ВЕСТНИК ТОМСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА
2017 Управление, вычислительная техника и информатика № 40

УДК 004.75
DOI: 10.17223/19988605/40/9

М.К. Павликов

АЛГОРИТМ РАСПРЕДЕЛЕНИЯ НАГРУЗКИ В ПРОГРАММНОЙ СИСТЕМЕ, ПОСТРОЕННОЙ НА ОСНОВЕ ПРОТОКОЛА HDP

Предложен алгоритм эффективного распределения нагрузки в гетерогенной программной системе, построенной на основе протокола HDP. Данный алгоритм позволил получить равномерное распределение нагрузки в произвольный момент времени эксплуатации программной системы за счет использования данных обратной связи от запущенных узлов. Использование алгоритма позволило обеспечить отказоустойчивость системы и повысить толерантность к изменению аппаратного обеспечения. Даны описания алгоритма и его сравнение с другими алгоритмами распределения нагрузки в программной системе.
Ключевые слова: HDP; распределенная программная система; балансировка нагрузки.

На сегодняшний день большинство программных систем подвергаются внушительным функциональным нагрузкам. Порой система развивается настолько молниеносно, что наращивание мощности аппаратного обеспечения не позволяет адекватно справляться с возрастающей нагрузкой. А в случае высоконагруженных программных систем одного физического узла попросту не хватит, чтобы корректно обработать все поступающие запросы. Поэтому такие системы строятся, как правило, из одинаковых копий узлов, каждый из которых выполняет одинаковые функции, что приводит к возникновению задач маршрутизации и равномерного распределения нагрузки между существующими однотипными узлами программной системы. Также остро стоит вопрос о возможности поддержки: ввод нового узла или вывод старого узла из эксплуатации без полного выключения системы. Все эти задачи ложатся на методы и алгоритмы балансировки, но большинство таких методов и алгоритмов при распределении нагрузки не учитывают реальную ситуацию в текущий момент времени: при изменении набора узлов в них вносятся изменения, а дальше система функционирует без стороннего вмешательства.

Важнейшим фактором, приводящим к появлению копий одних и тех же узлов, является и потребность отсутствия единой точки отказа. Система должна быть устойчива к выходу из строя одного или нескольких узлов и продолжать корректно реагировать на поступающие запросы. Такие системы могут быть построены на основе протокола взаимодействия HDP, в котором реализован алгоритм балансировки нагрузки с учетом данных обратной связи, который позволяет в режиме реального времени менять нагрузку на тот или иной узел, чтобы поддерживать оптимальную нагрузку на программную систему в целом.

1. Алгоритмы распределения нагрузки в распределенных системах

Балансировка нагрузки – это метод распределения запросов между несколькими одинаковыми узлами распределенной вычислительной системы с целью оптимизации использования всех узлов совместно. Существуют различные алгоритмы балансировки нагрузки, которые имеют разное применение в различных областях, их целью также является обеспечение отказоустойчивости системы в целом.

Алгоритм round robin представляет из себя циклический перебор всех узлов по кругу, каждый следующий запрос отправляется на следующий в наборе узел. Главное достоинство – это простота алгоритма, что позволяет использовать его во многих системах, и архитектура таких систем остается максимально простой, так как эксплуатационные особенности системы не вносят дополнительного вклада в сложность архитектуры. Из недостатков алгоритма стоит выделить неравномерное распределение нагрузки, если узлы сети имеют разные технические характеристики. Технически более мощный узел будет лучше справляться с запросами, чем остальные, которые будут более нагружены. Также не реше-

на проблема недоступного узла, т.е. узла, который во время эксплуатации системы внезапно вышел из строя.

Алгоритм weighted round robin представляет собой модификацию алгоритма round robin, в которой каждому узлу вручную назначается некий вес – вероятность, с которой балансировщик нагрузки в следующий раз выберет именно этот узел. По сравнению с round robin в данном алгоритме удалось компенсировать возможные технические отличия узлов сети. Из недостатков алгоритма стоит выделить отсутствие решения всех задач отказоустойчивости, ведь если узел сети с высоким приоритетом перестает быть доступным, на него все равно отправляется большинство запросов.

2. Алгоритм распределения нагрузки feedback node в программной системе, использующей протокол HDP для взаимодействия узлов

Протокол HDP представляет собой протокол для обмена гетерогенными данными в распределенной вычислительной среде. Протокол использует жестко структурированные данные в формате JSON со специальными служебными метками согласно спецификации протокола. Согласно спецификации протокола все взаимодействующие узлы в HDP делятся на два типа: клиент и поставщик. Клиент – узел, запрашивающий данные. Поставщик – узел, предоставляющий доступ к данным и функциям их обработки.

