УДК 004.75

В. Гайдамако, dolpha@gmail.com
Институт машиноведения и автоматики НАН КР, Бишкек, Кыргызстан

АНАЛИЗ АЛГОРИТМОВ И МОДЕЛИРОВАНИЕ БАЛАНСИРОВКИ НАГРУЗКИ В ОБЛАЧНОЙ ИНФОРМАЦИОННО-ИЗМЕРИТЕЛЬНОЙ СИСТЕМЕ

Балансировка нагрузки играет важнейшую роль в обеспечении производительности облачных вычислений. Назначение балансировки – равномерное использование вычислительных ресурсов – центров обработки данных, серверов, виртуальных машин для обеспечения высокого качества обслуживания запросов пользователей. Некоторые из алгоритмов балансировки нагрузки, условия их применения, достоинства и недостатки рассматриваются в статье. Также рассматриваются задачи, которые должны быть решены при разработке модели облачной информационно-измерительной системы с учетом балансировки нагрузки.

Ключевые слова: облачные вычисления, облачные ИИС, балансировка нагрузки, Round Robin, Throttled Algorithm, Equally Spread Current Execution, Min-Min, Max-Min, исчисление реального времени, моделирование.

Введение. В связи с широким распространением распределенных и облачных вычислений в последнее время вопросы балансировки нагрузки привлекают внимание и практиков, и многочисленных исследователей. Особенностью облачных вычислений является применение технологии виртуализации, позволяющей консолидировать распределенные ресурсы и предоставляющей пользователю видимость цельной системы, а также способность динамически изменять состав оборудования, обслуживающего запросы пользователя в зависимости от нагрузки [1]. Физической средой облака являются центры обработки данных (ЦОД), связанные друг с другом через глобальные или корпоративные сети. Балансировка необходима и при принятии решения, на какой ЦОД облака будет направлен запрос пользователя, и внутри ЦОД, при планировании размещения запроса на вычислительном ресурсе – виртуальной машине (ВМ). Пользователи генерируют запросы на услуги случайным образом, и, если запросы будут направляться на исполнение без предварительного планирования, может появиться дисбаланс в распределении нагрузки – какие-то ЦОД, серверы или виртуальные машины (ВМ) будут перегружены, в то время как другие простаивать. Если принять во внимание, что облачные ЦОД состоят из тысяч физических серверов, на каждом из которых могут быть запущены несколько экземпляров виртуальных ресурсов (ВМ или контейнеров), становится ясно, что задача балансировки нагрузки в облачной среде является одной из важнейших.

Балансировка нагрузки распределяет рабочую нагрузку между несколькими вычислительными ресурсами таким образом, чтобы, с одной стороны, минимизировать время отклика на запрос пользователя, а с другой— обеспечить эффективное использование

вычислительных и коммуникационных ресурсов. При планировании виртуальных ресурсов возникает возможность переноса нагрузки с перегруженных ВМ на недогруженные, а также миграции, добавления и удаления ВМ с физических серверов [2, 3, 4]. Алгоритмы балансировки нагрузки в облаке позволяют эффективно предоставлять ВМ для обслуживания запроса пользователя.

Типы алгоритмов балансировки. Алгоритмы балансировки нагрузки в облачной среде подразделяются на статические и динамические. Исходными данными для алгоритмов статической балансировки нагрузки является информация о ресурсах системы и выполняемых задачах, они просты в разработке и требуют небольших вычислительных ресурсов и времени. Недостатком статических алгоритмов является то, что применяться они могут только в системах, где все серверы и ВМ идентичны по характеристикам, они не учитывают текущее состояние системы, не гибки, плохо приспосабливаются к изменяющейся нагрузке.

Алгоритмы динамической балансировки нагрузки более гибки, надежны и способны обрабатывать большее количество запросов пользователей, чем статические алгоритмы, могут изменять нагрузку в процессе обработки задач. Эти алгоритмы используют информацию о текущем состоянии системы и лучше всего подходят для облачной среды и непредсказуемых рабочих нагрузок. Недостатком является увеличение накладных расходов на балансировку. Динамические алгоритмы делятся на централизованные и распределенные. В централизованных алгоритмах решения принимаются на одном или нескольких узлах балансировки, в распределенных системах в принятии решения участвуют все узлы, которые взаимодействуют и обмениваются информацией для обеспечения лучшей балансировки. Распределенные решения более устойчивы к отказам и хорошо работают в гетерогенной среде, но требуют высокой частоты репликации, что приводит к большой нагрузке на коммуникации и может привести к снижению производительности системы в целом. Распределенные алгоритмы в свою очередь могут быть кооперативными и некооперативными [2]. При кооперативном подходе существует общая цель, критерий работы системы в целом, к которой «стремятся» все узлы. При некооперативном у каждого узла своя цель, например, улучшить время отклика для «своих» задач. Данные о состоянии ВМ собираются периодически в процессе мониторинга, при изменении состояния или по необходимости.

