ИУ7-54Б, 16_KOZ, Булдаков, Турчанский, Рунов

ОПРЕДЕЛЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие термины с соответствующими определениями.

Вычислительный узел (узел) — устройство, выполняющее основную логику обработки запроса [1].

Хеш-таблица — структура данных, реализующая интерфейс ассоциативного массива, позволяет хранить пары (ключ, значение) [2].

—

ВВЕДЕНИЕ

В современном обществе практически все общение и взаимодействие с приложениями осуществляются через интернет. Сетевые приложения востребованы как обычными пользователями, так и крупными корпорациями, проводящими сложные вычисления и обмен данными. Одной из задач, с которыми сталкиваются интернет-компании, является обеспечение бесперебойного доступа клиентов к предоставляемым компаниями интернет-ресурсам [3].

За последние 5 лет количество пользователей в интернете выросло на 30 процентов [4], что привело к резкому увеличению нагрузки на многие системы. Обработка выросшей нагрузки и обеспечение бесперебойного доступа к интернет-ресурсам, требует добавления в систему новых вычислительных узлов, для эффективной работы которых, необходимо осуществлять балансировку нагрузки [1; 5—7].

Целью данной работы является описание методов балансировки нагрузки в высоконагруженных системах.

Балансировка нагрузки — это механизм приблизительного выравнивания рабочей нагрузки между всеми узлами системы [8].

Для достижения поставленной цели необходимо выполнить следующие задачи:

— описать принципы балансировки нагрузки;

— классифицировать методы балансировки нагрузки;

— описать алгоритмы, используемые для балансировки нагрузки.

1 Аналитический раздел

С ростом числа запросов к системе, встает вопрос о ее масштабировании. Масштабирование — это процесс роста системы со временем, для эффективной обработки все большего и большего количества запросов в единицу времени [9]. Выделяют два вида масштабирования: горизонтальное и вертикальное [1; 7; 10]. Вертикальное масштабирование происходит за счет увеличения мощности вычислительного узла. Однако, использования только такого подхода часто не хватает, поскольку постоянно увеличивая мощность, однажды будет достигнут «потолок» производительности и дальнейшие аппаратные улучшения будут недоступны. В таком случае для дальнейшего роста производительности применяют горизонтальное масштабирование, которое заключается в добавлении новых вычислительных узлов, выполняющих одинаковые функции. Для расширения возможностей горизонтального масштабирования используются балансировщики нагрузки [7; 10].

Балансировщик нагрузки — это программа, принимающая весь входящий трафик запросов и распределяющая его между несколькими вычислительными системами с целью оптимизации использования ресурсов, сокращения времени обслуживания запросов, а также обеспечения отказоустойчивости [10].

1.1 Постановка задачи

В вычислительных сетях, использование распределения нагрузки для оптимизации использования ресурсов, называют балансировкой нагрузки [6].

Постановка задачи балансировки нагрузки выглядит следующим образом: имеется множество, состоящее из n запросов, которое должно быть обслужено M узлами. Каждый узел может обслуживать не более одного запроса в каждый момент времени. Каждый запрос обслуживается не более, чем одним узлом в каждый момент времени, а процесс обслуживания запроса не может быть прерван.

Под расписанием понимается функция, которая каждому узлу l и моменту времени t сопоставляет запрос, обслуживаемый узлом l в момент времени t, либо указывает, что узел l в момент t простаивает. Каждому запросу i сопоставлена неубывающая функция штрафа $\phi_i(t)$. Тогда решением задачи балансировки нагрузки является составление расписания s, которое миними-

зирует выражение (1.1) [6].

$$F_{max} = \max_{i \in n}\{\phi_i(t_i(s))\}, \tag{1.1}$$

где $t_i(s)$ — момент завершения обслуживания запроса i при расписании s.

1.2 Алгоритмы решения задачи балансировки

Балансировщик нагрузки работает по одному из алгоритмов, решающих задачу балансировки. На вход этому алгоритму подается некоторое число запросов, приходящих в систему и набор вычислительных узлов, которыми располагает система. Задача алгоритма сводится к минимизации времени обработки запросов, за счет распределения запросов по вычислительным узлам.

