Разработка и реализация облачного планировщика, учитывающего топологию коммуникационной среды при высокопроизводительных вычислениях[1]

И.А. Дудина, А.О. Кудрявцев, С.С. Гайсарян

Аннотация. Для эффективного решения высокопроизводительных задач в облаке необходимо планирование виртуальных машин на серверах, учитывающее особенности высокопроизводительных вычислений, чтобы уменьшить потери производительности, связанные с задержками в сети. В данной работе рассматривается подход к планированию, основанный на оценке размещения с помощью Hop-Byte метрики. Для конкретного варианта коммуникационной среды (сети Infiniband с топологией «толстое дерево») производится подсчет Hop-Byte метрики в предположении, что взаимодействие между виртуальными машинами одинаково. На основе полученных результатов разработан алгоритм планирования, выполнена его реализация в облачной системе OpenStack. Показано, что для отдельных тестов удается получить прирост производительности, при этом существенно снижается разброс значений при повторных запусках.

Ключевые слова: планирование, виртуализация, облачные вычисления, hop-byte метрика

Введение

Согласно проведенному министерством энергетики США исследованию Magellan User Survey [1] среди ученых, использующих в своей деятельности высокопроизводительные вычисления, возрастает интерес к облачным сервисам. Причиной тому является целый ряд уникальных преимуществ, присущих облачным вычислениям. К наиболее значительным из них можно отнести экономию от масштаба, гибкие условия аренды ресурсов, простоту их приобретения и использования,

[1] Работа выполнена при финансовой поддержке Минобрнауки России по государственному контракту от 15.06.2012 г. № 07.524.11.4018 в рамках ФЦП «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы»

наличие у пользователей возможности самостоятельного управления ПО, а также удобный пользовательский интерфейс и возможность создания научного портала для организации совместной работы с коллегами. Однако ключевым препятствием для использования облачных систем в этой области является падение производительности приложений в облачной среде. Некоторые успехи в решении этой проблемы уже достигнуты, в частности уменьшение падения производительности, вызванного накладными расходами на виртуализацию [2].

Одним из факторов, ограничивающих производительность приложений не только в облачных системах, но и на суперкомпьютерах, является пропускная способность сети, обеспечивающей взаимодействие между процессами в рамках одной задачи. В то время как собственно вычисления дешевеют, узким местом при масштабировании параллельных приложений является взаимодействие между процессами. Учитывая особенности высокопроизводительных вычислений, даже незначительное снижение производительности может привести к сильному ухудшению при масштабировании числа узлов. Таким образом, для повышения производительности необходимо как можно более эффективно использовать ресурсы сети.

В настоящее время существует несколько облачных платформ, как закрытых, так и с открытым исходным кодом. К закрытым относятся: Amazon EC2, Microsoft Azure, Google Compute Engine. Среди открытых платформ наиболее известны системы OpenNebula, OpenStack, Nimbus, Eucalyptus. Наиболее активно развивающейся из них является OpenStack – комплекс проектов свободного программного обеспечения для создания вычислительных облаков и облачных хранилищ (распространяется под лицензией Apache License 2.0).

Целью данной работы являлась разработка алгоритма планировщика OpenStack для сокращения падения производительности, вызванного задержками сети. Одним из компонентов OpenStack является контроллер вычислительных ресурсов Nova, частью которого является планировщик, отвечающий за выбор узлов для запуска виртуальных машин. Задача заключается в том, чтобы модифицировать имеющийся планировщик с учетом требований для высокопроизводительных вычислений.

1 Обзор работ

В работе [3] были предприняты попытки повышения эффективности использования сети для высокопроизводительных вычислений. Здесь рассматриваются сети InfiniBand с иерархической структурой, организованные в несколько уровней с помощью коммутаторов. Идея оптимизации основана на том, что время на передачу сообщения тем меньше, чем меньше коммутаторов должно оно пройти. Разработанный

метод заключается в определении топологии сети и использовании полученных данных для оптимизации работы коллективных операций. Авторы проанализировали производительность операций MPI Gather и MPI Scatter на таких системах и разработали для них эффективный алгоритм, учитывающий топологию сети. Увеличение производительности на уровне микробенчмарков при использовании этих алгоритмов достигло почти 54%.