В отдельную группу выделяют [5, 6] алгоритмы, основанные на изучении природных процессов, прежде всего поведения коллективных насекомых – бионические алгоритмы.

Критерии оценки. Результатом работы балансировка нагрузки должно стать эффективное использование ресурсов и, следовательно, распределение задач на ВМ таким образом, чтобы не было перегруженных, недогруженных или простаивающих ВМ. После этого уже на ВМ применяются алгоритмы планирования задач. В совокупности все эти действия должны обеспечить выполнение требований, указанных в Соглашении о качестве обслуживания (Service Level Agreement – SLA).

В качестве метрик для оценки алгоритмов балансировки нагрузки могут рассматриваться [5, 7]:

- Использование процессорного времени (для центрального процессора – ЦП): общий процент времени, в течение которого процессор или процессоры использовались или не использовались. Также важно использование других ресурсов – памяти, пространства хранения, пропускной способности сети.
- Пропускная способность: количество выполненных на ВМ задач (или обслуженных запросов) за единицу времени.
- Пропускная способность линий связи. Количество данных, пересылаемых из одного узла другому.
- Масштабируемость: алгоритм должен эффективно работать независимо от увеличения/уменьшения количества задач и нагрузки, в том числе при пиковых нагрузках.
- Время отклика: время, необходимое для выдачи первого результата запроса. Сюда включается время ожидания, время передачи и время обслуживания. При оценке эффективности работы алгоритма балансировки может рассматриваться время отклика для балансировки – время от поступления запроса до начала выполнения первой команды обслуживания запроса на ЦП [5].
- Время ожидания: общее время, затраченное запросом на ожидание в очереди готовности после первого выполнения на ЦП.
- Время выполнения: общее время, затраченное на полное обслуживание запроса, включая время ответа, время ожидания и время обслуживания.
- Общее время выполнения набора задач (makespan), необходимое для выполнения всех задач набора, включая время распределения ресурсов пользователям в системе.
- Справедливость: принцип, согласно которому каждый запрос должен получать равную долю процессорного времени.
- Отказоустойчивость (надежность): балансировка нагрузки должна работать вне зависимости от отказа некоторых элементов системы. Если одна ВМ становится недоступной по причине отказа или перегруженности, другая доступная ВМ должна иметь возможность выполнять задачи.
- Стоимость ресурсов: общая стоимость ресурсов, приобретенных или используемых для обслуживания запросов различными потребителями облака.
- Сопутствующие накладные расходы: объем затрат, образованных при выполнении алгоритма балансировки нагрузки.
- Время миграции: это общее время, необходимое для миграции задачи с одной ВМ на другую. Процесс миграции не должен влиять на доступность системы. Чем ниже время миграции, тем лучше будет общая производительность системы и производительность алгоритма балансировки нагрузки.
- Нарушение Соглашения о качестве услуг (Service Level Agreement – SLA: количество нарушений SLA с точки зрения ограничения сроков, приоритетов и т.

д. Нарушения SLA происходят из-за ситуаций, когда ресурсы недоступны из-за перегруженности ВМ.

- Энергоэффективность и снижение углеродного следа –алгоритм балансировки не должен приводить к значительному повышению потребления электроэнергии, и, следовательно, увеличению углеродного следа.

Эти критерии могут использоваться при оценке алгоритма балансировки нагрузки как на реальных облачных системах, так и при моделировании облачных и распределенных систем.

