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

СОДЕРЖАНИЕ

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 для реализации ассоциативных массиов [2].

NAT — Network Address Translation, метод трансляции сетевых адресов [3].

ВВЕДЕНИЕ

В современном мире люди часто используют интернет для взаимодействия с приложениями. 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 к предоставляемым интернет-ресурсам [4].

За последние 5 лет количество пользователей в интернете выросло на 30 процентов, что привело к резкому увеличению нагрузки на многие системы [5]. 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; 6—8].

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

Балансировка нагрузки — это механизм приблизительного выравнивания рабочей нагрузки между всеми узлами системы [7; 9].

Для достижения поставленной цели необходимо выполнить следующие задачи:

— описать основные подходы к решению задачи балансировки нагрузки;

— сформулировать критерии сравнения методов решения задачи балансировки нагрузки;

— классифицировать методы решения задачи 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 [10]. Выделяют два вида масштабирования: горизонтальное и вертикальное [1; 8; 11]. Вертикальное масштабирование происходит за счет увеличения мощности вычислительного узла. Горизонтальное масштабирование, заключается в добавлении новых вычислительных узлов, выполняющих одинаковые функции. Для расширения возможностей горизонтального масштабирования используются балансировщики нагрузки [8; 11].

Балансировщик нагрузки — это программа, принимающая весь входящий трафик запросов и распределяющая его между несколькими вычислительными узлами с целью оптимизации использования ресурсов, сокращения времени обслуживания запросов, а также обеспечения отказоустойчивости [11].

1.1 Постановка задачи

Постановка задачи балансировки нагрузки формулируется 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 минимизирует выражение (1.1) [7].

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

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 подаются набор запросов, приходящих в систему и набор вычислительных узлов, которыми система располагает. Задача алгоритма сводится к минимизации времени обработки запросов за счет распределения запросов по вычислительным узлам.

Для анализа алгоритмов балансировки выделяют следующие критерии [1]:

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

— отказоустойчивость — показатель устойчивости 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; 12; 13].

1.3 Статическая балансировка

Статическая балансировка — это метод распределения нагрузки на узлы, основанный на заранее определенных параметрах [1; 14].

К статическим алгоритмам балансировки относят [1; 8; 15—17]

— алгоритмы циклического перебора (англ. Round Robin);

— алгоритмы на основе хеширования.

1.3.1 Алгоритмы циклического перебора

В алгоритмах циклического перебора каждый следующий запрос отправляется на новый вычислительный узел по заранее определенному порядку [11].

Если имеется N запросов и M узлов, то 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 равномерного распределения:

— 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 в балансировщике;

— различия в технических характеристиках узлов не учитываются, что может привести к неравномерному распределению нагрузки.

Взвешенный алгоритм циклического перебора

Алгоритм взвешенного циклического перебора (англ. Weighted Round Robin) представляет собой модификацию алгоритма равномерного распределения, в которой каждому узлу вручную назначается некоторый параметр, называемый весом, с помощью которого можно варьировать количество запросов, отправляемых на конкретный узел [8]. Если помимо множества запросов и узлов, задан массив весов $weights$, длины M, то 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 взвешенного циклического перебора:

— 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.

1.3.2 Алгоритмы на основе хеширования

Алгоритмы 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 [15; 16; 18]. Запросы, имеющие один и тот же IP-адрес, будут обслужены одним и тем же узлом. 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:

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

Особенности алгоритма «Хеширование на основе IP-адреса»:

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

— алгоритм обладает высокой степенью точности прогнозирования;

— 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 [15; 17; 18]. 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:

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

Особенности алгоритма «Хеширование на основе URL-адреса»:

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

— алгоритм обладает высокой степенью точности прогнозирования;

— изменение структуры URL может потребовать перенастройки балансировщика;

— популярные URL могут создавать неравномерную нагрузку на узлы.

1.4 Динамическая балансировка

Динамическая балансировка — это метод распределения нагрузки на узлы, который предусматривает перераспределение вычислительной нагрузки на узлы во время работы [14].

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 [12].