Продолжением этого исследования стала работа [4]. Авторы рассматривают случай сети, неоднородной по скорости, и предлагают учитывать это свойство сети наряду с топологией для оптимизации коллективных операций. В результате этого исследования был разработан фреймворк, автоматически определяющий топологию сети и скорость между узлами и предоставляющий эти данные пользователю. Кроме того, были внесены изменения в библиотеку MPI: добавлены возможности динамического получения информации о топологии и скорости сети, а также построения модели сети на основе этих данных; модифицирован алгоритм вещания для оптимизации с учетом имеющейся модели. Эксперименты показали, что для однородной по скорости сети и сообщений большого размера увеличение производительности коллективных операции на уровне микробенчмарков достигает 14%. На уровне приложения использование фреймворка позволяет сократить общее время работы почти на 8%, особенно при увеличении размера задачи. Увеличение производительности сетевых алгоритмов на неоднородной по скорости сети достигает 70%-100%.

Диссертация A. Bhatele [5] посвящена автоматическому размещению процессов на узлах суперкомпьютера с учетом топологии сети. В этой работе предлагается использовать для оценки загруженности сети hop-byte метрику вместо диаметра графа. Показано, что эта метрика более адекватна для поставленной задачи, чем метрики, использованные ранее другими авторами. Разработан фреймворк, позволяющий автоматически размещать процессы на узлах с учетом топологии. В первую очередь происходит определение графа коммуникаций и выделение повторяющихся образцов. Далее для регулярных и нерегулярных графов применяются эвристики, позволяющие найти удачное размещение с низким значением hop-byte метрики. Кроме того, в данной работе рассматриваются перспективы виртуализации высокопроизводительных вычислений, в частности, метод балансировки нагрузки с учетом топологии сети.

К сожалению, задача оптимизации hop-byte метрики является NP-трудной, т.к. сводится к квадратичной задаче о назначениях. В статье [6] приводится эвристика для этой задачи, позволяющая сократить значение этой метрики. Использование эвристики в библиотеке MPI, разрабатываемой фирмой Cray, позволило сократить значение метрики на 75%. Однако временная сложность этой эвристики выше по

сравнению с другими эвристиками для этой задачи, поэтому данный алгоритм разумно использовать в случае статичных, пусть и нерегулярных графов топологии и коммуникаций задачи.

Попытки доработать планировщик Openstack для высокопроизводительных вычислений были предприняты в статье [7]. Первая идея, реализованная в этой работе, заключается в том, что планировщик должен рассматривать размещение N процессов одной задачи на N узлах как единый запрос, а не N последовательных независимых запросов. Это позволяет сразу выбрать группу наиболее подходящих для этой задачи узлов. Далее, решая задачу размещения для группы из N сильно связанных процессов, планировщик выбирает наиболее незагруженную стойку, а в ней отдает предпочтение узлам с наименьшим количеством уже запущенных виртуальных машин. Вторая идея основана на том, что в высокопроизводительных задачах вычисления часто прерываются коллективными операциями, т.е. происходит периодическая синхронизация процессов. Вследствие этого все процессы выполняются со скоростью самого “медленного”. Поэтому для эффективного использования ресурсов разумно размещать всю группу из N процессов на одинаковых серверах.

2 Существующие возможности системы Openstack

Начиная с версии Essex, в OpenStack по умолчанию используется так называемый «фильтрующий» планировщик. Получая запрос на размещение нескольких виртуальных машин, он принимает решение о размещении на конкретных серверах в два этапа.