Статические алгоритмы балансировки нагрузки

Циклический алгоритм – Round Robin (RR). Балансировщик поддерживает список доступных виртуальных машин в виде циклической очереди и выделяет для обслуживания поступившего запроса первую ВМ в очереди, затем следующему и т.д. При достижении конца списка указатель очереди перемещается на первый элемент списка. По этому принципу, например, работают серверы DNS. Так как первая в очереди ВМ была назначена для обслуживания какое-то время назад, велика вероятность того, что она к этому моменту выполнила все задачи и является наименее загруженной. Но, конечно, это верно не всегда, возможно, задача или задачи, назначенные ранее, требуют больших затрат вычислительных ресурсов и времени, и эта ВМ в данное время как раз перегружена. Алгоритм лучше всего работает для схожих по характеристикам ВМ и однотипных задач. Главное достоинство алгоритма – простота и легкость реализации. Основной недостаток – требует предварительной информации о задачах пользователя и ресурсах, не принимает во внимание текущее состояние системы, серверы и ВМ не отличаются по характеристикам [2, 5]. Существует также циклический алгоритм RR для планирования заданий на одной машине, его также называют алгоритмом с разделением времени, это один из самых распространенных алгоритмов планирования. Каждой задаче для выполнения выделяется один квант времени, по истечении которого задача вытесняется, происходит смена контекста и на исполнение идет другая задача. Этот алгоритм может nприменяться в ВМ, но не является алгоритмом балансировки.

Циклический алгоритм с весами -– Weighted Round Robin (WRR). Циклический алгоритм с весами (WRR) отличается от традиционного RR тем, что каждому узлу присваивается вес в зависимости от его характеристик, и для обслуживания запроса выбирается ВМ с наибольшим весом из следующей в очереди. Таким образом повышается загруженность более производительных ВМ, а ВМ с низкой производительностью становятся не перегруженными. Разрабатываются гибридные алгоритмы с использованием WRR, например, в [8] предлагается эффективный алгоритм балансировки нагрузки, основанный на комбинации WRR и Max-Min (WmaxMin), сокращающий время ожидания и время отклика. Он использует Max-Min для выбора задачи с максимальным временем обработки и назначает ее ВМ, имеющей самые высокие возможности обработки. В [9] предложен метод, учитывающий приоритеты запросов. Исходный балансировщик нагрузки разделяет запросы на обычные и срочные. Каждая ВМ отправляет свой вес балансировщику,

чтобы определить количество запросов, которые могут быть распределены на нее. Этот подход имеет ограничение на большое время выполнения по сравнению с WRR и предполагает, что все запросы имеют одинаковый уровень приоритета.

Алгоритмы Min-Min и Max-Min. Алгоритм Min-Min (MM) так же, как и Max-Min алгоритм, использует для планирования время выполнения задачи и скорость обработки процессора [10]. Min-Min задачу с наименьшим временем выполнения направляет на самый «быстрый» ресурс. В этом случае более «длинные» задачи могут долго находиться в очереди ожидания («голодать») из-за того, что приоритет отдается более мелким задачам. Из-за того, что всегда выбирается ресурс с минимальным временем исполнения, невзирая на то, занят он или нет, может возникнуть дисбаланс нагрузки. Max-Min алгоритм, наоборот, отправляет на самый быстрый ресурс наиболее «крупную» задачу с максимальным временем выполнения. Если количество «мелких» задач превышает количество «крупных», Max-Min алгоритм выглядит предпочтительнее, однако при большом количестве «крупных» задач общая пропускная способность системы может пострадать.

Алгоритм учета ресурсов – Resource Aware Scheduling Algorithm (RASA). Этот алгоритм сочетает функции алгоритмов Max-Min и Min-Min. Алгоритмы Max-Min и Min-Min один за другим применяются в соответствии с количеством доступных ресурсов, если количество ресурсов четное, тогда применяется Max-Min, иначе применяется Min-Min алгоритм. В [10] предложен Resource Aware Min-Min(RAMM) алгоритм, который ведет учет ресурсов и их состояний «доступно/занято», и, если ресурс с минимальным временем исполнения занят, выбирается самый быстрый ресурс из оставшихся.

Динамические алгоритмы балансировки нагрузки

Алгоритм минимума соединений – Least Connections Algorithm. Запрос направляется на узел с наименьшим количеством активных соединений. Для этого нужно поддерживать список узлов с количеством активных соединений и периодически обновлять его. Может применяться при принятии решения, в какой ЦОД запрос будет направлен для обслуживания [2, 11].

