УДК 004.056

А.Н. Волокита, Ву Дык Тхинь, А.В. Щербина

АЛГОРИТМ ГИБКОГО МАСШТАБИРОВАНИЯ ДЛЯ ОБЛАЧНЫХ СИСТЕМ С ОГРАНИЧЕНИЕМ НА ВРЕМЯ ОБСЛУЖИВАНИЯ

Анннотация: Для cloud систем, в которых решаются задачи реального времени, накладываются дополнительные требования на алгоритмы планирования и масштабирования, связанные с ограничениями на время выполнения. Предложено модификацию алгоритма масштабирования для удовлетворения требованиям реального времени. Создана модель системы реального времени, проведенные эксперименты показали гибкость предложенного алгоритма масштабирования.

Ключевые слова: cloud computing, планирование, масштабирование.

Введение

Использование облачных систем для выполнения задач с учетом требований реального времени, позволяет использовать все преимущества cloud computing: масштабируемость, прозрачность, распределенность, однако накладывает дополнительные условия на функционирование таких систем.

Основные требования для облачных систем с гарантированным временем обслуживания:

1. Высокоскоростные источники данных – данные должны быть переданы в облако для вычислений, а затем назад к пользователю. Опрос готовности данных требует слишком много времени и пропускной способности.

2. Низкие задержки при передаче данных – данные должны быстро передаваться в базу данных реального времени. Реляционные базы данных, обычно используемые для бизнес-систем, являются слишком медленными.

3. Поддержка различных типов пользователей – тонкие клиенты, базы данных, электронные таблицы должны иметь доступ к источнику данных.

4. Независимая система резервирования - возможность обеспечить полное резервирование каналов передачи данных в случае любого перерыва в обслуживании.

5. Локальная синхронизация – система должна содержать копию данных с источника, и в реальном времени отправлять ее локальным клиентам и серверам по внутренней сети. Если канал связи с облаком прерван, то должны появиться отдельные области управления, которые смогут продолжить работу, пока связь с облаком не будет восстановлена [1].

Распределение задач по ресурсам в Cloud системе

Для оптимальной загрузки ресурсов Cloud системы и для предоставления задачам требуемого уровня сервиса необходимо планировать распределения задач по ресурсам, также необходимо учитывать ограничения реального времени. Рассмотрим существующие алгоритмы планирования.

Обзор алгоритмов планирования в облачных системах

Алгоритм планирования **Backfill** [2] разработан для максимально эффективного использования ресурсов и достижения высокой эффективности системы, предотвращения потенциально чрезмерных задержек, которые появляются при большой потребности в ресурсах.

Алгоритм обратного заполнения backfill работает по следующему принципу: размещая наиболее приоритетное задание, он определяет момент времени, когда освободится достаточное количество ресурсов, занятых уже выполняющимися заданиями, и производит резервирование этих ресурсов. Задание с меньшим приоритетом может быть запущено вне очереди, но только в том случае, если оно не будет мешать запуску всех (в консервативном варианте Backfill) более приоритетных заданий. Backfill может работать в параллельном режиме.

Отличительной чертой алгоритма **Min-Min** [3] является скорость выделения ресурсов для задач. Алгоритм как можно быстрее выделяет ресурсы на каждую из задач, которые могут быть выполнены в кратчайшие сроки. Алгоритм будет выполняться, пока весь набор задач не будет пуст. Задачи заносятся в короткие и длинные очереди, которые параллельно обрабатываются. Главным недостатком Min-Min является короткий этап планирования задач на начальных этапах, пока не начнут выполняться длинные задачи.

Алгоритм A* [4] – алгоритм поиска по первому наилучшему совпадению на графе, который находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной). Порядок обхода вершин определяется эвристической функцией “расстояние + стоимость”, обозначаемой как $f(x)$. Эта функция — сумма двух других: функции $g(x)$ стоимости достижения рассматриваемой вершины (x) из начальной (может быть как эвристической, так и нет) и эвристической оценки $h(x)$ расстояния от рассматриваемой вершины к конечной. Поиск продолжается до тех пор, пока процесс не выберет узел с полным назначением для расширения.

Для реализации модели Cloud системы реального времени используется последний алгоритм, поскольку он может быть моди-

фицирован (как будет показано дальше) для улучшения масштабируемости.

Масштабируемость в облачных системах

При работе с облачными real-time системами необходимо учитывать, что данная система должна быть доступна без простоев в любой момент времени и должна обеспечивать заданное время реагирования, независимо от количества одновременно работающих пользователей.

Основная проблема планирования заключается в необходимости прогнозирования количества пользователей, которые будут обращаться к сервису. Решение состоит в том, чтобы динамически масштабировать приложение и позволить количеству серверов и прочих компонентов расти (или уменьшаться) по требованию [5]. Масштабируемая облачная архитектура, которая эффективно реализует данный сценарий, показана на рис. 1.