В протоколе заложена возможность указания нескольких узлов: однотипных копий одних и тех же сервисов для того, чтобы была возможность балансировать нагрузку между копиями, выдающими одинаковые данные и предоставляющими одинаковые функции над данными.

Предлагаемый автором алгоритм feedback node является алгоритмом распределения нагрузки в программной системе, построенной на основе протокола HDP. Он учитывает гетерогенные данные о текущей загрузке каждого узла и на основе этих данных распределяет между ними нагрузку. На отслеживаемый узел устанавливается агент, который передает серверу данные об общей загрузке, загрузке процессора и использовании памяти на данном узле. Балансировщик HDP на основе этих данных принимает решение о передаче запроса тому или иному узлу в зависимости от его загруженности в текущий момент времени.

Среди выбранных отслеживаемых метрик используются:

– LA (Load Average) – среднее значение загрузки системы за период в 1 минуту. Значение LA строится на основе процессов, стоящих в очереди ожидания ресурсов, поэтому данное значение трактуется в зависимости от количества ядер процессора: значением по данной метрике является отношение количества ожидающих процессов к общему количеству ядер;

– CPU (CPU utilization) – загрузка процессора в текущий момент времени, выраженная в процентах;

– MU (Memory usage) – использование памяти, выраженное в процентах относительно общего количеств.

Стоит отметить, что алгоритм может подстраиваться к исключению указанных выше метрик или добавлению новых.

В любой момент времени на сервере поддерживается в актуальном состоянии список приоритетных узлов, в котором перечислены адреса всех узлов в определенном порядке. Один и тот же узел может быть указан в списке многократно. Балансировщик нагрузки обходит этот список по кругу и передает каждый следующий запрос последовательно следующему узлу, указанному в списке.

У этого списка есть предельное время жизни (в секундах), в течение которого список не обновляется и признается актуальным. Предельное время жизни определяется на предположении, что за определенный промежуток времени значения отслеживаемых метрик кардинальным образом не изменятся, поэтому не имеет смысла нагружать узлы лишним получением метрик, тем самым повышая нагрузку на узел.

Алгоритм распределения нагрузки feedback node:

1. Присвоить N количество всех доступных однотипных узлов в программной системе.

2. Отправить запросы получения текущих значений *LA, CPU, MU* всем узлам с установленными агентами.

3.1. Сравнить значения параметра *LA* у всех узлов.

3.1.1. Если значение параметра *LA* хотя бы у 2 узлов отличается, то перейти к пункту 3.1.2., иначе перейти к пункту 3.2.

3.1.2. Вычислить *SLA* – сумму значений всех параметров:

$$SLA = \sum_{i=1}^{N} LA_i .$$

3.1.3. Сформировать таблицу «Узел-Значение», где «Значение» вычислить по следующей формуле для каждого узла из списка:

$$SLA_i = LA_i \frac{SLA}{100}.$$

3.1.4. Инвертировать столбец «Значение», перейти к пункту 4.

3.2. Сравнить значения параметра *CPU* у всех узлов.

3.2.1. Если значение параметра *CPU* хотя бы у 2 узлов отличается, то перейти к пункту 3.2.2., иначе перейти к пункту 3.3.

3.2.2. Вычислить *SCPU* – сумму значений всех параметров:

$$SCPU = \sum_{i=1}^{N} CPU_i .$$

3.2.3. Сформировать таблицу «Узел-Значение», где «Значение» вычислить по следующей формуле для каждого узла из списка:

$$ACPU_i = CPU_i \frac{SCPU}{100}.$$

3.2.4. Инвертировать столбец «Значение», перейти к пункту 4.

3.3. Сравнить значения параметра *MU* у всех узлов.

3.3.1. Если значение параметра *MU* хотя бы у 2 узлов отличается, то перейти к пункту 3.3.2., иначе составить таблицу «Узел-Значение», в которой каждому узлу поставить 1 в «Значение» и перейти к пункту 4.

3.3.2. Вычислить *SMU* – сумму значений всех параметров:

$$SMU = \sum_{i=1}^{N} MU_i .$$

3.3.3. Сформировать таблицу «Узел-Значение», где «Значение» вычислить по следующей формуле для каждого узла из списка:

$$AMU_i = MU_i \frac{SMU}{100}.$$

3.3.4. Инвертировать столбец «Значение», перейти к пункту 4.

4. Сформировать список приоритетных узлов на основе таблицы «Узел-Значение» следующим образом: обойти таблицу и внести в список один и тот же узел столько раз, сколько указано в столбце «Значение».

5. Первый входящий запрос $R_{i=1}$ передать первому узлу из списка.