Регулируемый алгоритм балансировки нагрузки – Throttled Algorithm. Запрос пользователя отправляется контроллеру центра обработки данных (Data Cnter Controller – DCC). Контроллер ЦОД обращается к балансировщику нагрузки для поиска ВМ, которая может справиться с данной рабочей нагрузкой. Балансировщик поддерживает список виртуальных машин и их статус «доступна/занята». Если подходящая ВМ найдена (по производительности ядер ЦП, объему памяти, доступности), балансировщик принимает запрос на обслуживание и направляет задачу выбранной ВМ. Если же подходящая ВМ не свободна, задача отправляется в очередь до тех пор, пока подходящая ВМ не станет доступной [12, 13]. Этот подход хорош для балансировки нагрузки, поскольку он рассматривает текущее состояние всех ВМ в ЦОД. Основным недостатком является то, что он работает правильно только в том случае, если все виртуальные машины в центре

обработки данных имеют одинаковую конфигурацию оборудования, не принимает во внимание время обработки. Существует множество модификаций этого алгоритма [5].

Алгоритм балансировки нагрузки с равным распределением – **Equally Spread Current Execution (ESCE).** Его также называют методом расширения спектра (Spread Spectrum technique), так как он «расширяет», распределяет нагрузку по различным узлам. Размер задачи определяет ее приоритет, балансировщик направляет задание на ВМ с небольшой нагрузкой случайным образом. Балансировщик поддерживает список ВМ в виде хэш-таблицы, с ключом-идентификатором ВМ и значением – количеством выполняющихся задач. Запросы пользователей поступают в очередь заданий. Балансировщик проверяет очередь заданий и список ВМ. Если какая-либо ВМ свободна, а в очереди есть задание, запрос направляется ей. Если же какая-либо ВМ оказывается перегруженной, балансировщик перемещает часть нагрузки на незанятую или недогруженную ВМ. При завершении запроса ВМ сообщает об этом балансировщику, так как таблица ВМ должна постоянно обновляться. Главный недостаток – высокие вычислительные и коммуникационные затраты для обновления таблицы ВМ [6].

Гибридные алгоритмы (TA+ESCE). Знание состояний виртуальных машин и выполняющихся на них приложений – две основные особенности алгоритмов TA и ESCE. Если объединить эти функции, алгоритм планирования становится более эффективным, а нагрузка распределяется более равномерно и справедливо. В [14] для уменьшения времени отклика в ЦОД предлагается гибридный балансировщик (TA & ESCE), который ведет хэш-таблицу ВМ и запросов и ищет доступную виртуальную машину. Но если ВМ заняты (при использовании TA запрос отправляется в очередь), используется ESCE – задача назначается ВМ с наименьшей загрузкой. Похожий подход описан в [15]. Балансировщик в случае занятости ВМ не помещает задание в очередь, а создает новую ВМ. Оба алгоритма уменьшают время отклика, но не принимают во внимание миграцию ВМ. Другие подходы используют пороговое значение нагрузки или состояние ВМ для перемещения части нагрузки на недогруженные машины, а в некоторых случаях создают новую ВМ для размещения задачи [5].

Бионические алгоритмы балансировки

Исследования социальных животных и социальных насекомых со сложным коллективным поведением привели к созданию ряда вычислительных моделей коллективного интеллекта роя. Этот тип алгоритмов балансировки относится к классу метаэвристичских алгоритмов [16], которые дают практически приемлемое решение задачи оптимизации в условиях недостаточной или ограниченной вычислительной мощности. Методы могут быть как относительно простыми, так и очень сложными и требующими высоких вычислительных мощностей. В этой области в последние годы проводится много исследований и экспериментов.

Алгоритм сбора меда – **Honey Bee algorithm.** Пчелы-разведчицы разлетаются в поисках нектара, если найдут – летят в улей и сообщают другим пчелам. Рабочие пчелы

летят в найденные места и уточняют местоположение источника и собирают нектар [17]. Если сопоставить улью — ЦОД (или несколько, если речь идет о выборе ЦОД), пчеле — задачу (с учетом приоритетов), цветку (источнику нектара) — ВМ, процессу поиска нектара — загрузку задачи на ВМ, можно использовать алгоритм поиска меда для поиска подходящей ВМ. Недостатком алгоритма является высокое время ожидания для задач с низким приоритетом. Создано множество реализаций и модификаций этого алгоритма, а также гибридных алгоритмов на его основе [5].