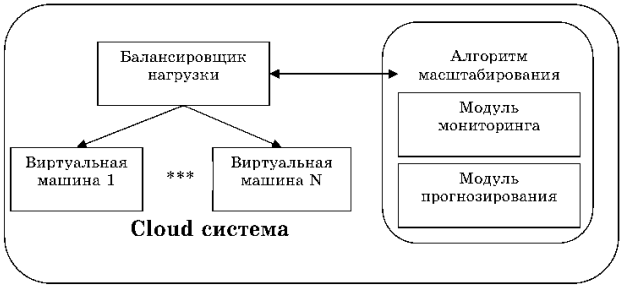

Рис. 1 – Масштабируемая облачная архитектура

Данная архитектура включает в себя интерфейсный балансировщик нагрузки, виртуальные машины с приложениями, подсистему резервирования и модуль мониторинга нагрузки с динамическим алгоритмом масштабирования.

Балансировщик нагрузки

Использование подсистемы балансировки нагрузки позволяет маршрутизировать входящие запросы на свободные ресурсы системы. Конфигурация данного модуля может обновляться в реальном времени, что позволяет системе автоматически динамически добавлять экземпляры приложения и запускать их на новых серверах.

Алгоритм масштабирования

Алгоритм динамического масштабирования, основан на показателе масштабирования в каждом экземпляре виртуальной машины. На первом шаге алгоритм определяет приложения с числом

активных сеансов выше или ниже заданных пороговых значений. Если количество активных сессий на всех экземплярах выше верхнего порога, то новый экземпляр приложения будет развернут, запущен и добавлен в интерфейс балансировщика нагрузки. Если имеются случаи с числом активных сеансов ниже заданного нижнего порога и, по крайней мере, один из экземпляров не имеет активных сеансов, то ресурс будет временно удален из балансировщика нагрузки и отключен от системы.

В каждом случае, коэффициенты нагрузки для всех активных экземпляров будут пересчитаны и применены к балансировщику для перераспределения равномерной нагрузки.

Модификация алгоритма для эффективной работы в условиях реального времени и уровней доступа

В условиях реального времени на первый план выходит время отклика и выполнения задачи. Необходимо учитывать, что время переноса задачи на другую машину, а также время запуска нового экземпляра, могут отразится на времени отклика. Поэтому дополним параметры задачи еще одним – временем отклика. И при возникновении ситуации, когда нужно переместить задачу на другую машину, проводится расчет времени перезапуска, если это время больше установленного времени отклика, то такой перезапуск не допускается и данная операция не выполняется.

Ниже представлен алгоритм балансировки нагрузки и динамического увеличения или уменьшения количества машин в облачной системе (рис. 2).

Поскольку более приоритетные задачи требуют более высокого уровня сервиса, то и требования к ресурсам различны. Потому нужно дифференцировать уровни пороговых значений для каждого приоритета. Для этого дополнительно добавляются два массива значений: верхнего и нижнего предела для каждого уровня приоритета, что позволяет гибко учитывать требования задач к уровню обслуживания.

Моделирование облачной системы реального времени

Одной из важнейших характеристик моделирования произвольной вычислительной системы, являются вычислительные ресурсы необходимые для проведения моделирования. К наиболее существенным ресурсам следует отнести объем требуемой оперативной памяти и время. Создание модели, учитывающей все свойства вычислительной системы возможно, однако ее решение требует чрезвычайно больших вычислительных ресурсов. Поэтому, используем упрощенную приближенную модель, и соответственно, уменьшаем ресурсы необходимые для моделирования. При этом точность определяемых параметров ухудшается незначительно.

```
For i:=0 to N {
  Определить загрузку i-ой машины -- MachineLoad;
  If(machineLoad>M_upper[уровень доступа машины]) {
    Запустить новый экземпляр машины
    с таким же уровнем доступа;
  }
  If(machineLoad<M_lower[уровень доступа]){
    RemovedTaskCount:=0;
    For j:=0 to Mi {
      Расчитать время перезапуска задачи T[j]
         -- TaskRebootTime;
      If(TaskRebootTime<TaskBorderTime[j]){
       Перезапустить задачу на другой машине;
       Удалить задачу из машины;
       RemovedTaskCount++;
      }
      If(RemovedTaskCount== M_i){
       Остановить машину i;
      }
   }
   Где:
   TaskBorderTime[]:массив времен отклика длявсех задач
   M_{upper}[]:Верхний порог нагрузки
   M_{lower}[]:Нижний порог нагрузки N-количество машин;
   Mi-количество задач на i-ой машине
```

Рис. 2 – Алгоритм балансировки нагрузки в Cloud системах реального времени

Моделирование входного потока заявок проводилось сложением 100 потоков Эрланга, каждый из которых моделирует поведения одного пользователя, на выходе получается поток Эрланга, поскольку в обращении клиентов наблюдается периодичность, моделирование входной очереди также проводится с периодом.

