Высокопроизводительные вычисления в облачных системах с использованием OpenFlow*

П.Н. Полежаев

Оренбургский государственный университет

В настоящей работе описывается подход к созданию грид-системы, функционирующей поверх вычислительной облачной системы, построенной на базе OpenStack и программно-конфигурируемых OpenFlow сетей. Предложена архитектура облачного вычислительного центра обработки данных с поддержкой OpenFlow, разработаны алгоритмы назначения групп виртуальных машин с учетом состояния физической сети облачного центра обработки данных.

1. Введение

Облачные вычислительные системы стали стандартом де факто для многих прогрессивных областей сферы информационных технологий. Российские компании и университеты используют их с целью размещения своих научных и бизнес-приложений, что позволяет им избежать расходов на создание и поддержание функционирования собственных центров обработки данных (ЦОД). С другой стороны, владельцы облачных вычислительных ЦОД путем консолидации вычислительных ресурсов и систем хранения данных (СХД) способны снизить совокупную стоимость владения ИТ-инфраструктурой за счет обслуживания значительного количества клиентов, а также использования эффективных технических средств планирования и балансировки нагрузки, управления пересылкой данных в сети, интеграции нескольких территориально разрозненных сегментов ЦОД.

В последнее время облачные системы стали использоваться в качестве основы для высокопроизводительных систем (High performance computing, HPC), включая вычислительные кластеры и грид-системы. Это стало возможным за счет развития средств виртуализации ресурсов процессора, памяти, а также ввода-вывода.

В настоящей работе предлагается подход к улучшению существующей облачной системы OpenStack [1] за счет интеграции инфраструктурных средств грид-системы, облачной системы OpenStack и программно-конфигурируемых сетей (ПКС) сегментов ЦОД на основе OpenFlow [2].

2. Облачная система OpenStack

Наиболее распространенным открытым программным обеспечением для управления облачными вычислительными ЦОД является OpenStack. Его популярность, прежде всего, обусловлена, открытостью, документированностью, наличием исходных кодов, а, следовательно, возможностью адаптации его компонент под произвольные архитектуры вычислительных систем.

OpenStack позволяет создавать многоарендные архитектуры IaaS облачных вычислительных ЦОД, когда вычислительные и сетевые ресурсы разделяются между несколькими пользователями-арендаторами. Каждому арендатору выделяется виртуализованные ресурсы в виде группы виртуальных машин, связанных виртуальной сетью. Пользователь вправе сам настраивать конфигурацию виртуальных машин, включая параметры процессора, памяти, загружаемый

* Исследования поддержаны федеральной целевой программой «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007–2013 годы» (госконтракт №07.514.11.4153), федеральной целевой программой «Научные и научно-педагогические кадры инновационной России на 2009-2013 годы» (соглашение №14.B37.21.1881) и РФФИ (проект №12-07-31089).

образ с операционной системой, программами и данными. Хранение образов виртуальных машин обеспечивается компонентой Glance, а данных – облачным хранилищем Swift.

OpenStack имеет следующие недостатки:

– Отсутствие эффективных средств управления потоками данных в используемой компьютерной сети. Как правило, применяются стандартные протоколы маршрутизации, такие, как RIP, OSPF, EIGRP и др., которые опираются на распределенные алгоритмы выбора маршрута, без учета логических путей передачи (потоков) данных, существующих на уровне приложений (в том числе параллельных программ, выполняющихся в HPC системе). Данные протоколы поддерживаются существующими коммутаторами без ПКС.

– Использование неэффективного алгоритма планирования экземпляров виртуальных машин. Он заключается в следующем: для каждой планируемой виртуальной машины из нескольких одинаковых экземпляров, сначала вычислительные узлы фильтруются в соответствии с ресурсными требованиями виртуальной машины, затем они сортируются в порядке возрастания значения интегрального критерия, представляющего собой линейную свертку характеристик вычислительного узла, после чего выбирается необходимое количество узлов. Данный алгоритм не учитывает топологию сети, ее состояние в текущей момент времени, а также коммуникационные схемы передачи данных между виртуальными машинами.

– Отсутствие встроенных средств для эффективной реализации HPC поверх облака. OpenStack позволяет размещать виртуальные кластеры и грид-системы в группах экземпляров виртуальных машин, функционирующих в облаке. При этом облако не имеет информации о том, что запущено внутри него, включая сведения о выполняемых параллельных программах, а HPC-система не обладает сведениями о состоянии физической сети облака. С одной стороны, это обеспечивает абстракцию и универсальность облака, а с другой, отрицательно сказывается на производительности HPC-системы, т.к. планировщик задач в HPC-системе способен эффективнее назначать вычислительные задачи (особенно коммуникационно-интенсивные) при наличии информации о топологии и состоянии сети облака.

