УДК 001.817

УЛУЧШЕНИЕ НАДЁЖНОСТИ СИСТЕМЫ ЗА СЧЁТ ДИНАМИЧЕСКОЙ БАЛАНСИРОВКИ НАГРУЗКИ

В. М. Вергасов

Белорусский государственный университет информатики и радиоэлектроники, Минск, Беларусь, vadim.vergasov2003@gmail.com

В работе рассматривается проблема неравномерного распределения нагрузки в распределённых системах с долгоживущими соединениями, которая проявляется при последовательных обновлениях. Для повышения надёжности и предотвращения перегрузок предлагается метод динамической балансировки на основе метрик серверов. В качестве инструмента используется HAProxy, анализируются его механизмы для получения весов в реальном времени. Рассмотрены необходимые доработки бэкенда для интеграции. Реализованный подход позволяет создать саморегулирующуюся систему, которая повышает отказоустойчивость системы.

Ключевые слова: балансировка нагрузки; распределённые системы; динамическая балансировка; haproxy; agent health checks.

IMPROVING SYSTEM RELIABILITY THROUGH DYNAMIC LOAD BALANCING

V. M. Verhasau

Belarusian State University of Informatics and Radioelectronics, Minsk, Belarus, vadim.vergasov2003@gmail.com

This paper examines the problem of uneven load distribution in distributed systems with long-lived connections, a challenge that is exacerbated during rolling updates. To enhance system reliability and prevent overloads, a method of dynamic load balancing based on real-time server metrics is proposed. The article analyzes the use of HAProxy as the primary tool, focusing on its mechanisms for dynamic weight adjustment. Necessary backend modifications for integration are also discussed. This approach enables the creation of a self-regulating architecture, thereby improving the system's overall fault tolerance.

Keywords: load balancing; distributed systems; dynamic load balancing; haproxy; agent health checks.

1. Введение

В современных распределённых системах, особенно тех, что работают в режиме реального времени, надёжность и отказоустойчивость являются ключевыми требованиями [1]. Приложения, использующие протокол WebSocket для долгоживущих соединений, такие как чаты, игровые

сервисы или голосовые помощники, сталкиваются с уникальной проблемой: неравномерное распределение нагрузки. Классические балансировщики отлично справляются с распределением новых подключений при их малой длительности жизни, а вот при длительностях приближающимся к часам или дням – появляется проблема как неравномерность распределения сессий, т.е. нагрузки.

Проблема обостряется во время последовательных обновлений (rolling deployment), когда серверы в одной зоне доступности выводятся из эксплуатации, а их клиенты переподключаются к оставшимся. Это приводит к каскадному накоплению нагрузки на зонах, которые обновляются последними. В системе с тремя зонами доступности последняя зона может получить до 60% всех пользователей, что создаёт риск перегрузки и деградации сервиса. Для решения этой проблемы необходим механизм, способный распределять новые подключения по более рациональному сценарию.

Анализ существующих решений [2–4] в балансировке WebSocket подключений позволили выделить то, что нет готовых решений для гибкой балансировки нагрузки в зависимости от каких-то специфичных метрик. Поэтому целью данного исследования является изучение и реализация балансировки, которая будет опираться на конкретную метрику системы, но при этом позволит взять любой показатель системы как ключевой в вопросе балансировке трафика.

2. Оценка распределения при обновлениях

Для оценки распределения коротко сформулируем, как работает система.

Существует сервис, который работает по WebSocket протоколу. Клиенты подключаются к сервису на продолжительное время. Так же подключение происходит через какой-либо балансировщик нагрузки. Система существует в рамках нескольких зон доступности.

Т.е. это классический сервис, который работает не по HTTP, а через WebSocket. Единственным отличием является то, что клиенты держат подключение долго.

Так же обновления в этой системе происходят последовательно по каждой зоне. Т.е. сначала обновляется зона 1, далее зона 2 и т.д. Это довольно классический способ обновления, так же известный как Rolling deployment.