Согласно распределению Эрланга генерировались задачи, поступающие во входную очередь, и параметры этих задач. При моделировании системы были получены следующие результаты, которые представлены ниже в виде графиков.

На рис. 3 по вертикальной оси отложено количество ресурсов в системе, относительно начального количества, по горизонтальной оси – загрузка системы относительно начальной загрузки. При достижении системой уровня загрузки больше порога нагрузки начинают подключаться дополнительные ресурсы. На графике видно ступенчатое изменение количества ресурсов, это объясняется тем, что ресурсы подключаются дискретными частями.

На рис. 4 вертикальная ось – количество ресурсов в системе,

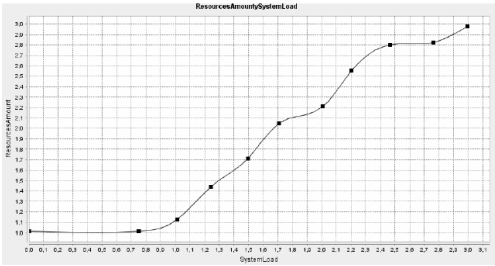

Рис. 3 – Зависимость кол-ва ресурсов от уровня загрузки системы

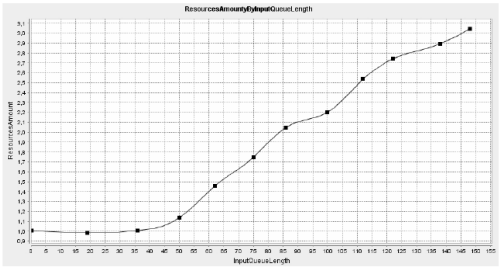

Рис. 4 – Зависимость кол-ва ресурсов от длины входной очереди

относительно начального количества, горизонтальная ось – количество задач во входной очереди. Как видно из рис. 4 длина входной очереди определяет загрузку системы. Поэтому характер графика совпадает с графиком зависимости количества ресурсов от уровня загрузки системы.

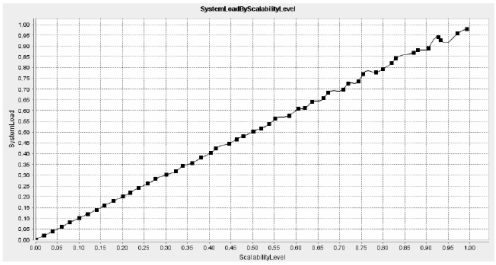

Рис. 5 – Зависимость загрузки от среднего порога нагрузки

Средний порог нагрузки – среднее арифметическое порогов нагрузки для всех уровней приоритета. Порог нагрузки определяет уровень загрузки для уровня приоритета после достижения которого, происходит подключение дополнительных ресурсов. Как ви-

дно из графика, загрузка системы растет при увеличении порога, поскольку чем меньше порог, тем раньше начинают выделяться ресурсы.

Выводы

Для удовлетворения требованиям реального времени, предложено улучшение алгоритма планирования A* для Cloud систем. Предложено ввести верхние и нижние пороговые значения для каждого уровня приоритета. В зависимости от текущей загрузки ресурсов системы и пороговых значений производится изменение количества виртульных машин. Алгоритм позволяет учитывать время установки новой машины и переноса на нее задачи, поскольку это может быть существенным в условиях реального времени.

Результаты моделирования показывают, что при уменьшении уровня пороговой нагрузки вероятность отказа системы снижается, но и увеличивается количество необходимых ресурсов.

Литература

1. *Zaina Afoulki.* A Security-Aware Scheduler for Virtual Machines on IaaS Clouds / Zaina Afoulki, Aline Bousquet, Jonathan Rouzaud-Cornabas. // Rapport de Recherche. – 2011. – 12p.
2. *Коваленко В.Н.* Использование алгоритма Backfill в грид / Коваленко В.Н., Семячкин Д.А. // Распределенные вычисления и Грид-технологии в науке и образовании : труды международной конференции, Дубна, 29 июня-2 июля 2004 г., Россия. – Дубна, 2004, – с. 139-144
3. *Muhammad Kafil.* Optimal Task Assignment in Heterogeneous Distributed Computing Systems / Muhammad Kafil and Ishfaq Ahmad // Complex Distributed Systems. – July–September 1998. – p. 42-51.
4. M. Dorigo. Optimization, Learning and Natural Algorithms: PhD thesis / Marco Dorigo. – Politecnico di Milano, Italy, 1992.
5. *Trieu C. Chieu.* Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment / Trieu C. Chieu, Ajay Mohindra, Alexei A. Karve and Alla Segal // IEEE International Conference on e-Business Engineering. – 2009. – p.281 – 286.

Отримано 08.02.2013