Для анализа алгоритмов балансировки могут быть выделены следующие параметры [1]:

— точность прогнозирования — степень соответствия расчетных результатов работы алгоритма их фактическому значению;

— отказоустойчивость — показывает устойчивость алгоритма к возникновению разнообразных ошибок;

— время обработки нового запроса — время от поступления нового запроса до его перенаправления к цели.

Методы балансировки условно разделяют на статические и динамические [1; 11; 12].

При динамической балансировке нагрузки, распределение запросов происходит на основе собранной информации об узлах.

В статических алгоритмах, запрос распределяется в узел при появлении в балансировщике, а состояние узлов не оказывает влияния на это распределение [1; 12].

1.3 Статическая балансировка

Статическая балансировка — это метод распределения нагрузки на узлы, основанный на заранее определенных параметрах.

К самым распространенным статическим алгоритмам относят: Round Robin и его модификацию Weighted Round Robin [1; 7; 10; 11].

1.3.1 Round Robin

Round Robin — это алгоритм балансировки нагрузки, который направляет каждый следующий запрос на новый вычислительный узел по заранее определенному порядку [10].

Пусть имеется N запросов и M узлов. Алгоритм состоит из следующих шагов.

1) Сформировать массив, содержащий узлы.
2) Создать переменную $i = 0$.
3) Для каждого запроса из N:
   - отправить текущий запрос на i-й узел в массиве;
   - увеличить значение i;
   - если $i >= M$, то $i = 0$.

Особенности алгоритма Round Robin:

- высокая степень точности прогнозирования;
- невозможно отследить вышел ли из строя узел, в результате, возможно, что на неработающий узел будут посылаться запросы, т. е. низкая отказоустойчивость;
- никакие затратные по времени операции не производятся и попавший в балансировщик запрос практически сразу направляется в узел;
- различия в технических характеристиках узлов не учитываются, что может привести к неравномерному распределению нагрузки.

Weighted Round Robin

Алгоритм Weighted Round Robin представляет собой модификацию алгоритма Round Robin, в которой каждому узлу вручную назначается некоторый

параметр, называемый весом, с помощью которого можно варьировать количество запросов, отправляемых на конкретный узел [7]. Если помимо множества запросов и узлов, задан массив весов $weights$, длины M. Тогда алгоритм состоит из следующих шагов.

1) Сформировать массив $nodes$ содержащий узлы, при этом повторить каждый j-й узел $weights[j]$ раз.

2) Сохранить длину массива $nodes$ в переменную L.

3) Создать переменную $i = 0$.

4) Для каждого запроса из N:

   — распределить текущий запрос на i-й узел в массиве;

   — увеличить значение i;

   — если $i >= L$, то $i = 0$.

Особенности алгоритма Weighted Round Robin:

— высокая степень точности прогнозирования;

— низкая отказоустойчивость, поскольку невозможно отследить вышел ли из строя некоторый узел, при этом, если вышел из строя узел с самым высоким весом, то на него все еще будет посылаться большее число запросов;

— благодаря весам, возможно осуществить настройку алгоритма таким образом, чтобы он учитывал различия в технических характеристиках узлов.

1.4 Динамическая балансировка

Особенностью динамических алгоритмов балансировки является необходимость в постоянном обмене актуальной информацией о узлах. Простой способ периодического децентрализованного обмена информацией о состоянии заключается в том, что каждый узел периодически отправляет свое текущее состояние всем другим узлам [11].

К динамическим алгоритмам относятся следующие [11; 13; 14]:

— Dynamic Round Robin;

— Least Connections;

— Weighted Least Connections;

— Least Response Time;

— Хеширование на основе IP-адреса;

— Хеширование на основе URL-адреса;

— Метод фиксированных весов.

1.4.1 Dynamic Round Robin

Алгоритм Dynamic Round Robin динамически изменяет расписание, т. е. распределение запросов по узлам, в зависимости от текущих характеристик узлов [11]. Dynamic Round Robin может исключать недоступные узлы, перенаправляя задачи на доступные узлы, что позволяет избежать проблем при работе с неисправными узлами. В алгоритме Dynamic Round Robin на расписание могут влиять следующие характеристики [1; 7; 13]:

— количество соединений;

— среднее значение загрузки системы за период в 1 минуту, строится на основе процессов, т. е. программ в стадии выполнения, стоящих в очереди ожидания ресурсов, выражается как отношение количества ожидающих процессов к общему количеству ядер;

— загрузка процессора узла в текущий момент времени, выраженная в процентах;

— использование памяти узла, выраженное в процентах относительно общего количества;

— географическое расстояние между узлами.