OpenStack был выбран в качестве основы для построения HPC систем, функционирующих поверх облака. Разделы 3 и 4 раскрывают предложенные автором подходы к устранению данных недостатков. В разделе 5 описывается предложенная архитектура облачного вычислительного ЦОД с поддержкой OpenFlow.

3. Разработанные алгоритмы назначения групп виртуальных машин

Обозначим в качестве Φ_{assign} функцию, назначающую каждой группе виртуальных машин $VMGroup_j$ набор вычислительных узлов

$$\Phi_{assign}(VMGroup_j) = (Node_{k_1 i_1}, Node_{k_2 i_2}, ..., Node_{k_{g_j} i_{g_j}}),$$

где $Node_{k_r i_r}$ – узел для назначения виртуальной машины VM_{jr}. Условно зададим $Node_{k_r i_r} = \varphi_{assign}(VM_{jr})$.

Для оценки качества алгоритма назначения групп экземпляров виртуальных машин Φ_{assign} может использоваться интегральный критерий оценки размещения отдельной виртуальной машины $VMGroup_j$:

$$I(\Phi_{assign}, VMGroup_j, DataCenter) \to min,$$

где $DataCenter$ – мультиграф, описывающий весь вычислительный ЦОД.

Пусть $dist_{phys}(d_1, d_2)$ – топологическое расстояние между физическими сетевыми устройствами $d_1 \in Devices$ и $d_2 \in Devices$ в облачной системе. Как правило, в качестве топологического расстояние берется оценка суммарной задержки времени передачи пакета с учетом состояния сети на текущий момент времени. Для двух физических узлов внутри одного сегмента топологическое расстояние будет гораздо меньше, чем расстояние для двух узлов внутри разных, территориально удаленных сегментов.

В качестве интегрального критерия могут быть использованы:

1) суммарное попарное топологическое расстояние между назначенными узлами для группы экземпляров виртуальных машин

$$I_{SDMA}(\Phi_{assign}, VMGroup_j, DataCenter) = \sum_{VM_{jr_1}, VM_{jr2} \in VMs_j} dist_{phys}(\varphi_{assign}(VM_{jr_1}), \varphi_{assign}(VM_{jr_2})) \to min;$$

2) максимальное топологическое расстояние между двумя назначенными узлами для группы экземпляров виртуальных машин

$$I_{MDMA}(\Phi_{assign}, VMGroup_j, DataCenter) =$$

$$= \max_{VM_{jr_1}, VM_{jr2} \in VMs_j} dist_{phys}(\varphi_{assign}(VM_{jr_1}), \varphi_{assign}(VM_{jr_2})) \to min.$$

Опишем общую схему предложенных алгоритмов назначения групп экземпляров виртуальных машин.

Входной параметр: $VMGroup_j$ – назначаемая группа экземпляров виртуальных машин.

Выходной параметр: $R = (Node_{k_1 i_1}, Node_{k_2 i_2}, ..., Node_{k_{g_j} i_{g_j}})$ – набор назначенных физических вычислительных узлов.

1. С помощью алгоритма Флойда-Уоршелла вычисляется топологическое расстояние $dist_{phys}(d_1, d_2)$ между всеми парами физических сетевых устройств $d_1, d_2 \in Devices$ облачной системы.

2. Для каждого физического сетевого устройства $d \in Devices$ облачной системы выполняются следующие действия:

2.1. Пока имеются не назначенные экземпляры виртуальных машин, перечисляются все физические узлы $Node_{k_r i_r} \in Devices$ в порядке возрастания величины $dist_{phys}(d, Node_{k_r i_r})$, при равных значениях расстояния – упорядочиваются по производительности.

2.2. Формируется список экземпляров виртуальных машин L группы $VMGroup_j$, ресурсные требования которых способен удовлетворить узел $Node_{k_r i_r}$.

2.3. Из списка L выбирается экземпляр виртуальной машины VM_{jr}, требования которого наиболее близки к характеристикам физического узла:

$$VM_{jr} = arg \min_{VM_{jr} \in L} Similarty(VM_{jr}, Node_{k_r i_r}).$$

Функция схожести представляет собой линейную свертку разностей характеристик узла $Node_{k_r i_r}$ и соответствующих величин запрашиваемых VM_{jr} ресурсов, взятых с некоторыми весовыми значениями.

