Архитектурно-зависимые решения задачи оптимизации запросов к базам данных в облачной инфраструктуре

Д. В. Леонов
Московский технологический институт «ВТУ»

Аннотация. Большинство методов оптимизации поиска, в общем, и запросов, в частности, так или иначе, являются аппаратно-зависимыми. Под архитектурно-зависимыми понимаются методы, жестко ориентированные и использующие свойства конкретных программно-аппаратных реализаций. Проведены анализ и оценка эффективности методов.
Ключевые слова: облачные вычисления, облачные инфраструктуры, оптимизация запросов, базы данных.

В последние несколько лет в области ИТ набирает популярность направление облачных вычислений (*Cloud Computing*) [1]. Крупнейшие мировые ИТ вендоры (Microsoft, Amazon, Google и прочие) так или иначе внедряют сервисы «облачных» вычислений. Сегодня под облачными вычислениями обычно понимают возможность получения необходимых вычислительных мощностей по запросу из сети, причем пользователю не важны детали реализации этого механизма и он получает из этого «облака» все необходимое.

В оптимизацию реляционных запросов входят два различных аспекта [2]. Во-первых, это — внутренняя задача СУБД, которая заключается в определении наиболее оптимального (эффективного) способа выполнения реляционных запросов. Во-вторых, это — задача программиста, которая заключается в написании таких реляционных запросов, для которых СУБД могла бы использовать более эффективные способы нахождения данных.

Большинство методов оптимизации поиска, в общем, и запросов в частности, так или иначе, являются аппаратно-зависимыми [3]. Под архитектурно-зависимыми мы понимаем методы, жестко ориентированные и использующие свойства конкретных программно-аппаратных реализаций.

Рассмотрим подход к оптимизации поиска в облачных базах данных, предложенный в [4]. Основной интерес указанного подхода заключается в том, что его целью является балансировка нагрузки на оборудование посредством миграции виртуальных машин в облачном окружении, что опосредованно приводит к повышению качества поиска (очевидно, что скорость, с которой «облако» отвечает на запросы пользователя, является одним из критериев качества поиска).

На рис. 1 приведена базовая архитектура крупномасштабных облачных центров обработки данных. На втором уровне приведенной архитектуры располагается слой виртуальных машин, которые могут быть автоматически запущены или остановлены в случае необходимости согласно общей загруженности облачного оборудования. Использование виртуальных машин позволяет на одном узле запускать приложения, разработанные под различные операционные системы, что обеспечивает высокую гибкость и удобство для нужд конечного облачного пользователя.

Так как входящая нагрузка непостоянна и может значительно варьироваться, потребности в ресурсах каждой виртуальной машины будут так же меняться [5, 6]. Чтобы консолидировать рабочую нагрузку на физическое оборудование и задействовать малоиспользуемые ресурсы, виртуальные машины могут мигрировать между узлами. Например, на рис. 1 присутствуют два узла, находящихся в режиме ожидания, в то время как на активных узлах параллельно запущено несколько виртуальных машин.

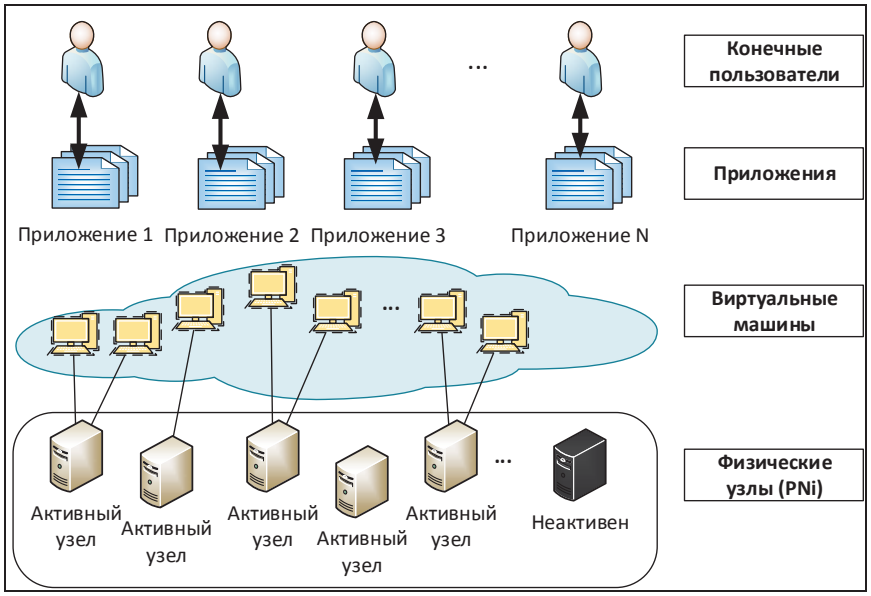

Рисунок 1. Базовая архитектура облачных центров обработки данных.