Алгоритм оптимизации колонии муравьев (Муравьиный алгоритм) – **Ant Colony Optimization Algorithm.** Этот алгоритм основан на поведении муравьев при пище. Муравьи беспорядочно исследуют окрестности в поисках пищи и, если находят, возвращаются в колонию, помечая путь феромоном (химическое вещество, которое воспринимают другие муравьи) . Другие муравьи, пересекая эту тропу, с большой вероятностью последуют по этому пути и, если найдут пищу, также будут помечать ее. Таким образом, наиболее удачные тропы будут помечены большим количеством феромона. Феромон испаряется со временем, поэтому более короткий путь будет больше помечен [18, 19]. Применение муравьиного алгоритма и гибридные алгоритмы обсуждаются в [5, 18-20].

Алгоритм оптимизации роя частиц – **Particle Swarm Algorithm.** Использует социальное поведение скоплений живых существ в природе, например, стремительный полет группы летающих существ или быстрые перемещения стаи рыб с резким изменением направления движения всей стаи – часто кажется, что ими кто-то управляет, настолько «разумно» поведение. Каждая особь является частицей. Частицы ищут цель с заданной скоростью (стая уток в поисках пищи), стараясь не сталкиваться и держась на равном расстоянии друг от друга. Поскольку частицы изменяют и обновляют ситуацию относительно себя и своих соседей, алгоритм может выполнять как локальный, так и глобальный поиск. В алгоритме рой составляют возможные решения – частицы, оптимум находится путем перемещения частиц в пространстве поиска, что описывается простыми математическим формулами [21]. Определяется лучшая найденная локальная позиция, а также лучшие найденные позиции в пространстве поиска, они обновляются как лучшие позиции, а затем их находят другие частицы. Ожидается, что это переместит рой к лучшим решениям. Перемещения подчиняются принципу наилучшего найденного в этом пространстве положения, которое постоянно изменяется при нахождении частицами более выгодных положений. Алгоритм роя частиц широко применяется в задачах машинного обучения. В [5] обсуждаются различные реализации и модификации этого алгоритма.

Генетический алгоритм – **Genetic Algorithm (GA).** Основан на процессе естественного отбора в природе. Генетические алгоритмы служат главным образом для поиска решений в многомерных пространствах поиска. [22]. Решение задачи представляется в виде набора генов – генотипа. Ген может быть битом или числом. Создается набор генотипов — начальная популяция. Состояние генотипа оценивается с точки зрения его «приспособленности» с помощью функции приспособленности (fitness-функции).

Выбираются наиболее приспособленные особи (этап Отбора или Селекции), и проводится Скрещивание (Crossover) и Мутация (изменение), и получается новое поколение решений. Процесс повторяется, пока не будет найдено решение [23]. Алгоритм усложняется при увеличении пространства поиска, что делает его затратным по времени.

Алгоритм с использованием модульного анализа производительности на основе исчисления реального времени

В [24] был предложен метод оценки производительности облачных приложений с использованием математического моделирования (ИРВ) – модульный анализ на основе исчисления реального времени (МА-ИРВ). Для проведения модульного анализа производительности с ИРВ (МА-ИРВ) необходимо построить абстрактную модель производительности, включающую модель нагрузки, модель обслуживания и модель обработки. Модель должна включать сведения о доступных вычислительных и коммуникационных ресурсах, о приложениях и об архитектуре самой системы. Реальная система представляется в виде абстрактных аналитических компонент (ИРВ-компоненты). Компоненты обработки – это ВМ. Для каждого ресурса на основе его характеристик и дисциплины обслуживания строятся кривые обслуживания. Ресурсная модель собирает информацию о доступных ресурсах обработки, включенных в обработку запросов и о распределении задач по этим ресурсам. Также рассматриваются характеристики потока событий, входящих в систему (запросов клиентов, запросов на обслуживание от сервера приложений к серверу баз данных), которые описываются их кривыми поступления. Используя аналитический фреймворк ИРВ, мы можем вычислить максимальную задержку на одном ресурсе и максимальную задержку при прохождении потока событий через несколько ресурсов обработки (для многоуровневых приложений). Обработка представляется как преобразование кривых поступления и обслуживания в выходные. ИРВ позволяет получить выходные кривые из входных с помощью уравнений. Таким образом могут быть получены значения жесткого реального для времени обслуживания потока запросов и размера очереди. В [24] предлагается использовать МА-ИРВ в мониторинге облачных систем (рис. 1).