Мы можем оценить, как будет вырастать нагрузка, для этого решим следующую задачу:

1) пусть всего имеется K зон доступности: $Z_1,\ Z_2, \ldots, Z_K$;

2) пользователь при подключении попадает в любую из них с вероятностью $\frac{1}{K}$ (равновероятно);

3) до обновления: в каждой зоне по $\frac{1}{K}$ пользователей, т.е. система находится в сбалансированном состоянии;

4) обновление (релиз) выкатывается «по одной зоне»: сначала в Z_1, потом Z_2 и т.д.;

5) после обновления зоны Z_i клиенты переподключаются заново.

Необходимо найти долю пользователей в последней зоне доступности перед тем, как она начнёт обновляться. Обозначим за Q_i^j количество клиентов в зоне *i* после обновления зоны *j*. Найти нам нужно Q_K^{K-1}. Так же отметим, что $Q_i^0 = \frac{1}{K}$.

Для этого необходимо составить рекуррентную формулу для Q_i^j.

Попытка решить это рекуррентное соотношение через Maple и упростить это выражение не вышла, поэтому найти выражение Q_i^j, которое бы удовлетворяло отношению (1), вероятно, невозможно.

После обновления первой зоны будет $Q_1^1 = \frac{1}{K^2}$ и $Q_i^1 = \frac{1}{K} + \frac{1}{K^2}$, $i \neq 1$. Далее будет $Q_2^2 = \frac{Q_2^1}{K}$, $Q_i^2 = Q_i^1 + \frac{Q_2^1}{K}$, $i \neq 2$. Выразим в общем виде:

$$Q_i^{i-1} = \frac{1}{K} * Q_{i-1}^{i-2} + \frac{1}{K} * Q_{i-2}^{i-3} + \ldots + \frac{1}{K} * Q_1^0 + Q_i^0. \quad (1)$$

Но можно построить график первых 25 значений Q_i^{i-1}, для этого в системе Maple была реализована рекурсивная функция и был построен график точек, результат можно увидеть на рисунке.

Можно заметить, что количество клиентов в последней зоне будет уменьшаться при увеличении количества зон доступности, т.к. функция имеет вид обратной пропорциональности.

3. Конфигурирование HAProxy для динамической балансировки

Стандартные проверки состояния (health checks) в HAProxy являются бинарными: они могут определить лишь, работает сервер (UP) или нет (DOWN). Хотя этого достаточно для обеспечения базовой отказоустойчивости, но в случае работы в WebSocket подключениями необходимо уметь