Алгоритм Dynamic Round Robin, выбирающий узлы по их текущей нагрузке, состоит из следующих шагов для каждого входящего запроса [1].

1) Установить переменную *target* на первый доступный узел.

2) Цикл по всем доступным узлам, кроме первого:

   — если нагрузка на текущий рассматриваемый узел меньше нагрузки узла *target*, то установить *target* на текущий узел.

3) Отправить запрос на узел *target*.

Особенности алгоритма Dynamic Round Robin:

— низкая точность прогнозирования, поскольку распределение запросов сильно зависит от внешних факторов;

— высокая отказоустойчивость, поскольку в алгоритме учитывается ситуация отказа узлов.

1.4.2 Least Connections

Алгоритм Least Connections распределяет нагрузку между узлами, в зависимости от количества активных соединений, обслуживаемых каждым узлом. Узел с наименьшим числом соединений будет обрабатывать следующий запрос, а узлы с большим числом соединений будут перераспределять свою нагрузку на узлы с меньшей загрузкой [15].

Если имеется N запросов, M узлов и для каждого узла есть количество активных соединений *conns*. Тогда алгоритм состоит из следующих шагов:

1) сформировать массив, содержащий узлы;

2) установить указатель $target$ на первый узел;

3) пройтись циклом по всем узлам массива, кроме первого:

   — если *conns* текущего узла меньше *conns* узла $target$, то установить $target$ на текущий узел;

4) отправить запрос на узел $target$.

Особенности алгоритма Least Connections:

— низкая степень прогнозирования;

— низкая стабильность;

— высокая отказоустойчивость, поскольку постоянно собирается информация об узлах, и, в случае отказа, система перераспределит ресурсы;

— высокая потребность в ресурсах, поскольку необходимо постоянно собирать информацию о узлах в реальном времени;

— высокое время обработки нового запроса, поскольку балансировщику нагрузки необходимо время, чтобы правильно перенаправить задачу.

Weighted Least Connections

Данный алгоритм комбинирует принципы алгоритмов Least Connections и Weighted Round Robin [7]. Он учитывает как веса узлов, так и количество активных соединений. Новое сетевое подключение предоставляется узлу, который имеет минимальное отношение количества текущих активных подключений к его весу [14].

Если имеется N запросов, M узлов и для каждого узла есть количество соединений $conns$ и вес $weight$. Тогда алгоритм состоит из следующих шагов:

1) сформировать массив, содержащий узлы;

2) установить указатель $target$ на первый узел;

3) пройтись циклом по всем узлам массива, кроме первого:

   — если отношение $conns$ и $weight$ текущего узла меньше отношения $conns$ и $weight$ узла $target$, то установить $target$ на текущий узел;

4) отправить запрос на узел $target$.

Особенности алгоритма Weighted Least Connections:

— низкая степень прогнозирования;

— низкая стабильность;

— высокая отказоустойчивость;

— высокая потребность в ресурсах;

— высокое время обработки нового запроса;

— благодаря весам, возможно осуществить настройку алгоритма таким образом, чтобы он учитывал различия в технических характеристиках узлов.

1.4.3 Least Response Time

Данный алгоритм имеет схожесть с алгоритмом Least Connections, только при распределении нагрузки он руководствуется наименьшим временем ответа узла. При выборе учитывается производительность узлов и балансировщик стремится направить запрос к наиболее подходящему узлу [16].

Если имеется N запросов, M узлов и для каждого узла есть время ответа на предыдущий запрос $time$. Тогда алгоритм состоит из следующих шагов:

1) сформировать массив, содержащий узлы;

2) установить указатель $target$ на первый узел;

3) пройтись циклом по всем узлам массива, кроме первого:

   — если $time$ текущего узла меньше $time$ узла $target$, то установить $target$ на текущий узел;

4) отправить запрос на узел $target$.

Особенности алгоритма Least Response Time:

— низкая степень прогнозирования;

— низкая стабильность;

— высокая отказоустойчивость;

— высокая потребность в ресурсах;

— высокое время обработки нового запроса;

— если время ответа каждого узла одинаково, то алгоритм следует выбору по правилам алгоритма Round Robin.