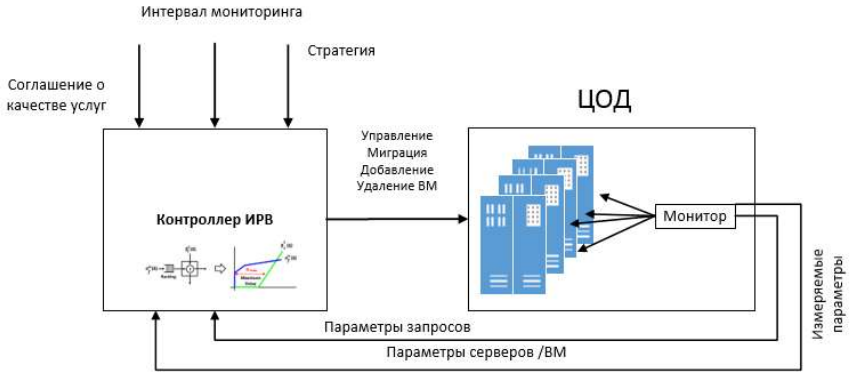

Рис.1. Мониторинг облака с на основе ИРВ

Все параметры, необходимые для построения ИРВ-модели, собираются в процессе мониторинга, контроллер ИРВ анализирует текущее состояние, и, если время обработки запросов и/или размер очереди выходит за заданные предельные величины, производится управляющее воздействие – добавление, удаление или миграция ВМ. Таким образом, система динамически адаптируется к изменениям.

Моделирование облачной информационно-измерительной системы с учетом балансировки нагрузки

Моделирование облачной системы с использованием балансировщика нагрузки позволяет провести модельные эксперименты с различными параметрами при проектировании для выбора оптимальной политики балансировки. Практически во всех рассмотренных работах эффективность работы алгоритма балансировки проверялась с помощью инструмента моделирования Cloudanalyst, который был разработан как средство визуализации на основе пакета моделирования облачных и распределенных систем CloudSim [25], построенного на основе GridSim. В инструменте реализованы алгоритмы (политики) балансировки нагрузки на уровне ЦОД - Round Robin, Equally Spread Current Execution Load и Trottled. Возможна разработка собственных модулей балансировки нагрузки для проведения экспериментов с другими алгоритмами. В работах [2, 7-10, 12-15,18,20, 21] рассматривается использование инструмента Cloudanalyst для моделирования работы ЦОД с различными политиками балансировки нагрузки.

Однако этот инструмент не позволяет моделировать работу облачных информационно-измерительных систем (ОИИС), которые имеют разные типы нагрузки, что должно быть учтено при моделировании:

- нагрузка от беспроводных сенсорных сетей (БСС) и отдельных датчиков, это предсказуемая нагрузка с известным временем и объемом трафика. Для балансировки приема данных могут применяться статические алгоритмы;
- нагрузка от виртуальных датчиков, предсказуемая, но меняющаяся;
- нагрузка от пользователей веб-портала.

Это означает, что и в системе, и в модели должны быть реализованы три вида балансировщиков нагрузки, реализующих разные политики балансировки. В [26] описана модель простого ЦОД для ОИИС, создаваемая с помощью пакета моделирования распределенных систем Simgrid. Модель содержит следующие компоненты:

- хост Sensor;
- хост Координатор (Gateway). Управляет подсоединенными датчиками (мотами), собирает сенсорную информацию, возможно, частично обрабатывает ее и передает в облако, на облачную станцию;
- хост Облачная Станция (Cloud Station) – собирает сенсорную информацию, а также информацию о состоянии датчиков с подсоединенных координаторов и перенаправляет по назначению – на виртуальные датчики или серверы хранения;
- хост Сервер Хранения;
- виртуальная машина.

Для моделирования балансировки нагрузки добавляются следующие компоненты:

- хост DCController (контроллер ЦОД, координирует всю работу ЦОД) ;
- хост LoadBalanser (балансировщик нагрузки) собирает и анализирует данные мониторинга и обращается к модулю расчета;
- программный модуль расчета для алгоритма балансировки (например, контроллер ИРВ на рис.1);
- агент – программный модуль для сбора информации о текущем состоянии ВМ (мониторинга), загружается на ВМ при создании;
- генератор нагрузки на веб-сервере, нагрузка от физических и виртуальных датчиков генерируется соответствующими хостами и ВМ;
- база данных пользователей – содержит информацию о расположении и активности пользователей (часы активности, объем трафика на запрос).