2.4. За экземпляром виртуальной машины VM_{jr} закрепляется физический вычислительный узел $Node_{k_r i_r}$, задается $\varphi^d_{assign}(VM_{jr}) = Node_{k_r i_r}$.

2.5. В конце для сформированного назначения вычисляется значение интегрального критерия I^d.

3. В качестве результата R выбирается набор $(\varphi^{d_{min}}_{assign}(VM_{j1}), \varphi^{d_{min}}_{assign}(VM_{j2}), ..., \varphi^{d_{min}}_{assign}(VM_{jg_j}))$, который для первоначального физического сетевого устройства d_{min} обеспечивает минимум $I^{d_{min}}$ интегрального критерия:

$$I^{d_{min}} = \min_{d \in Devices} I^d.$$

4. Набор назначенных физических вычислительных узлов R передается подсистеме маршрутизации трафика облачной системы для реализации схемы проактивной маршрутизации всевозможными узлами.

В зависимости от используемого интегрального критерия можно получить различные алгоритмы назначения групп виртуальных машин. В качестве критериев были выбраны формулы

I_{SDMA} и I_{MDMA}. Данные алгоритмы были соответственно названы Summed Distance Minimization Assignment и Maximum Distance Minimization Assignment. В силу наличия аналогии между группой виртуальных машин и параллельной программой с несколькими процессами, данные алгоритмы являются аналогами ранее предложенных автором методов назначения параллельных программ для вычислительной грид-системы [3].

Результаты исследования данных алгоритмах будут опубликованы позднее, после создания облачного вычислительного ЦОД с поддержкой OpenFlow в Оренбургском государственном университете.

4. Предложенный подход к организации HPC поверх облачных центров обработки данных с поддержкой OpenFlow

Для организации HPC поверх облачных центров был выбран подход, заключающийся в создании неавтономной грид-системы, функционирующей поверх вычислительного облака. Опишем основные преимущества и особенности реализации данного подхода.

Грид-система представляет собой совокупность вычислительных кластеров, каждый из которых развернут внутри группы экземпляров виртуальных машин в отдельном сегменте ЦОД, а также экземпляра виртуальной машины грид-диспетчера, который поддерживает единую очередь параллельных вычислительных задач.

Грид-система и облачная система не изолированы полностью друг от друга, как это характерно для обычных подходов к реализации HPC поверх облаков. Облачная система предоставляет планировщику грид-системы актуальную информацию о состоянии сети облака (оценку задержек при передаче пакетов между виртуальными вычислительными узлами), а грид-система передает информацию подсистеме маршрутизации облака о выполненных назначения процессов параллельных задач и их коммуникационные схемы.

Такой осознанный подход к снижению уровня абстракции и универсализации облачной системы позволяет, с одной стороны, планировщику грида, обладающему актуальной информацией о высокопроизводительной сети, эффективно и локализовано назначать процессы параллельных задач. А с другой, подсистема маршрутизации облачной системы может использовать проактивную схему маршрутизации. Данная схема заключается в том, что, когда известны шаблоны коммуникации (например, виртуальные топологии MPI-2 для параллельных задач) между сетевыми устройствами, то маршруты передачи данных между ними можно проложить заранее, до начала коммуникаций. Это позволяет избежать задержки обработки первого пакета, которая характерна для реактивной схемы, в которой при получении первого пакета каждого нового потока маршрут прокладывает контроллер и устанавливает соответствующие правила во все OpenFlow-коммутаторы.

При использовании обеих схем имеет место проблема ограниченности размера таблицы записей о потоках OpenFlow-коммутаторов. В случае реактивной схемы она решается путем установки каждого правила коммутации на некоторый интервал времени, после оно автоматически удаляется из таблицы. В случая проактивной схемы предложен вариант, в котором агрегируются потоки между процессами параллельных задач, при условии, что процессы запущены в одних и тех же виртуальных машинах.

Все описанное выше позволяет снизить среднее время выполнения коммуникационно-интенсивных задач, а также увеличить среднюю загрузку, как грид-системы, так и всей облачной системы.

В разрабатываемой облачной системе планируется использовать смешенную схему маршрутизации: проактивную – для передачи данных между процессами параллельных программ и реактивную – для управления инфраструктурными потоками в облаке, а также при чтении или записи данных в СХД.