Зачастую для управления ресурсами внутри крупномасштабных центров обработки данных разрабатываются и внедряются централизованные решения, однако, в этом случае, возникновение сбоя на управляющем узле приводит к неработоспособности всей системы в целом. В [4] приводится описание децентрализованного механизма, позволяющего избежать указанной проблемы.

На рис. 2 представлены три типичных ситуации превышения верхнего порогового значения загруженности процессора. Очевидно, что для ситуации, представленной на рис. 2a, не составит труда выбрать виртуальную машину, перенос которой позволит снизить общую загруженность узла до установленных пороговых значений. Рис. 2b демонстрирует более сложный случай: каждая из виртуальных машин потребляет незначительную часть процессорного времени. В этой ситуации рекомендуется выполнять перенос нескольких виртуальных машин до тех пор, пока общая загруженность узла не снизится ниже порогового значения. В последнем случае (рис. 2c) невозможно выполнить перенос, который позволил бы не выйти за границы нижнего порога загруженности процессора.

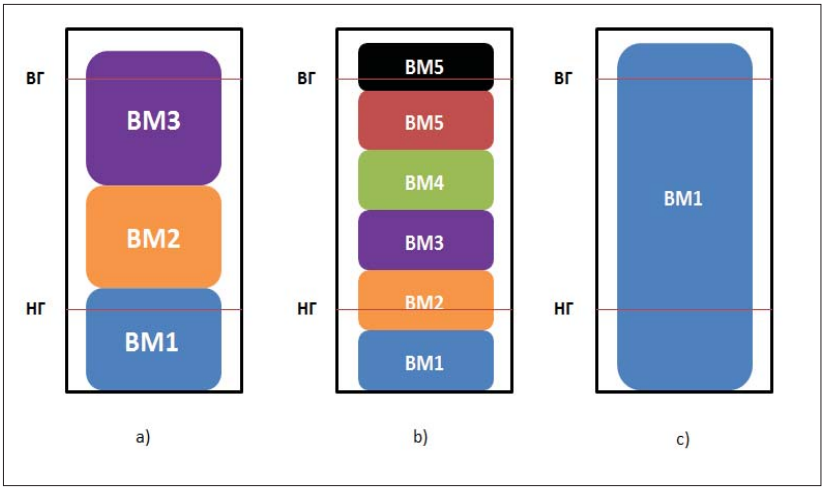

Рисунок 2. Типичные ситуации превышения верхнего порогового значения загруженности процессора (ВГ — верхняя граница, НГ — нижняя граница).

После того как принято решение о миграции виртуальной машины стартует поиск узла назначения [4]. Для этого выполняется обход вектора загруженности текущего узла с целью обнаружения узла с наименьшим потреблением ЦПУ при условии попадания в заданные интервалы. Если такой узел обнаружить не удается, выполняется поиск такого узла, индекс загруженности которого при переносе на него выбранной ВМ не превышает нижней границы загруженности. Если же и в этом случае поиск не дает результатов, один из узлов, находящихся в «спящем» режиме, переводится в активное состояние и выполняется миграция.

Анализ результатов тестирования приведенного подхода позволил выполнить оценку верхней и нижней границы индекса загруженности. Наименьшее количество отказов в обслуживании было выявлено при установлении верхней границы индекса загруженности узла равной 90% и нижней границы — 10%.

Разработанный подход обладает эффектом сбалансированности нагрузки [5]. Для подтверждения данной гипотезы были использованы три сценария распределения виртуальных машин по узлам экспериментального центра обработки данных [6] (количество физических узлов равно 50):

Статическое распределение (*Static*): на этапе инициализации виртуальные машины распределялись по узлам до тех пор, пока индекс загруженности каждого узла не достигал 100%.