К динамическим алгоритмам относят [12; 19; 20]

— алгоритм динамического циклического перебора (англ. Dynamic Round Robin);

— алгоритмы наименьших соединений (англ. Least Connections);

— алгоритм наименьшего времени ответа (англ. Least Response Time);

— алгоритм фиксированных весов (англ. Fixed Weighting);

— алгоритм 2 (N) случайных выборов (англ. Random 2 (N) choices);

— алгоритм на основе ресурсов (англ. Resource based).

1.4.1 Алгоритм динамического циклического перебора

Динамический алгоритм циклического перебора изменяет расписание, т. е. распределение запросов по узлам, в зависимости от текущих характеристик узлов [12]. Алгоритм может исключать недоступные узлы, перенаправляя

задачи на доступные узлы, что позволяет избежать проблем при работе с неисправными узлами. В алгоритме динамического циклического перебора на расписание могут влиять следующие характеристики [1; 8; 19]:

— 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 динамического циклического перебора, 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 динамического циклического перебора:

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

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

1.4.2 Алгоритмы наименьших соединений

Алгоритм наименьших соединений 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 [21].

Если имеется N запросов, M узлов и для каждого узла есть количество активных соединений $conns$, то 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 наименьших соединений:

— 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.

Алгоритм взвешенных наименьших соединений

Данный алгоритм комбинирует принципы алгоритма наименьших соединений и алгоритма взвешенного циклического перебора [8]. 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 [20].

Если имеется N запросов, M узлов и для каждого узла есть количество соединений $conns$ и вес $weight$, то 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 взвешенных наименьших соединений:

— 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 Алгоритм наименьшего времени ответа

Данный алгоритм имеет схожесть с алгоритмом наименьших соединений, 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 $time$, то 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 наименьшего времени ответа:

— 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 циклического перебора.

1.4.4 Алгоритм фиксированных весов

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

Если имеется M узлов, записанных в массив $nodes$ и $weights$ — массив весов, назначенных администратором, то алгоритм для каждого приходящего запроса состоит из следующих шагов:

1) записать в w максимальное значение $weights$;

2) $request\_sent\_flag = 0$;

3) пока $request\_sent\_flag = 0$ и $w > 0$:

— записать в $node$ узел, вес которого равен w;

— если узел $node$ может принять запрос то, перенаправить запрос узлу $node$, установить значение $request\_sent\_flag = 1$;

— 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 алгоритма фиксированных весов:

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

— алгоритм обладает низкой степенью прогнозируемости;

— веса узлов назначаются вручную;

— вес узла не меняется в процессе работы.

1.4.5 Алгоритм 2 (N) случайных выборов

Алгоритм 2 (N) случайных выборов — алгоритм, при котором определяется нагрузка $N \geq 2$ серверов, выбранных случайным образом, и запрос отправляется на наименее загруженный из них. В случае $N = 2$ максимальная нагрузка на n серверов с большой вероятностью составит $\Theta(\log \log n)$ [22].

Данный метод может быть использован, когда запрос требуется отправить на наименее загруженный сервер. Однако, полная информация о загрузке всех серверов может оказаться дорогостоящей для получения. Например, для получения загрузки на сервер может потребоваться отправка сообщения и ожидание ответа, обработка прерывания сервером [22].

Альтернативный подход при котором информация о загрузке серверов не требуется, заключается в том, чтобы распределить запрос на случайный сервер. В таком случае максимальная нагрузка на n серверов с высокой вероятностью составит $\Theta(\log n / \log \log n)$ [22].

Если имеется K запросов и M узлов, при этом $2 <= N <= M$, то алгоритм состоит из следующих шагов:

1) сформировать массив $nodes$, содержащий узлы;

2) для каждого запроса из K:

   — сформировать массив $randoms$, содержащий N узлов, выбранных случайным образом из массива $nodes$;

   — установить переменную $target$ на первый доступный узел — $randoms[0]$;

   — создать переменную $i = 1$;

   — пока $i < M$:

      – если нагрузка узла $randoms[i]$ меньше нагрузки узла $target$, то установить $target$ на узел $randoms[i]$;

      – увеличить значение i;

   — отправить запрос на узел $target$.