2.1 *Этап фильтрации*

Первый этап заключается в отсеивании всех серверов, не удовлетворяющих требованиям запроса. Для этого к списку серверов применяется набор фильтров, определенный в конфигурационном файле Nova. Каждый фильтр отсеивает сервера, не удовлетворяющие определенному требованию, и в результате для дальнейшего рассмотрения формируется список серверов, прошедших через все фильтры. Так, например, **ram_filter** исключает из рассмотрения сервера, не обладающие достаточной памятью, **disk_filter** — местом на дисковом пространстве и т. д.

Стоит отметить, что в случае высокопроизводительных вычислений для минимизации накладных расходов разумно размещать на каждом сервере не более одной виртуальной машины и предоставлять ей все ресурсы этого сервера. В нашей задаче будем считать, что среди прошедших фильтрацию серверов останутся только полностью незанятые.

2.2 *Этап взвешивания*

К началу этапа взвешивания планировщик имеет список серверов, на каждом из которых можно разместить запускаемые ВМ. На этом этапе необходимо среди всех допустимых серверов найти наиболее подходящий. Сравнение различных вариантов размещения необходимо проводить по многим факторам одновременно. Для этого каждому из факторов ставится в соответствие оценочная функция, возвращающая численное значение для рассматриваемого сервера, а также коэффициент, отражающий значимость этого фактора. Таким образом вычисляется вес сервера, который равен сумме произведений значений оценочных функций на соответствующие им коэффициенты:

$$W_{h_i} = \sum_{j=1}^{m} c_j f_j(h_i),$$

где h_i – i-ый сервер, W_{h_i} – его вес, f_j – оценочная функция, c_j – коэффициент, m – количество оценочных функций.

Далее серверы сортируются по весу, и для размещения очередной ВМ выбирается сервер с наибольшим весом. После этого этапы фильтрации и взвешивания повторяются, пока не будут размещены все ВМ из запроса [8].

3 Планирование высокопроизводительных задач

Задача планирования распределенных (в том числе параллельных) программ в рамках вычислительного кластера в целом аналогична задаче распределения ВМ по узлам кластера, при условии, что в виртуальных машинах выполняется какая-либо распределенная программа. В данном разделе описывается концепция групп ВМ, аналогичная набору процессов одного приложения. К задаче планирования групп ВМ применимы подходы, разработанные для планирования процессов распределенной программы. Описывается применение подхода, основанного на Hop-Byte метрике.

3.1 *Группы ВМ*

Для того чтобы при планировании учесть взаимодействие ВМ, планировщик должен решать проблему размещения ВМ, решающих конкретную задачу, ориентируясь на положение других ВМ этой задачи. Для того чтобы логически объединить виртуальные машины одной задачи, была введена концепция групп виртуальных машин в OpenStack. Каждая виртуальная машина может принадлежать какой-то группе (соответствующей определенной высокопроизводительной задаче). Таким образом, цель планировщика – разместить виртуальные машины

наиболее “компактно” – может быть поставлена в рамках одной группы виртуальных машин. Пришло время сформулировать критерий “компактности”, т.е. удачности конкретного варианта размещения виртуальных машин одной группы на определенных серверах.

3.2 Hop-Byte метрика

Эффективность конкретного варианта размещения процессов данного приложения на узлах сети зависит от двух параметров: топологии сети и характера взаимодействия процессов задачи между собой. Для оценки этой величины была предложена Hop-Byte (HB) метрика. В данной модели рассматриваются два графа.

Первый взвешенный неориентированный граф $G = (V, E)$ отражает топологию сети. Вершины данного графа соответствуют серверам. Каждое ребро между двумя вершинами i, j имеет вес e_{ij}, равный количеству скачков между двумя серверами, соответствующими данным вершинам (количество скачков равно количеству коммутаторов, которое проходит сообщение при передаче от одного узла к другому).