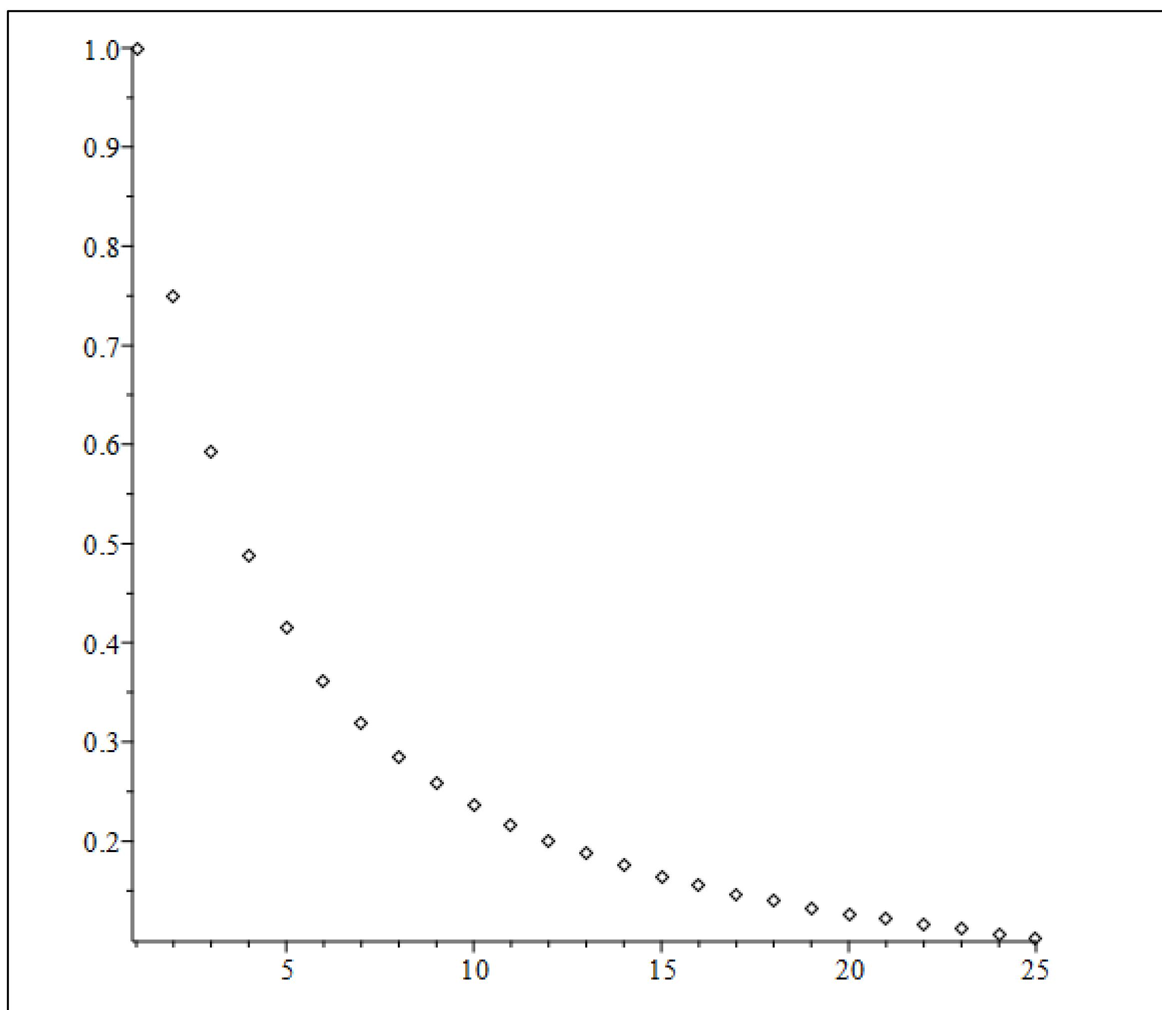

График первых 25 значений Q_i^{i-1}, $i \in [1;25]$

более гибко настраивать правила балансировки. Необходимо адаптировать распределение трафика не только к отказам, но и к изменяющимся условиям работы бэкендов, таким как их текущая загрузка.

Для решения этой задачи HAProxy предлагает два мощных механизма: agent-check и agent-program [5, с. 22–24]. Оба позволяют бэкендам или внешним скриптам динамически управлять своим состоянием в пуле балансировщика, изменяя вес, переводя серверы в режим обслуживания и все это – в реальном времени, без перезагрузки конфигурации, в отличии от базового nginx [6].

Agent-check – это механизм, при котором HAProxy активно опрашивает бэкенд по специальному порту. HAProxy инициирует соединение, а в ответ бэкенд отправляет текстовую команду, описывающую его желаемое состояние.

Принцип работы:

1) настройка в HAProxy: в конфигурации для каждого сервера указывается директива agent-check и agent-port, на котором приложение-бэкенд будет слушать запросы от HAProxy;

2) TCP-соединение: с заданной периодичностью HAProxy устанавливает соединение с каждым бэкендом;

3) ответ от бэкенда: бэкенд-приложение, проанализировав свое внутреннее состояние (загрузку CPU, количество активных задач, доступную память), отправляет в ответ данные с его целевым состоянием;

4) применение изменений: HAProxy анализирует полученную строку и немедленно применяет указанные директивы.

Agent-program – это альтернативный подход, при котором HAProxy не устанавливает сетевое соединение, а запускает локальный внешний скрипт. Этот скрипт выполняет проверку и печатает результат на стандартный вывод, который HAProxy затем считывает и применяет.