В качестве алгоритмов планирования параллельных задач в грид-системе планируется использовать модифицированные варианты ранее разработанных автором алгоритмов Backfill SDM и Backfill MDM [3].

5. Архитектура облачного вычислительного центра обработки данных с поддержкой OpenFlow

Предложена архитектура облачного вычислительного ЦОД с поддержкой OpenFlow. ЦОД состоит из нескольких территориально-распределенных сегментов (автономных систем), каждый из которых представляет собой ПКС на основе OpenFlow. Сегменты связаны через глобальную сеть Интернет с помощью протокола BGP, точками подключения сегментов к глобальной сети являются граничные шлюзы, обладающие сведениями о префиксах других сегментов. Сеть ПКС сегмента содержит, как OpenFlow-коммутаторы, так и обычные коммутаторы без поддержки OpenFlow, а также вычислительные узлы на которые назначаются виртуальные машины групп экземпляров виртуальных машин.

Также каждый сегмент ЦОД содержит систему управления OpenStack, включающий: планировщик OpenStack (решает инфраструктурные задачи по запуску, остановке и контролю выполнения экземпляров виртуальных машин), разработанный модуль диспетчера распределения нагрузки (реализует предложенные алгоритмы назначения групп виртуальных машин) и сетевой модуль OpenStack (модифицированный для взаимодействия с HPC системой).

Рис. 1 более детально раскрывает структуру одного сегмента облачного вычислительного ЦОД.

С целью обеспечения надежности и масштабируемости в каждом сегменте функционирует несколько экземпляров контроллера OpenFlow, каждый из них имеет свой экземпляр базы данных кластерной СУБД MySQL, хранящей текущее состояние сегмента.

Прежде всего это ориентированный взвешенный мультиграф сегмента, вершины – сетевые устройства, дуги – сетевые связи (гарантируется существование противоположной дуги). Веса вершин – статические параметры и динамические характеристики соответствующих устройств (например, для вычислительных узлов – максимальный размер оперативной и локальной дисковой памяти, количество вычислительных ядер, текущие объемы свободной оперативной и дисковой памяти, загрузку каждого вычислительного ядра), веса дуг – их максимальная пропускная способность и текущая загрузка.

Имеется несколько систем распределения нагрузки, одна из них активная, остальные – находятся в резерве. Активный экземпляр осуществляет назначение OpenFlow-коммутаторов на контроллеры, руководствуясь заложенной политикой выравнивания нагрузки. Таким образом, каждый OpenFlow-коммутатор подключен ровно к одному контроллеру.

Согласно спецификации OpenFlow существует следующий принцип функционирования OpenFlow-коммутаторов. Каждый коммутатор имеет таблицу потоков, состоящую из записей о потоках (правил). Каждая запись включает: шаблоны заголовков пакета (match) – часть, по которой определяется, применимо ли данное правило к обрабатываемому пакету, действия (actions) – набор операций, совершаемых над пакетом (передача в заданный порт, помещение в заданную очередь, ассоциированную с конкретным портом коммутатора, изменение конкретного заголовка и др.), а также статистические счетчики.

При поступлении пакета, коммутатор ищет подходящие для него правило в таблице потоков, руководствуясь match-частями. Если правило найдено, то над пакетом выполняются соответствующие действия, после чего обновляются значения счетчиков. Если же правило отсутствует, то пакет пересылается контроллеру OpenFlow. Контроллер анализирует его заголовки, после чего устанавливает в данный и возможно в другие, подключенные в него, коммутаторы правила, для обработки подобных пакетов.

OpenFlow позволяет упростить коммутатор и вынести логику управления в отдельный контроллер. В рамках предложенного подхода модифицирован контроллер OpenFlow. Реализованы эффективные алгоритмы маршрутизации потоков данных с учетом текущего состояния сети ЦОД и информации о параллельных программах, исполняющихся поверх вычислительного ЦОД.

Система сбора статистики через регулярные небольшие интервалы времени с помощью SNMP и OpenFlow собирает сведения о текущем состоянии сетевых устройств и связей. Полученные данные сохраняются в локальную копию базы данных кластерной СУБД.

Хранение данных в облаке реализовано на базе службы сетевого хранилища Swift.