Второй взвешенный граф $G' = (V', E')$ отражает взаимодействие между процессами. Вершины этого графа соответствуют процессам рассматриваемого приложения. Ребро между двумя вершинами i, j имеет вес e'_{ij}, равный суммарному объему сообщений, посылаемых между этими вершинами в обоих направлениях.

Рассмотрим конкретное размещение процессов из V' на серверах из V:

$$x_{hp} = \begin{cases} 1, \text{если процесс } p \text{ размещен на сервере } h \\ 0, \quad \text{в противном случае} \end{cases}, \qquad h \in V, p \in V';$$

$$HB = \sum_{i,j} \sum_{k,l} e_{ik} e'_{jl} x_{ij} x_{kl};$$

$$\sum_i x_{ij} = 1, \text{для всех } j \in V', \qquad \sum_j x_{ij} = 1, \text{для всех } i \in V.$$

Полученное значение **HB** и является показателем, отражающим успешность данного расположения процессов на узлах. Доказано, что задача минимизации этой величины является NP-трудной, т.к. сводится к квадратичной задаче о назначениях. Кроме этого, заранее о приложениях пользователей ничего неизвестно, поэтому необходимо прилагать дополнительные усилия для выяснения значений e'_{ij}.

Для решения поставленной задачи за приемлемое время можно её упростить, сделав некоторые дополнительные предположения. Во-

первых, при разработке алгоритма можно ориентироваться на какой-то конкретный тип топологии сети (предположения относительно графа $\boldsymbol{G}$ в нашей постановке). Во-вторых, можно отказаться от решения дополнительной весьма сложной задачи по выяснению значений e'_{ij}, приняв их за одинаковые константы (это соответствует ситуации, в которой все процессы равноправны и взаимодействуют между собой с одинаковой интенсивностью).

3.3 *Топология сети*

Мы рассматриваем топологию сети Infiniband «Fat tree», т.к. она является одной из распространенных топологий для высокопроизводительных задач. Такая сеть представляет собой дерево, листьями которого являются вычислительные узлы, а внутренними вершинами – коммутаторы (см. рис. 1).

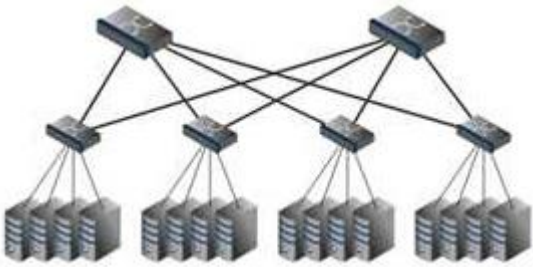

Рис. 1. Топология Fat Tree

У коммутаторов более высоких уровней пропускная способность каналов больше, т.е. связи с другими вершинами более «толстые». Поэтому эта топология названа «толстое дерево»

3.4 *Алгоритм планирования*

С учетом сделанных предположений о равномерности коммуникаций между процессами и топологии сети можно предложить алгоритм планирования, ориентированный на минимизацию HB-метрики для каждой из групп виртуальных машин.

3.4.1 Применяемая эвристика

Из предположения о равномерном взаимодействии процессов одной задачи следует, что на значение HB-метрики влияет только суммарное значение скачков между всеми парами процессов. С другой стороны, количество скачков между процессами любой пары зависит только от того, с каким из коммутаторов нижнего уровня связан каждый из двух серверов, размещающих рассматриваемые процессы. Для удобства

можно объединить все сервера, связанные с определенным коммутатором нижнего уровня в один “пучок” (см. Рис. 2).

