настроенный образ операционной системы RHEL 5.3, который нельзя использовать как из-за лицензионных ограничений данного дистрибутива, так и из-за невозможности извлечь образы виртуальных машин из среды Amazon EC2.

Таким образом, оба способа неприменимы для использования в облачной среде Openstack из-за сложности настройки отдельных компонентов для первого способа и из-за явной привязки к Amazon EC2 для второго способа.

4. Построение решения

Существует несколько подходов для достижения поставленной задачи:

1. переиспользование существующих скриптов настройки кластера в Amazon EC2 при помощи адаптеров API в среде Openstack (подсистема Heat);
2. использование систем оркестрации;
3. адаптация (портирование) решения для среды Openstack.

Рассмотрим эти подходы подробнее.

4.1 Использование подсистемы Openstack Heat

На момент начала работы над проектом в проекте Openstack существовал специальный побочный проект под названием Heat [16]. Целью проекта являлось обеспечение совместимости с Openstack скриптов, предназначенных для использования в Apache EC2. Предполагалось, что при использовании этой прослойки Openstack сможет «притворяться» Amazon EC2 для конечного пользователя и обеспечивать точно такое поведение. Тем не менее, в тот момент проект находился в зачаточном состоянии и предоставлял менее половины от вызовов к API Amazon EC2.

Необходимо отдельно отметить, что Amazon EC2 и Openstack реализуют совершенно разные модели архитектуры облачных сервисов. В частности, обе облачные платформы предлагают пользователю хранить определенный набор метаданных, связанных с виртуальными машинами, но различается как и сам набор метаданных, так и уровни доступа к ним. Openstack предоставляет модель, в которой каждая виртуальная машина может иметь доступ только к собственным метаданным, в то время как Amazon EC2 позволяет использовать метаданные и «снаружи» виртуальной машины.

Кроме того, облачные сервисы по-разному реализуют виртуальную сетевую инфраструктуру. В рамках Amazon EC2 каждая виртуальная машина получает два IP-адреса: один является «внешним» и используется для доступа из любой точки сети Интернет, второй адрес является «внутренним» и доступен только в пределах виртуальной сети между виртуальными машинами конкретного пользователя. Эти настройки являются фиксированными, и повлиять на них нельзя, так что и вызовов в API Amazon EC2 для настройки сети не существует.

В то же время, Openstack позволяет очень гибко настраивать сетевое окружение. Каждая сеть в Openstack обладает собственным именем и идентификатором, который задается на этапе настройки среды Openstack. При создании виртуальной машины необходимо явно указывать, какие именно

сети будет использовать виртуальная машина; если этого не сделать, то виртуальная машина получает сетевые интерфейсы по умолчанию, которые задаются администратором Openstack, и настройки этих сетей нужно знать заранее, поскольку Heat не предоставляет данных о конфигурации сети.

Таким образом, адаптер Heat в рамках этого проекта использовать было невозможно с самого начала по двум причинам.

- Поскольку Heat в точности повторяет вызовы Amazon EC2, с его помощью невозможно использовать сети, отличные от заданных администратором Openstack в качестве сетей по умолчанию. Это накладывает ограничения на пользователя, так как ему необходимо иметь непосредственный контакт с администратором Openstack для назначения нужных настроек по умолчанию, а этот вариант не всегда доступен.
- Недостаточный уровень совместимости вызов Heat с API Amazon EC2. В частности, в данном случае оказалось, что система именования виртуальных машин в Openstack API и Heat API отличается: Heat API не способен установить hostname виртуальной машины, и виртуальные машины, созданные при помощи API Heat получают свой внутренний IP-адрес в качестве hostname. В то же время, Apache Spark для настройки узлов требует именования узлов в соответствии с FQDN, и в случае, если узел имеет hostname, не соответствующий FQDN, настройки считаются некорректными. Более того, ответ на вызов API Heat, соответствующий запросу «какой у меня hostname», всегда возвращается пустым.

Таким образом, мы не можем переиспользовать существующий набор утилит для настройки виртуального кластера без изменений.

4.2 Использование систем оркестрации

На момент начала работы над проектом, существовало несколько распространенных систем оркестрации: Puppet [17], Chef [18], Salt [19] и Ansible [20].

Puppet, Chef и Salt используют разделение ролей на управляющие узлы и дочерние узлы и предполагают установку специальных агентов на дочерние узлы. Это накладывает существенные ограничения на использование этих систем, поскольку в таком случае необходимо специальным образом настраивать управляющий узел. Кроме того, необходимо использовать специально настроенный образ операционной системы на дочерних узлах, в котором будет содержаться настроенный агент системы оркестрации. Это кажется избыточным шагом, требует особых знаний от пользователя и усложняет систему в целом, поэтому от Puppet, Chef и Salt было решено отказаться. Кроме того, указанные системы являются коммерческими и ограничивают возможности использования бесплатной версии.

Ansible выгодно отличается от этих систем.