6. Входящий запрос R_i передать узлу R_i из списка.

7. При получении запроса R_{N+1} проверить, превышено ли предельное время жизни списка приоритетных узлов. Если время жизни списка было превышено, перейти к пункту 1, иначе присвоить $i = 1$ и перейти к пункту 5.

Заключение

В данной работе был проведен анализ проблемы оптимального распределения нагрузки в гетерогенной программной системе, указаны достоинства и недостатки существующих алгоритмов, предложен алгоритм распределения нагрузки, учитывающий обратные данные о текущем состоянии узлов и меняющий вектор нагрузки в режиме реального времени. Данный алгоритм позволяет эффективно ис-

пользовать имеющиеся аппаратные ресурсы в виде распределенных узлов, равномерная загрузка которых, в свою очередь, ведет к повышению отказоустойчивости системы в целом и уменьшению времени реагирования на поступающие входящие запросы. В будущем данный алгоритм достаточно гибко может подстраиваться под новые метрики и толерантен к включению или выключению учета отдельных метрик в произвольных системах. Это приводит к тому, что данный алгоритм можно применить в высоконагруженных распределенных программных системах, где крайне важно избежать единой точки отказа, обеспечить отказоустойчивость и сбор метрик не является критичным фактором, способным значительно увеличить общую нагрузку на программную систему.

ЛИТЕРАТУРА

1. Павликов М.К. Протокол HDP // Вестник компьютерных и информационных технологий. 2016. № 8. С. 52–56.
2. Richardson L., Amundsen M., Ruby S. RESTful Web APIs. O'Reilly Media, 2013. 404 p.
3. Bourke T. Server Load Balancing. O'Reilly Media, 2011. 194 p.
4. Kopparapu C. Load Balancing Servers, Firewalls, and Caches. Wiley, 2012. 224 p.

Павликов Максим Константинович. E-mail: severemax@yandex.ru
Московский авиационный институт

Поступила в редакцию 6 февраля 2017 г.

Pavlikov Maxim K. (Moscow Aviation Institute, Russian Federation).

Load balancing algorithm in a distributed programming system, built on the basis of the HDP protocol.

Keywords: HDP; distributed software system; load balancing.

DOI: 10.17223/19988605/40/9

In this paper, we discuss load balancing in a software system based on the HDP protocol. Load balancing is a method of distributing requests between several identical nodes of a distributed computing system in order to optimize the use of all nodes together. There are various load balancing algorithms that have different applications in different areas and their purpose is also to ensure the fault tolerance of the system as a whole.

The algorithm of round robin is a cyclic search of all nodes in a circle, each subsequent request is sent to the next node in the set. The main advantage is the simplicity of the algorithm, which makes it possible to use it in many systems and the architecture of such systems remains as simple as possible. The operational characteristics of the system do not add to the complexity of the architecture. From the disadvantages of the algorithm is to distinguish the uneven distribution of the load, if the nodes of the network have different technical characteristics. A technically more powerful node will better cope with requests than others, which will be more loaded. Also, the problem of an inaccessible node is not solved, i.e. which during the operation of the system suddenly went out of order.

The algorithm weighted round robin is a modification of the round robin algorithm, in which each node is manually assigned a certain weight - the probability with which the load balancer next time will choose this particular node. Compared with the round robin in this algorithm, it was possible to compensate for possible technical differences between the nodes of the network. From the disadvantages of the algorithm, it is worth highlighting the lack of a solution to all fault-tolerance problems, because if a network node with a high priority ceases to be available, queries are still sent to it.

The main problem with most balancing algorithms is that they do not use the state information of each node that is in the list from which the balancer selects the appropriate node each time based on a certain algorithm. The feedback node algorithm proposed by the author is an algorithm for load distribution in a distributed system based on the HDP protocol. This algorithm takes into account heterogeneous data about the current load of each node and, based on these data, distributes the load between them. An agent is installed on the monitored node, which transmits to the server the data on the overall load, the processor load, and the use of memory on this node. The HDP balancer uses this data to decide whether to send a request to a particular node, depending on its load at the current time.

REFERENCES

1. Pavlikov, M.K. (2016) HDP protocol. *Vestnik komp'yuternykh i informatsionnykh tekhnologiy – Herald of Computer and Information Technologies.* 8. pp. 52–56. (In Russian).
2. Richardson, L., Amundsen, M. & Ruby, S. (2013) *RESTful Web APIs*. O'Reilly Media. [Online] Available from: https://www.oreilly.com/
3. Bourke, T. (2011) *Server Load Balancing*. O'Reilly Media.
4. Kopparapu, C. (2012) *Load Balancing Servers, Firewalls, and Caches.* Wiley.