Так как предполагается, что модель будет использоваться для тестирования эффективности работы различных алгоритмов, модули расчета должны легко подключаться и иметь стандартный интерфейс, в идеале также подходящий для включения в модель Cloudanalyst, что позволит быстрее проверить работу алгоритма и сравнить с результатами уже проведенных тестов.

Для создания модели ОИИС должны быть выполнены следующие задачи:

1. Создание программных модулей для компонент модели, включая коммуникации.
2. Создание программных модулей для осуществления облачного мониторинга.
3. Создание программных модулей для осуществления и учета миграции задач и ВМ.
4. Выбор алгоритмов балансировки для разных типов балансировщиков и создание программных модулей для их реализации.
5. Создание программных модулей для генерации потоков запросов в соответствии с базами пользователей, потоков запросов с заданными характеристиками.
6. Создание программных модулей сбора и обработки данных эксперимента.
7. Создание программных модулей для визуализации результатов эксперимента.
8. Создание графического интерфейса пользователя для задания конфигурации модели и параметров эксперимента.

Часть этих задач может быть выполнена с помощью пакета моделирования Simgrid, часть должна быть разработана как самостоятельные модули, которые могут быть использованы для других целей и в других моделях.

Заключение

Задача балансировки нагрузки в облачных системах является одной из важнейших в этой области, проводится огромное количество исследований, создано множество алгоритмов балансировки. В статье очень коротко рассматриваются лишь некоторые из алгоритмов балансировки нагрузки в облачных средах, особый интерес представляет алгоритм с использованием исчисления реального времени, информации о его тестировании еще нет в литературе. Представленный обзор проводился прежде всего с целью выбора алгоритмов, которые планируется реализовать и тестировать в создаваемой модели

облачной ИИС для разных типов балансировщиков. Рассмотрены вопросы построения модели облачной ИИС с учетом балансировки нагрузки и миграции ВМ и поставлены задачи развития создаваемой модели.

Литература

1. Q. Zhang, L. Cheng, R.Boutaba, "Cloud Computing: state of the art and research challenges", Journal of Internet Services and Applications. – Vol. 1, No. 1. –April 2010, p.7–18.
2. S.K.Mishra, B. Sahoo, P.P. Parida. Load balancing in cloud computing: A big picture., J. King Saud Univ. – Comput Inf. Sci.,32 (2) (2020), p.149–158, URL:10.1016/j.jksuci.2018.01.003(дата обращения: 10.06.2021)
3. S.K. Mishra, D. Puthal, B. Sahoo, S.K. Jena, M.S. Obaidat. An adaptive task allocation technique for green cloud computing., J. Supercomput. (2017), p. 1–16.
4. Дворников В.С., Долгов В.В., Венцов Н.Н. Обзор методов балансировки нагрузки в гетерогенных распределенных файловых системах // Фундаментальные исследования, 2017. – № 9–2. – С. 295–302; URL: http://www.fundamental-research.ru/ru/article/view?id=41743 (дата обращения: 10.06.2021).
5. Dalia Abdulkareem Shafiq, N.Z. Jhanjhi, Azween Abdullah. Load balancing techniques in cloud computing environment: A review. // Journal of King Saud University–Computer and Information Sciences, 2021; URL: 10.1016/j.jksuci.2021.02.007 (дата обращения: 10.06.2021).
6. Thakur, M.S. Goraya. A taxonomic survey on load balancing in cloud**.** Netw. Comput. Appl., 98 (September) (2017), URL: 10.1016/j.jnca.2017.08.020(дата обращения: 10.06.2021).
7. S. Mishra, R. Tondon. A shared approach of dynamic load balancing in cloud computing**.,**Int. J. Sci. Res. Sci. Eng. Technol., 2(2) (2016), p.632– 638.
8. B. Khatavkar, P. Boopathy. Efficient WMaxMin static algorithm for load balancing in cloud computation. Int. Conf. Innov. Power Adv. Comput. Technol., i-PACT2017 (2017), p. 1–6, 10.1109/IPACT.2017.8245166.
9. S. Manaseer, M. Alzghoul, M. Mohmad. An advanced algorithm for load balancing in cloud computing using MEMA technique. Int. J. Innov. Technol. Explor. Eng., 8 (3) (2019), p. 36–41.
10. Arshad Ali, S., Member, S., Alam, M., 2019. Resource Aware Min-Min (RAMM) Algorithm for Resource Allocation in Cloud Computing Environment, 3, p. 1863–1870 doi: 10.35940/ijrte.C5197.098319.
11. URL: https://www.ibm.com/docs/en/datapower-gateways/10.0.1?topic=groups-algorithms-making-load-balancing-decisions (дата обращения: 06.06.2021).
12. Durgesh Patel, Mr. Anand S Rajawat, “Efficient Throttled Load Balancing Algorithm in Cloud Environment’, International Journal of Modern Trends in Engineering and Research. – p. 463 – 480, 2015.