- Ansible не накладывает никаких функциональных ограничений на бесплатную версию.
- В качестве зависимостей Ansible использует только ssh и python 2.6 или выше.
- Ansible очень прост в освоении.
- Сценарии настройки можно переиспользовать в любой среде: настройка дочерних узлов и настройка окружения (будь то облачный сервис или физические сервера) не взаимосвязаны. То есть возможно создать один большой сценарий настройки Apache Spark и отдельное множество маленьких сценариев для настройки нижележащей инфраструктуры, и сценарий Apache Spark будет работать везде одинаково.

К сожалению, на момент начала проекта в Ansible еще не было поддержки Openstack (из сред виртуализации была поддержка только Amazon EC2 и Vagrant). Но на момент публикации поддержка всех открытых облачных сред уже есть, так что перенос решения на Ansible с целью унификации и отстранения от вопросов инфраструктуры, кажется перспективным направлением для дальнейшей работы.

4.3 Адаптация решения для среды Openstack

В качестве последнего подхода для достижения поставленной задачи, предлагается полная адаптация существующего решения для Amazon EC2 к среде Openstack. Именно этот способ и был реализован в данной работе.

Процесс адаптации состоит из двух этапов:

1. Создание базового образа виртуальной машины на основе CentOS 6.4.
2. Портирование скриптов для создания виртуального кластера Apache Spark с использованием нативного API Openstack.

Создание базового образа не представляет особой сложности, тем не менее, занимает много времени. Зависимости и необходимые программы для базового образа уже были перечислены в разделе 3. Кроме того, необходимо было подготовить образ к настройке собственных параметров в среде Openstack, для этого существуют уже настроенные образы дистрибутива CentOS (хотя автор делал эти шаги самостоятельно, чтобы исключить дополнительные источники возможных ошибок). Заранее подготовленные образы для различных версий Apache Spark можно найти по адресу http://spark.at.ispras.ru/ , список образов пополняется по мере выхода новых версий дистрибутива. В подготовленные образы также включен настроенный дистрибутив хранилища ключей/значений Redis [21].

Портирование скриптов было произведено при помощи библиотек, предоставляемых проектом Openstack. Исходные коды публично доступны по адресам https://github.com/ispras/incubator-spark и https://github.com/ispras/spark-

openstack под лицензией Apache 2.0. Инструкции по использованию находятся там же.

Разработанные скрипты также обладают дополнительной функциональностью по сравнению с изначальным вариантом для Amazon EC2:

- Явное указание времени ожидания старта виртуальных машин теперь не используется: вместо этого реализован механизм опроса состояния виртуальных машин, поэтому ожидание длится ровно столько, сколько требуется машинам для запуска.
- Добавлен слой гибкой конфигурации виртуальных сетей, используемых виртуальными машинами.

5. Результаты

В рамках данной работы было проведено исследование методов развертывания и масштабирования программных средств для обработки сверхбольших объемов данных в облачной среде Openstack, и было разработано открытое и свободное программное решение для автоматизации создания виртуальных кластеров Apache Spark и Apache Hadoop/HDFS в облачной среде Openstack. Побочно был реализован механизм настройки Apache Hive и Apache Shark, но эти инструменты протестированы не были, и были исключены из репозитория из-за недостаточного тестирования.

Данное решение уже активно используется в текущих исследованиях, в скором времени будет опубликована научная работа, посвященная генерации случайных социальных графов [21], использующая в своей основе Apache Spark и результаты данной работы. Использовались кластеры размером в 49 виртуальный узел по 8ГБ оперативной памяти и по 2 вычислительных ядра на каждом из них.

Список литературы

[1]. Страница проекта Apache Hadoop — http://hadoop.apache.org/
[2]. Страница проекта Cloudera CDH Apache Hadoop — http://www.cloudera.com/content/cloudera/en/products-and-services/cdh.html
[3]. Страница проекта Infinispan — http://infinispan.org/
[4]. Страница проекта Basho Riak — http://basho.com/riak/
[5]. Страница проекта Apache Spark — http://spark.apache.org/
[6]. M. Chowdhury, M. Zaharia, I. Stoica. Performance and Scalability of Broadcast in Spark. 2010.
[7]. Gu, Lei, and Huan Li. Memory or Time: Performance Evaluation for Iterative Operation on Hadoop and Spark. High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on. IEEE, 2013.
[8]. Страница проекта VMWare Serengeti — http://www.vmware.com/hadoop/serengeti
[9]. Страница проекта Cloudera Manager — http://www.cloudera.com/content/cloudera/en/products-and-services/cloudera-enterprise/cloudera-manager.html