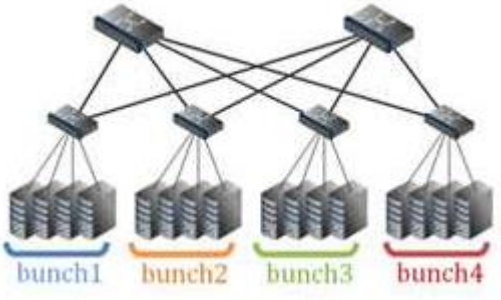

Рис. 2. Объединение серверов в пучки

Очевидно, что количество скачков между любой парой серверов зависит только от того, принадлежат ли эти серверы одному пучку. В связи с этим в планировщик была добавлена новая сущность, хранящая информацию о пучке, принадлежащих ему серверах, о числе виртуальных машин каждой группы, уже запущенных на этих серверах и т.д.

Итак, пусть планировщику поступил запрос на планирование N виртуальных машин одной группы, причем ранее уже запущено M машин этой группы. Задача планировщика разместить N машин так, чтобы значение HB-метрики для $(M+N)$ машин было как можно меньше. Пусть есть P пучков, и на них соответственно размещено $n_1, n_2, \dots, n_P$ машин (см. Рис. 3):

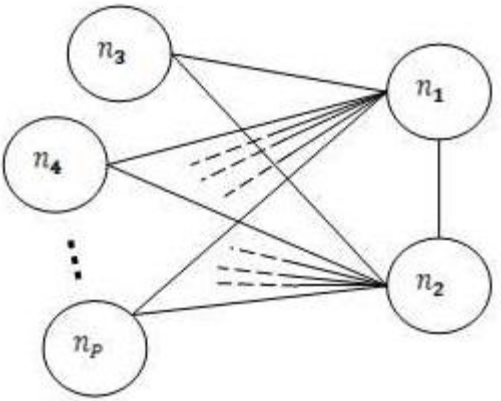

Рис. 3. Подсчет HB-метрики

$$\sum_{i=1}^{P} n_i = M + N.$$

Без ограничения общности выделим первые два пучка и посчитаем значение HB-метрики. Для начала посчитаем вклад взаимодействия внутри пучков (каждая связь – один скачок):

$$HB_{in} = \sum_{\substack{i=1..P,\\ n_i>0}} \frac{n_i(n_i-1)}{2}.$$

Пусть $n_1 + n_2 = s,\ s > 0$. Теперь посчитаем вклад взаимодействия виртуальных машин между пучками (каждое взаимодействие происходит с тремя скачками):

$$\begin{aligned} HB_{out} &= 3\sum_{\substack{i,j=1..P,\\ i\neq j}} n_i n_j \\ &= 3\Big(\sum_{\substack{i,j=3..P,\\ i\neq j}} n_i n_j + n_1(n_3 + \cdots + n_P) \\ &+ n_2(n_3 + \cdots + n_P) + n_1 n_2\Big) \\ &= 3\Big(\sum_{\substack{i,j=3..P,\\ i\neq j}} n_i n_j + s(n_3 + \cdots + n_P) + n_1 n_2\Big). \end{aligned}$$

Отсюда видно, что от распределения s виртуальных машин между первым и вторым пучком зависят только слагаемые:

$$\begin{aligned} &\frac{n_1(n_1-1)}{2} + \frac{n_2(n_2-1)}{2} + 3n_1 n_2 \\ &\quad = \frac{n_1(n_1-1)}{2} + \frac{(s-n_1)(s-n_1-1)}{2} \\ &\quad + 3n_1(s-n_1) = \end{aligned}$$

$$-2n_1^2 + 2sn_1 + \frac{s^2 - s}{2}, \text{где } n_1 > 0, n_2 > 0.$$

(В случае $n_2 = 0$ эта сумма становится одним слагаемым $\frac{n_1(n_1-1)}{2}$, случай $n_1 = 0$ аналогичен.)

Меняя значение n_1 можно перераспределить виртуальные машины между пучками так, чтобы минимизировать эту сумму. На **Рис. 4** показана зависимость значения этой суммы от значения n_1. Очевидно, что чем дальше n_1 (а, значит, и n_2) от $s/2$, тем меньше значение рассматриваемой суммы, и, следовательно, меньше значение HB-метрики в целом.