Стратегия «Один к одному» (*RR*): на этапе инициализации виртуальные машины размещается на физических узлах по одной на каждый узел, миграция ВМ не выполняется.

Стратегия децентрализованного управления миграцией виртуальных машин (*DVM*): собственно этой стратегии посвящен текущий раздел.

Недостатки рассматриваемого подхода:

- алгоритмы не гарантируют обязательности выбора ВМ для миграции даже при условии необходимости в этом и наличии свободных физических узлов;
- не рассматривается оптимальность выбора ВМ для миграции;
- поиск целевого узла осуществляется не на всем наборе узлов (согласно описанию подхода);
- не говорится о том, как часто выполняется проверка необходимости миграции;
- алгоритмы не учитывают продолжительности нахождения узла в состоянии повышенной загруженности: очевидно, что при непродолжительной загруженности длительность миграции может снизить ее эффективность.

В [8] предложена схема исполнения и оптимизации динамических распределенных запросов в облачных пиринговых сетях (рис. 3), также разработан фреймворк (DObjects) для работы с p2p-сетями. Целью предложенного подхода является уменьшение времени отклика на запрос и увеличение производительности системы.

Ключевым элементом обработки запросов в предлагаемом подходе является наличие ядра, способного динамически адаптироваться к условиям сети и источникам. При этом подход обладает рядом новаторских идей:

Вместо генерации множества планов-кандидатов, их сравнения и выбора наилучшего, создается один план, который содержит подробное описание взаимосвязей между шагами и операциями, которые нужно выполнить для завершения исполнения запроса.

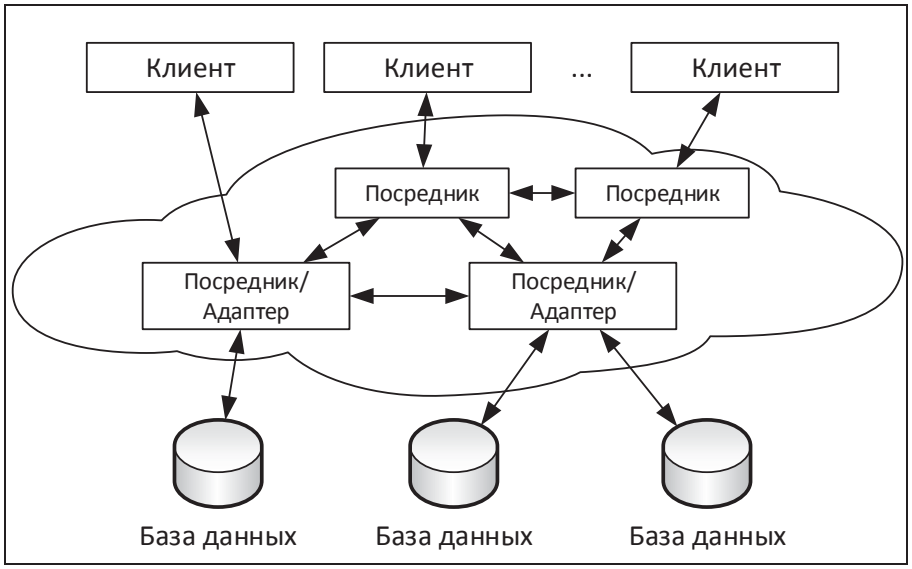

Рисунок 3. Архитектура облачной пиринговой (p2p) сети

Когда план исполняется, вывод результатов и физические расчеты по плану выполняются динамически и итеративно.

Такой подход гарантирует лучшую реакцию на изменение нагрузки и позволяет снизить задержки в системе. Следует отметить, что оптимизация исполнения запросов на локальных БД в текущем подходе ложиться на адаптеры и источники данных.

Исполнение и оптимизация запроса состоит из нескольких основных этапов. В момент получение узлом запроса от пользователя генерируется высокоуровневый план исполнения. На следующем шаге узел, исполняющий запрос, выбирает активные элементы плана сверху вниз в порядке следования. Однако исполнение активного элемента может быть делегировано любому узлу системы в целях достижения масштабирования нагрузки. Для выбора целевого узла исполнения в сети разворачивается модуль, способный адаптироваться к специфике сети и загруженности ресурсов. Если активный элемент передается на исполнение удаленному узлу, то управление его дочерними элементами так же возлагается на этот узел. Удаленный узел в свою очередь может принять решение о перемещении дочерних узлов элемента плана на исполнение другим узлам, либо выполнить локально.