[10]. Страница проекта Openstack Sahara, план очередности разработки — https://wiki.openstack.org/wiki/Sahara/Roadmap
[11]. Foley, Matt. High Availability HDFS. 28th IEEE Conference on Massive Data Storage, MSST. Vol. 12. 2012.
[12]. Hunt, Patrick, et al. ZooKeeper: Wait-free Coordination for Internet-scale Systems. USENIX Annual Technical Conference. Vol. 8. 2010.
[13]. Massie, Matthew, B. Chun, and D. Culler. The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing 30.7 (2004): 817-840.
[14]. Страница сервиса Amazon Elastic Compute Cloud (EC2) — http://aws.amazon.com/ec2/
[15]. Creeger, Mache. Cloud Computing: An Overview. ACM Queue 7.5 2009.
[16]. Страница проекта Openstack Heat — https://wiki.openstack.org/wiki/Heat
[17]. Yokoyama, Shigetoshi, and Nobukazu Yoshioka. Cluster as a Service for self-deployable cloud applications. Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on. IEEE, 2012.
[18]. Страница проекта Chef http://www.getchef.com/
[19]. Страница проекта Salt http://www.saltstack.com/
[20]. Страница проекта Ansible http://www.ansible.com/home
[21]. *Ожидает публикации*. К. Чихрадзе, А. Коршунов, Н. Бузун, Н. Кузюрин. Использование модели социальной сети с сообществами пользователей для распределённой генерации случайных социальных графов. 10-я Международная конференция «Интеллектуализация обработки информации» 2014.

Automating cluster creation and management for Apache Spark in Openstack cloud

O. Borisenko <al@somestuff.ru>
D. Turdakov <turdakov@ispras.ru>
S. Kuznetsov <kuzloc@ispras.ru>
ISP RAS, 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation

Abstract. This article is dedicated to automation of cluster creation and management for Apache Spark MapReduce implementation in Openstack environments. As a result of this project open-source (Apache 2.0 license) implementation of toolchain for virtual cluster on-demand creation in Openstack environments was presented. The article contains an overview of existing solutions for clustering automation in cloud environments by the start of 2014 year. The article provides a shallow overview of issues and problems in Openstack Heat project that provides a compatibility layer for Amazon EC2 API. The final implementation provided in the article is almost strainforward port of existing toolchain for cluster creation automation for Apache Spark in Amazon EC2 environment with some improvements. Also prepared base system virtual machine image for Openstack is provided. Plans for further work are connected with Ansible project. Using Ansible for observed problem will make possible to implement versatile environment-agnostic solution that is able to work using any cloud computing services provider, set of Docker containers or bare-metal clusters without any dependencies for prepared operating system image. Current article doesn't use Ansible due to the lack of key features at the moment of the project start. The solution provided in this article has been already tested in production environment for graph theory research arcticle.

Keywords: Apache Spark, Openstack, Amazon EC2, Map-Reduce, HDFS, virtual cluster, cloud computing, Big Data.

References

[1]. Apache Hadoop project web page — http://hadoop.apache.org/
[2]. Cloudera CDH Apache Hadoop project web page — http://www.cloudera.com/content/cloudera/en/products-and-services/cdh.html
[3]. Infinispan project web page — http://infinispan.org/
[4]. Basho Riak project web page — http://basho.com/riak/
[5]. Apache Spark project web page — http://spark.apache.org/
[6]. M. Chowdhury, M. Zaharia, I. Stoica. Performance and Scalability of Broadcast in Spark. 2010.
[7]. Gu, Lei, and Huan Li. Memory or Time: Performance Evaluation for Iterative Operation on Hadoop and Spark. High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on. IEEE, 2013.

[8]. VMWare Serengeti project web page — http://www.vmware.com/hadoop/serengeti
[9]. Cloudera Manager project web page — http://www.cloudera.com/content/cloudera/en/products-and-services/cloudera-enterprise/cloudera-manager.html
[10]. Openstack Sahara project web page, roadmap — https://wiki.openstack.org/wiki/Sahara/Roadmap
[11]. Foley, Matt. High Availability HDFS. 28th IEEE Conference on Massive Data Storage, MSST. Vol. 12. 2012.
[12]. Hunt, Patrick, et al. ZooKeeper: Wait-free Coordination for Internet-scale Systems. USENIX Annual Technical Conference. Vol. 8. 2010.
[13]. Massie, Matthew, B. Chun, and D. Culler. The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing 30.7 (2004): 817-840.
[14]. Amazon Elastic Compute Cloud (EC2) service webpage — http://aws.amazon.com/ec2/
[15]. Creeger, Mache. Cloud Computing: An Overview. ACM Queue 7.5 2009.
[16]. Openstack Heat project web page — https://wiki.openstack.org/wiki/Heat
[17]. Yokoyama, Shigetoshi, and Nobukazu Yoshioka. Cluster as a Service for self-deployable cloud applications. Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on. IEEE, 2012.
[18]. Chef project web page — http://www.getchef.com/
[19]. Salt project web page — http://www.saltstack.com/
[20]. Ansible project web page — http://www.ansible.com/home
[21]. *In print*. K. Chikhradze, A. Korshunov, N. Buzun, N. Kuzyurin. Ispol'zovanie modeli sotsial'noj seti s soobshhestvami pol'zovatelej dlya raspredelyonnoj generatsii sluchajnykh sotsial'nykh grafov [On a model of social network with user communities for distributed generation of random social graphs]. 10-ya Mezhdunarodnaya konferentsiya «Intellektualizatsiya obrabotki informatsii» [10th International conference “Intelligent Information Processing”] 2014.