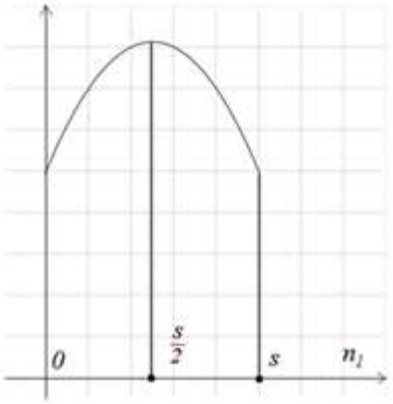

Рис. 4. Изменение значения НВ при перераспределении ВМ

Из этого можно можно предложить следующую стратегию: планировщик не должен заполнять пучки равномерно. Напротив, следует в первую очередь заполнять те из них, которые способны вместить наибольшее количество виртуальных машин данной группы (включая уже запущенные).

3.4.2 Взвешивающая функция

Было решено использовать имеющийся в OpenStack фильтрующий планировщик, дополнив его возможности, сохраняя общую логику планирования. Для того чтобы планировщик имел возможность учитывать особенности высокопроизводительных вычислений (а именно, высокие требования к производительности, зависящей от размещения виртуальных машин), была разработана взвешивающая функция, оценивающая размещение виртуальной машины определенной группы на отфильтрованных серверах-кандидатах. Гибкость данного подхода заключается в том, что он позволяет администратору изменять степень важности этого фактора, меняя коэффициент данной взвешивающей функции.

Значение оценочной функции для всех серверов одного пучка будет одинаковым. Принимая во внимание сделанные выше выводы, в первую очередь следует отсортировать пучки по максимальному числу виртуальных машин рассматриваемой группы, которое может оказаться в этом пучке после выполнения запроса к планировщику. Среди тех пучков, для которых это значение оказалось наибольшим, следует выбрать тот, в котором больше уже запущенных виртуальных машин этой группы. Для реализации сортировки сначала по первому, а затем по второму параметру, к значению второго параметра добавляется значение первого, умноженное на максимальное значение второго, увеличенное на единицу (см. *Листинг 1*).

```
first_ord = bunch.get_group_num(required_group) +
            min(bunch.free_hosts,
        num_instances_awaiting_shedule)
second_ord = bunch.get_group_num(required_group)
result = first_ord * (bunch.maxsize + 1) + second_ord
```

Листинг 1. Вычисление значения взвешивающей функции

4 Результаты тестирования

Для исследования влияния разработанного алгоритма планирования на производительность было проведено тестирование в два этапа. Во время первого запуска группы из 16 виртуальных машин планирование происходило средствами стандартного планировщика, и виртуальные машины были распределены произвольным образом между коммутаторами. Во время второго запуска планировщик учитывал разработанную взвешивающую функцию, и все виртуальные машины оказались в одном коммутаторе.

Исследование производительности было выполнено с использованием NAS Parallel Benchmarks. Это набор из 11 тестов производительности нацеленных на проверку возможностей вычислительной системы. Тесты разработаны и поддерживаются в NASA Advanced Supercomputing (NAS) Division, расположенном в NASA Ames Research Center [9].

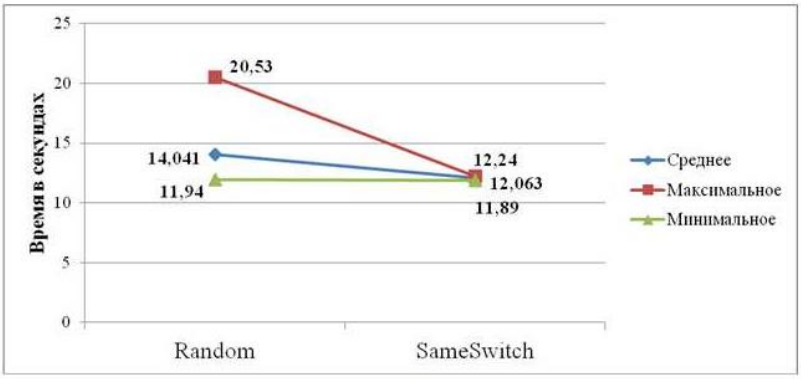

Рис. 5. Тест Block Tridiagonal.