13. Десятирикова Е.Н.,Хадж Али Муса,Ходар Алмосана, Алькади Усама, Раджаб Хаян. Балансировка нагрузки в облачных вычислениях. – Вестник ВГТУ,2017. – №3. URL: https://cyberleninka.ru/article/n/razrabotka-modeli-dlya-uluchsheniya-planirovaniya-zadach-v-oblachnyh-vychisleniyah-na-osnove-optimizatsii-roya-chastits (дата обращения: 06.07.2021).
14. R. Sachdeva, S.Kakkar. A Novel Approach in Cloud Computing for Load Balancing Using Composite Algorithms. Int. J. Adv. Res. Comput. Sci. Softw. Eng., 7(2) (2017). – p.51– 56, 10.23956/ijarcsse/v7i2/0119.
15. S. Subalakshmi, N. Malarvizhi. Enhanced hybrid approach for load balancing algorithms in cloud computing. Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol., 2 (2) (2017), p.136–142.
16. https://wikiboard.ru/wiki/Metaheuristic
17. B.Yuce, M.S.Packianather, E.Mastrocinque, D.T.Pham, A.Lambiase/ Honey bees inspired optimization method: the Bees algorithm // Insects, 4(4)(2013), p.646 – 662, URL:10.3390/insects4040646.
18. P.Verma, S.Shrivastava, R.K.Pateriya. Enhancing Load Balancing in Cloud Computing by Ant Colony Optimization Method // Int. J. Comput. Eng. Res. Trends, 4(6) (2017), p.277–284, 10.23883/ijrter.2018.4101.ss6y8(дата обращения: 11.06.2021).
19. Конников П. В., Кудинов В. А. Оптимизация методом муравьиной колонии как метаэвристика // Ученые записки. – Электронный научный журнал Курского государственного университета, 2008. – №4. URL:https://cyberleninka.ru/article/n/optimizatsiya-metodom-muravinoy-kolonii-kak-metaevristika (дата обращения: 11.06.2021).
20. Kumar, R., Prashar, T., 2015. Performance Analysis of Load Balancing Algorithms in Cloud Computing, Int. J. Comput. Appl. (0975 – 8887), 120(7), p. 19–27.
21. Parmesivan, Y.A.P., Hasan, S., Muhammed, A., 2018. Performance Evaluation of Load Balancing Algorithm for Virtual Machine in Data Centre in Cloud Computing, Int. J. Eng. Technol., 7(4.31), p. 386–390.
22. URL: https://ru.wikipedia.org/wiki/Генетический_алгоритм (дата обращения: 11.06.2021)
23. Gandhi, R., Genetic Algorithms – Data Driven Investor – Medium," 12-May-2018. URL: https://medium.com/datadriveninvestor/genetic-algorithms-9f920939f7cc. (дата обращения: 11.06.2021).
24. Гарай Г.Р., Черных А., Дроздов А.Ю. Сравнительный анализ методов оценки производительности многоуровневых облачных приложений. Труды ИСП РАН. – Том 27.– Вып. 6, 2015 г. – Стр. 199–224. DOI: 10.15514/ISPRAS-2015-27(6)-14
25. URL: http://www.cloudbus.org/cloudsim/. (дата обращения: 11.06.2021).
26. Гайдамако В.В. Моделирование облачной информационно-измерительной системы с помощью библиотеки Simgrid // Проблемы автоматики и управления. –Бишкек: Илим, №1 (36), 2019. – С.90–99.