Важным вопросом при исследовании алгоритмов оптимизации выполнения запросов является способность к масштабированию. Для того чтобы ответить на вопрос о том, как количество доступных узлов влияет на производительность системы, было измерено среднее время отклика для различного количества доступных системных узлов с 256 клиентами, одновременно выполняющими запросы к систе-

ме. Эксперимент продемонстрировал, что предложенный подход позволяет эффективно снизить среднее время отклика при наличии большего количества доступных узлов (за счет большего объема доступных вычислительных мощностей). Для небольших запросов увеличение количества узлов не приводит к снижению времени отклика, т.к. такие запросы не требуют значительных вычислительных ресурсов. Скорость сети, как и ожидалось, имеет большее влияние при работе с тяжелыми запросами. Причина в том, что количество данных, которые необходимо передать для тяжелых запросов больше, чем для средних и небольших запросов, и поэтому время, необходимое для передачи этих данных в медленной сети будет иметь гораздо большее воздействие на общую эффективность.

Достоинством данного подхода является то, что он может быть внедрен в кратчайшие сроки, поскольку алгоритмы были реализованы в виде фреймворка (набора библиотек), однако то, что последние изменения в исходный код вносились несколько лет назад (согласно сайту проекта в 2008 г.), говорит о не востребованности проекта. Причиной этому очевидно может служить специфика систем, для которых велась разработка, а именно облачные центры обработки данных, основанные на пиринговых сетях, которые на данный момент мало применяются на практике.

В настоящее время разрабатываются методы, использующие другие подходы к вычислительным процессам, в частности, использующие обратные связи [9].

Каждый из представленных методов обладает как достоинствами, так и недостатками. Общим для всех недостатком является синтетичность результатов, то есть то, что статистика по внедрению получена на искусственных системах (созданных только для тестирования подхода). Однако относительно большое количество исследований (для довольно молодой области) говорит о том, что проблема оптимизации актуальна и такие разработки в ближайшем времени будут востребованы.

Литература

[1] *Плужник Е. В., Никульчев Е. В.* Функционирование образовательных систем в облачной инфраструктуре // Известия высших учебных заведений. Проблемы полиграфии и издательского дела. 2013. № 3. С. 096-105.

[2] *Никульчев Е. В., Паяин С. В., Плужник Е. В.* Динамическое управление трафиком программно-конфигурируемых сетей в облачной инфраструктуре // Вестник РГРТУ. 2013. №3 (45). С. 54–57.

[3] *Плужник Е. В., Никульчев Е. В.* Слабоструктурированные базы данных в гибридной облачной инфраструктуре // Современные проблемы науки и образования. 2013. № 4. С. 95.

[4] *Xiaoying W., Xiaojing L., Lihua F., Xuhan J.* A Decentralized Virtual Machine Migration Approach of Data Centers for Cloud Computing // Mathematical Problems in Engineering. 2013. [Электронный ресурс]. http://www.hindawi.com/journals/mpe/2013/878542/

[5] *Rosenblum M., Garfinkel T.* Virtual machine monitors: current technology and future trends // Computer. 2005. Vol. 38. No. 5. P. 39–47.

[6] *Abdul-Rahman O., Munetomo M., Akama K.* Live migration-based resource managers for virtualized environments: a survey // Proceedings of the 1st International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING'10). 2010. P. 32–40.

[7] *Wang X., Du Z., Chen Y., Li S.* Virtualization-based autonomic resource management formulti-tierWeb applications in shared data center // Journal of Systems and Software. 2008. Vol. 81. No. 9. P. 1591–1608.

[8] Jurczyk P., Xiong L. Dynamic Query Processing for P2P Data Services in the Cloud // Proceedings of the 20th International Conference on Database and Expert Systems Applications, 2009. http://www.mathcs.emory.edu/~lxiong/research/pub/dobjects09dexa.pdf.

[9] *Pluzhnik E. V., Nikulchev E. V.* Use of Dynamical Systems Modeling to Hybrid Cloud Database // Int'l J. Communications, Network and System Sciences. 2013. Vol. 6. №.12.

Автор:
Леонов Д. В., магистрант Московского технологического института «ВТУ»