Пакет тестов в обоих случаях запускался по 10 раз. Для 10 из 11 тестов не было выявлено существенных различий в производительности. Однако на тесте Block Tridiagonal среднее значение времени выполнения уменьшилось на 14% при планировании в рамках одного коммутатора, а максимальное значение уменьшилось на 40%. Амплитуда колебаний

производительности снизилась с 42% до 3%, т.е. колебания почти исчезли (см. Рис. 5).

Из этого можно сделать вывод, что разработанный алгоритм планирования для отдельных приложений позволяет предотвратить падение производительности, при этом производительность других приложений не ухудшается.

5 Выводы

Для эффективного решения высокопроизводительных задач в облаке необходимо планирование виртуальных машин на серверах, учитывающее особенности высокопроизводительных вычислений, чтобы уменьшить потери производительности, связанные с задержками в сети. В данной работе был рассмотрен подход к планированию, основанный на оценке размещения с помощью Hop-Byte метрики. Эта задача сводится к NP-трудной. Для конкретного варианта коммуникационной среды (сеть Infiniband с топологией «толстое дерево») был произведен подсчет Hop-Byte метрики в предположении, что взаимодействие между виртуальными машинами одинаково.

Основным вкладом данной работы является разработка алгоритма планирования на основе полученных результатов и его реализация в рамках общего подхода к планированию в OpenStack с использованием концепции групп виртуальных машин. Таким образом, в OpenStack была добавлена возможность планирования виртуальных машин с учетом требований высокопроизводительных вычислений.

Результаты проведенных исследований говорят о том, что разработанный в рамках планировщика OpenStack алгоритм планирования позволяет повысить производительность некоторых приложений. При этом падений производительности, связанных с использованием алгоритма, ни на каких тестах выявлено не было.

Список литературы

[1]. U.S. Department of Energy, «The Magellan Report on Cloud Computing for Science,» Chicago, 2011.

[2]. А. О. Кудрявцев, В. К. Кошелев и А. И. Аветисян, «Перспективы виртуализации высокопроизводительных систем архитектуры x64,» *Труды Института системного программирования РАН,* т. 22, pp. 189-210, 2012.

[3]. K. Kandalla, H. Subramoni, A. Vishnu и D. K. Panda, «Designing Topology-Aware Collective Communication Algorithms for Large Scale InfiniBand Clusters:Case Studies with Scatter and Gather,» в *The 10th Workshop on Communication Architecture for Clusters (CAC 10), Int'l Parallel and Distributed Processing Symposium (IPDPS 2010).*, Ohio, 2010.

[4]. H. Subramoni, K. Kandalla, J. Vienne, S. Sur, B. Barth, K. Tomko, R. McLay, K. Schulz и D. K. Panda, «Design and Evaluation of Network Topology-/Speed-Aware Broadcast Algorithms for InfiniBand Clusters,» 2012.

[5]. A. Bhatele, «Automatic Topology Aware Mapping For Supercomputers,» 2010.

[6]. C. D. Sudheer и A. Srinivasan, «Optimization of the Hop-Byte Metric for Effective Topology Aware Mapping,» 2012.
[7]. A. Gupta, D. Milojicic и L. V. Kalé, «Optimizing VM Placement for HPC in the Cloud,» в *Workshop on Cloud Services, Federation, and the 8th Open Cirrus Summit*, San Jose, CA, USA., 2012.
[8]. «Nova Developer guide. Filter Scheduler,» [В Интернете]. Available: http://docs.openstack.org/developer/nova/devref/filter_scheduler.html. [Дата обращения: 16 04 2013].
[9]. «Википедия,» [В Интернете]. Available: http://ru.wikipedia.org/wiki/NAS_Parallel_Benchmarks. [Дата обращения: 27 05 2013].