1.4.6 Алгоритм на основе ресурсов

Алгоритм на основе ресурсов — алгоритм, при котором трафик распределяется балансировщиком нагрузки, в зависимости от текущей нагрузки на сервер [16].

Специализированное программное обеспечение, называемое агентом, запускается на каждом сервере и рассчитывает использование ресурсов сервера, таких как его вычислительная мощность и память. Затем агент проверяется балансировщиком нагрузки на наличие достаточного количества свободных ресурсов перед распределением трафика на данный сервер [16].

Если имеется N запросов и M узлов, то алгоритм состоит из следующих шагов:

1) сформировать массив $nodes$, содержащий узлы;

2) для каждого запроса из N:

   — сформировать массив $resources$, содержащий информацию об использовании ресурсов, соответствующим узлом;

   — установить переменную $target$ на первый доступный узел — $resources[0]$;

   — создать переменную $i = 0$;

— пока $i < M$:

  – узел $resources[i]$ обладает достаточным количеством свободных ресурсов для выполнения запроса:

    * установить $target$ на узел $resources[i]$;

    * прекратить выполнение цикла;

  – увеличить значение i;

— отправить запрос на узел $target$.

ЗАКЛЮЧЕНИЕ

В процессе выполнения данной научно-исследовательской работы была описана предметная область балансировки нагрузки, формализована задача балансировки нагрузки, а также были:

— описаны основные подходы к решению задачи балансировки нагрузки;

— сформулированы критерии сравнения методов решения задачи балансировки нагрузки;

— классифицированы методы решения задачи балансировки нагрузки.

Таким образом, все задачи для достижения цели данной научно-исследовательской работы были решены, и цель работы была достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Шуляк А.* Сравнительный анализ алгоритмов балансировки нагрузки в среде облачных вычислений // Научный журнал. — 2021. — № 6.
2. Hash Tables [Электронный ресурс]. — Режим доступа: `https://brilliant.org/wiki/hash-tables/` (дата обращения: 07.11.2023).
3. Understanding NAT type on PC [Электронный ресурс]. — Режим доступа: `https://www.ubisoft.com/en-us/help/connectivity-and-performance/article/understanding-nat-type-on-pc/000096222` (дата обращения: 04.01.2024).
4. *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.
5. Digital 2023: Global Overview Report [Электронный ресурс]. — Режим доступа: `https://datareportal.com/reports/digital-2023-global-overview-report` (дата обращения: 07.11.2023).
6. *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.
7. *Бершадский А. М.*, *Курилов Л. С.*, *Финогеев А. Г.* Разработка системы балансировки нагрузки // Гаудеамус. — 2012. — № 20.
8. *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.
9. *Асадова Шабнам Р. К.* Руководство по динамической балансировке нагрузки в распределенных компьютерных системах // ELS. — 2023.
10. *Макаров Д. А.*, *Шибанова А. Д.* Масштабирование веб-приложений // Теория и практика современной науки. — 2021. — № 1.

11. *Гусев А. О.*, *Костылева В. В.*, *Разин И. Б.* Сравнение алгоритмов балансировки нагрузки // Инновационное развитие техники и технологий в промышленности (ИНТЕКС-2020). — 2020. — № 1.

12. *Симаков Д. В.* Управление трафиком в сети с высокой динамикой метрик сетевых маршрутов // Вестник евразийской науки. — 2016. — № 1.

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

14. Балансировка нагрузки в распределенных системах [Электронный ресурс]. — Режим доступа: `https://intuit.ru/studies/courses/1146/238/lecture/6153?ysclid=lr4rglhejo908154193` (дата обращения: 04.01.2024).

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

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

17. *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.

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

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

21. *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.

22. The Power of Two Random Choices: A Survey of Techniques and Results. — Режим доступа: `https://www.eecs.harvard.edu/~michaelm/postscripts/handbook2001.pdf` (дата обращения: 07.10.2023).