Методы на основе хеширования

Методы балансировки нагрузки на основе хеширования работают по общему принципу:

1) Из пришедшего запроса выбрать информацию (например, IP-адрес или URL-адрес), которая считается ключом хеш-функции в рамках данного алгоритма.

2) На основе ключа вычислить значение хеш-функции, которое соответствует идентификатору узла, на который следует перенаправить запрос для его обработки.

3) Перенаправить запрос на узел, чей идентификатор был вычислен ранее.

Хеширование на основе IP-адреса

Алгоритм балансировки нагрузки «Хеширование на основе IP-адреса» работает по общему принципу методов балансировки нагрузки на основе хеширования. Ключом хеш-функции в данном алгоритме считается IP-адрес источника запроса [16—18]. Запросы, имеющие один и тот же IP-адрес, будут обслужены одним и тем же узлом. То есть, если имеются запросы r_1, r_2, узлы l_1, l_2, моменты времени t_1, t_2 и функция расписания s, то, в соответствии с данным алгоритмом, будет выполнено следующее:

$$(\forall t_1, t_2)\left(\begin{cases} s(l_1, t_1) = r_1, \\ s(l_2, t_2) = r_2, \\ r_1.ip\_address = r_2.ip\_address. \end{cases} \Rightarrow l_1 = l_2\right) \tag{1.2}$$

Особенности алгоритма «Хеширование на основе IP-адреса»:

— алгоритм гарантирует, что все запросы от одного и того же пользователя направляются на тот же сервер;

— алгоритм предсказуем;

— добавление новых серверов потребует лишь изменения хеш-функции для корректной работы алгоритма;

— неравномерная нагрузка на узел, если запросы начинают приходить из сети, использующей NAT, с большим количеством пользователей.

Хеширование на основе URL-адреса

Алгоритм балансировки нагрузки «Хеширование на основе URL-адреса» работает по общему принципу методов балансировки нагрузки на основе хеширования. Ключом хеш-функции в данном алгоритме считается URL-адрес, к которому обращается источник запроса [17—19]. Запросы к одному и тому же URL-адресу, будут обслужены одним и тем же узлом. То есть, если имеются запросы r_1, r_2, узлы l_1, l_2, моменты времени t_1, t_2 и функция расписания s, то, в соответствии с данным алгоритмом, будет выполнено следующее:

$$(\forall t_1, t_2)\left(\begin{cases} s(l_1, t_1) = r_1, \\ s(l_2, t_2) = r_2, \\ r_1.url\_address = r_2.url\_address. \end{cases} \Rightarrow l_1 = l_2\right) \tag{1.3}$$

Особенности алгоритма «Хеширование на основе URL-адреса»:

— алгоритм гарантирует, что все запросы к одному и тому же URL направляются на тот же сервер;

— алгоритм предсказуем;

— изменение структуры URL может потребовать перенастройки балансировщика;

— популярные URL могут создавать неравномерную нагрузку на узлы.

Метод фиксированных весов

В методе фиксированных весов, администратор назначает каждому узлу вес, после чего все запросы будут приходить на узел с максимальным весом [18]. Если узел перестаёт справляться с нагрузкой, запросы начинают перенаправляться на узел, с весом меньше.

Шаги инициализации алгоритма:

1) N = количество узлов

2) nodes = массив узлов

3) weights = массив весов таких, что weights[i] — вес узла i, назначенный администратором

4) i = 1

5) Пока i <= N:

   — nodes[i].weight = weights[i]

   — i = i + 1

Шаги работы алгоритма:

1) w = max(weights)

2) request_sent_flag = 0

3) Пока request_sent_flag = 0 и w > 0:

   — node = узел с весом w

   — Если узел node работоспособен, перенаправить запрос узлу node, request_sent_flag = 1

   — Иначе, w = w - 1

Таким образом, если имеется запрос r, узел l, момент времени t, функция расписания s, и выполняется равенство $s(l, t) = r$ то, l — узел с наибольшим весом, доступный в момент времени t.

Особенности метода фиксированных весов:

— алгоритм гарантирует, что все запросы будут направляться на доступный в текущий момент времени узел с максимальным весом;

— алгоритм предсказуем;

— веса узлов назначаются вручную;

— вес узла не меняется в процессе работы.

ЗАКЛЮЧЕНИЕ

В результате проделанной научно-исследовательской работы, была описана предметная область балансировки нагрузки.