Topology-aware cloud scheduling for HPC

I.A. Dudina: eupharina@ispras.ru, ISP RAS, Moscow, Russia
A.O. Kudryavtsev: alexk@ispras.ru, ISP RAS, Moscow, Russia
S.S. Gaissaryan: ssg@ispras.ru, ISP RAS, Moscow, Russia

Abstract. For some compute intensive applications cloud computing can be a cost-effective alternative or an addition to supercomputers. However, in the case of high-performance computing, overall application performance depends heavily on how processes are mapped to the network nodes. Therefore a cloud scheduler must be topology-aware to reduce network congestion. In this paper the Hop-Byte metric for the case of "fat tree" network topology was evaluated under the assumption that all pairs of processes communicate evenly. We propose a scheduling algorithm that tries to minimize this metric, which was implemented atop the OpenStack scheduler. All instances are divided into groups according to compute intensive application they belong to. Every time the scheduler receives a request for launching N new instances of the same group, it maps them to the nodes in such a way that entire group (including already running instances) uses as few lower-level switches in “fat tree” as possible.

We measured the impact of topology-aware scheduling on the performance of NASA Advanced Supercomputing Parallel Benchmarks. Results of 10 of 11 benchmarks changed insignificantly. The average time of Block Tridiagonal test decreased by 14%, the maximum time decreased by 40% and the difference between the maximum time and the minimum time decreased from 42% to 3%, that is, fluctuation almost disappeared.

Keywords: scheduling, virtualization, cloud computing, hop-byte metric

References

[1]. U.S. Department of Energy. The Magellan Report on Cloud Computing for Science. Chicago, 2011.

[2]. Kudryavtsev A.O., Koshelev V.K. and Avetisyan A.I. Perspektivy virtualizatsii vysokoproizvoditel'nykh sistem arkhitektury x64 [The prospects for virtualization of high performance x64 systems] Trudy ISP RAN [The Proceedings of ISP RAS], 2012, vol. 22, pp. 189-210, (in Russian)

[3]. Kandalla K., Subramoni H., Vishnu A. and Panda D. K. Designing Topology-Aware Collective Communication Algorithms for Large Scale InfiniBand Clusters: Case Studies with Scatter and Gather. The 10th Workshop on Communication Architecture for Clusters (CAC 10), Int’l Parallel and Distributed Processing Symposium (IPDPS 2010), Ohio, 2010.

[4]. Subramoni H., Kandalla K., Vienne J., Sur S., Barth B., Tomko K., McLay R., Schulz K. and Panda D. K., Design and Evaluation of Network Topology-/Speed-Aware Broadcast Algorithms for InfiniBand Clusters, 2012.

[5]. Bhatele A., Automatic Topology Aware Mapping For Supercomputers, Graduate College of the University of Illinois at Urbana-Champaign, 2010.

[6]. Sudheer C. D. and Srinivasan A. Optimization of the Hop-Byte Metric for Effective Topology Aware Mapping. 19th International Conference on High Performance Computing, 2012
[7]. Gupta A., Milojicic D. and Kalé L. V., Optimizing VM Placement for HPC in the Cloud, Workshop on Cloud Services, Federation, and the 8th Open Cirrus Summit, San Jose, CA, USA., 2012.
[8]. Filter Scheduler. Nova Developer guide. Accessed: 16 Apr. 2013. http://docs.openstack.org/developer/nova/devref/filter_scheduler.html
[9]. NAS Parallel Benchmarks. Wikipedia. Accessed: 27 May 2013. http://ru.wikipedia.org/wiki/NAS_Parallel_Benchmarks