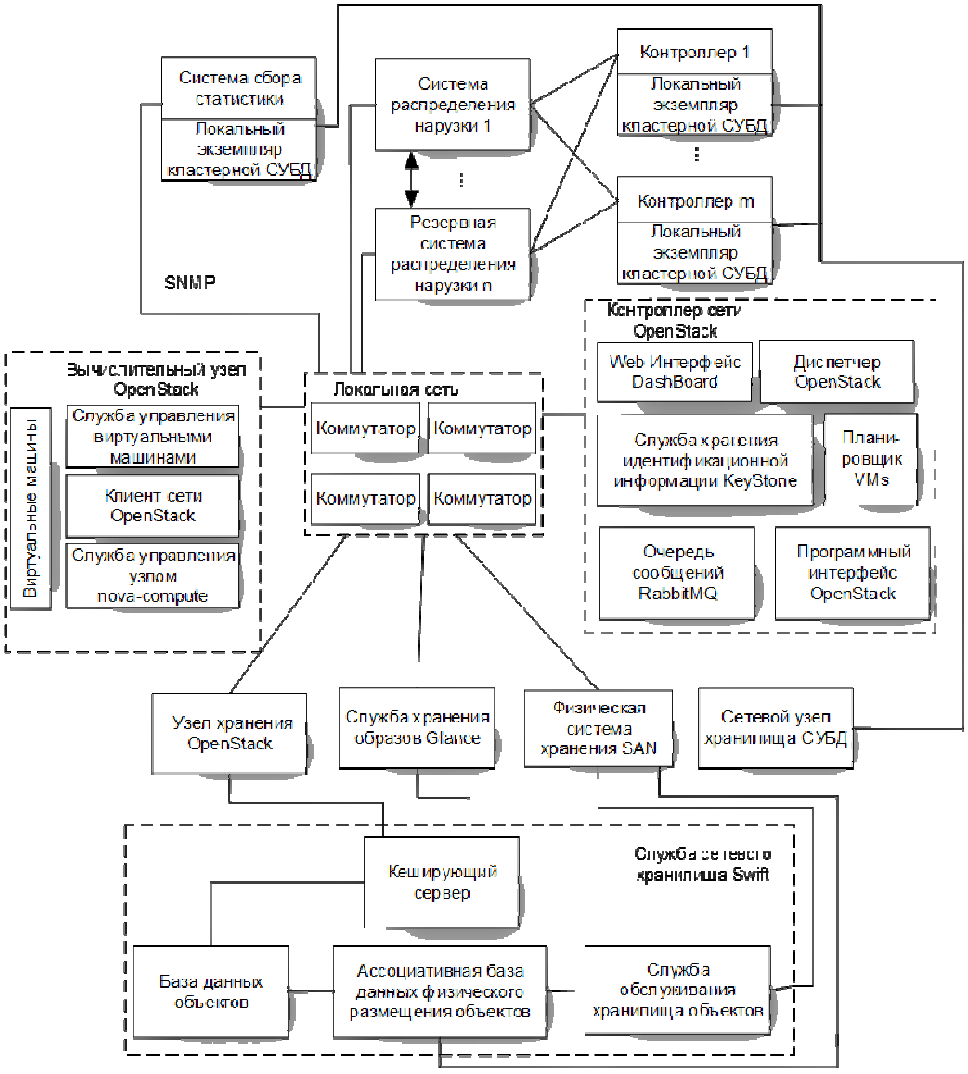

Рис. 1. Структура сегмента облачного вычислительного ЦОД с поддержкой OpenFlow

6. Заключение

Предложен подход к организации высокопроизводительных вычислений поверх вычислительного облака OpenStack с поддержкой OpenFlow. Описана архитектура облачного ЦОД с поддержкой его сегментами OpenFlow. Разработаны алгоритмы назначения групп виртуальных машин с учетом состояния физической сети облачного ЦОД.

В Оренбургском государственном университете находится в процессе создания экспериментальный образец облачного ЦОД с поддержкой OpenFlow его сегментами. В будущем планируется экспериментальное исследование всех разработанных алгоритмов и используемых технических решений.

Литература

1. OpenStack Open Source Cloud Computing Software. http://www.openstack.org/ (дата обращения: 20.11.2012).
2. Портал OpenFlow. http://www.openflow.org/ (дата обращения: 20.11.2012).
3. Полежаев П.Н. Экспериментальное исследование алгоритмов планирования задач для грид-систем с использованием симулятора // Труды международной конференции «Высокопроизводительные вычисления HPC-UA’2012» (г. Киев), 2012. С. 278-284.