При таком подходе выполняются следующие действия.

1. Настройка в HAProxy: вместо agent-check используется директива agent-program с путем к исполняемому скрипту. В global секции необходимо разрешить запуск внешних команд с помощью external-check.

2. Запуск скрипта: HAProxy периодически запускает этот скрипт. Информация о проверяемом сервере (его IP и порт) передается скрипту через переменные окружения.

3. Логика проверки: скрипт использует полученные переменные для выполнения проверки. Обычно это curl-запрос к HTTP-эндпоинту бэкенда (например, /health), который возвращает ответ с данными о нагрузке.

4. Вывод команды: скрипт анализирует ответ, вычисляет необходимое состояние или вес и выводит результат в стандартный вывод. Формат команд абсолютно идентичен формату для agent-check.

В случае балансировки WebSocket подключений можно использовать любой из методов. Но первый довольно прост в реализации (не требуется реализовывать и поддерживать дополнительную логику скрипта) и так же хорошо решает проблему.

4. Доработка бэкенда

Для внедрения по-настоящему динамической балансировки с помощью agent-check необходимо, чтобы бэкенд научился сообщать о своем состоянии. Этот подход требует более глубокой интеграции по сравнению с предоставлением простого HTTP-эндпоинта, но позволяет уменьшить пиковую нагрузку на бекэнды и сгладить эффект от обновлений системы.

Задача бэкенда — запустить HTTP-сервер, который будет получать запросы от HAProxy, анализировать собственную нагрузку и отправлять в ответ команды для управления своим весом в пуле балансировки.

В данном случае вес будет являться функцией, обратно зависящей от количества текущих подключений к бекэнду.

5. Заключение

В ходе выполнения данной работы была успешно решена задача повышения надежности и эффективности распределенной системы путем перехода от статической к динамической балансировке нагрузки. Было продемонстрировано, что традиционные подходы, основанные на фиксированных весах или простых алгоритмах вроде Round Robin, недостаточны для современных высоконагруженных сервисов, подверженных неравномерным и постоянно меняющимся нагрузкам.

В качестве ключевого инструмента для реализации динамической балансировки был выбран HAProxy, благодаря его встроенным возможностям для взаимодействия с бэкендами в реальном времени. Были проанализированы два механизма, предоставляемых HAProxy: agent-check и agent-program.

Так же было уделено внимание необходимым доработкам на стороне бэкенда, которые являются неотъемлемой частью внедрения этих схем.

В результате была спроектирована и реализована архитектура, которая позволяет системе в реальном времени реагировать на изменения нагрузки, динамически перераспределяя трафик на наименее загруженные серверы. Это не только предотвращает перегрузки и сбои, но и оптимизирует использование вычислительных ресурсов, что в конечном итоге повышает общую надежность и отказоустойчивость сервиса.

Переход от статически сконфигурированных систем к динамическим, саморегулирующимся архитектурам является ключевым трендом в современной IT-индустрии. Реализованный подход является не просто техническим решением конкретной задачи, а шагом к построению более устойчивых и экономически эффективной IT-инфраструктуры.

Библиографические ссылки

1. *Ваишнави П., Анандхи С.* Облачные вычисления : Эффективная балансировка нагрузки и выделения ресурсов в облачной среде. Sciencia Scripts, 2023.

2. WebSocket proxying. URL: https://nginx.org/en/docs/http/websocket.html (date of access: 12.08.2025).

3. HAProxy Enterprise Documentation. URL: https://www.haproxy.com/documentation/hapee/latest/onepage/#websockets (date of access: 12.08.2025).

4. Using multiple nodes. URL: https://socket.io/docs/v4/using-multiple-nodes/ (date of access: 14.08.2025).

5. HAProxy as an API Gateway. HAProxy Technologies, 2021.

6. *DeJonghe D.* NGINX Cookbook Advanced Recipes for High-Performance Load Balancing. 3rd ed. O’Reilly Media, 2024.