Было описано, что задача балансировки состоит в минимизации времени обслуживания запросов, а основными параметрами алгоритмов являются: отказоустойчивость, точность прогнозирования и время обработки запроса. Были рассмотрены такие алгоритмы статической балансировки, как Round Robin и Weighted Round Robin. Также были приведены основные различия между статическими и динамическими алгоритмами, в качестве примера динамических алгоритмов был рассмотрен алгоритм Dynamic Round Robin.

В результате, была достигнута цель научно-исследовательской работы, а именно, описаны методы балансировки нагрузки в высоконагруженных системах.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Шуляк А.* Сравнительный анализ алгоритмов балансировки нагрузки в среде облачных вычислений // Научный журнал. — 2021. — № 6.
2. Хеш-таблица [Электронный ресурс]. — Режим доступа: `https://neerc.ifmo.ru/wiki/` (дата обращения: 07.11.2023).
3. *Onay Dogan B.*, *Camdereli M.* Digital Communication Activities of Corporations in the Context of Corporate Communication and Governance // Online Journal of Communication and Media Technologies. — 2015. — С. 61—77.
4. Digital 2023: Global Overview Report [Электронный ресурс]. — Режим доступа: `https://datareportal.com/reports/digital-2023-global-overview-report` (дата обращения: 07.11.2023).
5. *Бершадский А. М.*, *Курилов Л. С.*, *Финогеев А. Г.* Исследование стратегий балансировки нагрузки в системах распределенной обработки данных // Известия вузов. Поволжский регион. Технические науки. — 2009. — № 4.
6. *Бершадский А. М.*, *Курилов Л. С.*, *Финогеев А. Г.* Разработка системы балансировки нагрузки // Гаудеамус. — 2012. — № 20.
7. *Павликов М. К.* Алгоритм распределения нагрузки в программной системе, построенной на основе протокола HDP // Вестн. Том. гос. ун-та. Управление, вычислительная техника и информатика. — 2017. — № 40.
8. *Асадова Шабнам Р. К.* Руководство по динамической балансировке нагрузки в распределенных компьютерных системах // ELS. — 2023.
9. *Макаров Д. А.*, *Шибанова А. Д.* Масштабирование веб-приложений // Теория и практика современной науки. — 2021. — № 1.
10. *Гусев А. О.*, *Костылева В. В.*, *Разин И. Б.* Сравнение алгоритмов балансировки нагрузки // Инновационное развитие техники и технологий в промышленности (ИНТЕКС-2020). — 2020. — № 1.
11. *Симаков Д. В.* Управление трафиком в сети с высокой динамикой метрик сетевых маршрутов // Вестник евразийской науки. — 2016. — № 1.

12. *Кхаинг М. Т.*, *Лупин С.* Сравнительный анализ методов оценки производительности узлов в распределнных системах // International Journal of Open Information Technologies. — 2023. — № 6.

13. *Haroon M.* Dynamic Load balancing by Round Robin and Warshall Algorithm in Cloud Computing // International Journal of Innovative Technology and Exploring Engineering. — 2021. — № 62.

14. *Gurasis S.*, *Kamalpreet K.* An Improved Weighted Least Connection Scheduling Algorithm for Load Balancing in Web Cluster Systems // IRJET. — 2018. — № 5.

15. *Husain N.*, *Timotius W.* Analisis Algoritma Round Robin, Least Connection, Dan Ratio Pada Load Balancing Menggunakan Opnet Modeler // Informatika: Jurnal Teknologi Komputer dan Informatika. — 2016. — Т. 12, № 1.

16. Amazon Web Services [Электронный ресурс]. — Режим доступа: `https://aws.amazon.com/what-is/load-balancing` (дата обращения: 12.11.2023).

17. What Is Load Balancing? [Электронный ресурс]. — Режим доступа: `https://www.nginx.com/resources/glossary/load-balancing` (дата обращения: 12.11.2023).

18. Load Balancing Algorithms and Techniques [Электронный ресурс]. — Режим доступа: `https://kemptechnologies.com/load-balancer/load-balancing-algorithms-techniques` (дата обращения: 04.01.2024).

19. *Ramirez N.* Load Balancing with HAProxy: Open-Source Technology for Better Scalability, Redundancy and Availability in Your IT Infrastructure //. — Nick Ramirez, 2016. — С. 172. — ISBN 9781519073846.