CPU throttling Go-приложений в Kubernetes

Бикеев Кирилл Алексеевич 25.М71-ММ

Что такое троттлинг и причины возникновения

Requests и Limits

- Вместо фиксированных значений CFS использует относительные веса. Это означает, что requests определяют лишь долю CPU, которую контейнер получит при возникновении конкуренции
- CFS отслеживает, сколько времени потребил каждый поток, в переменной vruntime. Планировщик всегда выбирает для выполнения тот поток, у которого vruntime меньше всего.
- У потоков с большим весом (высоким requests) vruntime растет медленнее, чем у потоков с низким весом. Это заставляет планировщик чаще выбирать их для исполнения, обеспечивая им «справедливую долю» ресурсов.

Requests и Limits

- В отличие от requests, limits всегда жестко ограничены и работают через механизм периодов.
- По умолчанию Kubernetes и ядро Linux используют период планирования, равный 100 мс.
- Лимит пересчитывается в количество микросекунд, которые контейнеру разрешено работать в рамках одного 100-миллисекундного окна. Например, лимит 0.5 CPU дает контейнеру 50 мс рабочего времени на каждые 100 мс реального времени.
- Если контейнер исчерпывает свою квоту до завершения 100-миллисекундного окна, ядро приостанавливает его выполнение до начала следующего периода

Requests и Limits

- Представьте, что CPU — это пирог, который пекут каждые 100 минут. Requests гарантируют, что вам всегда достанется ваш кусок, если за столом много людей. Limits — это правило, запрещающее вам съедать больше определенного веса, даже если остальной пирог никто не ест. Если вы съели свою норму за первые 10 минут, вам придется ждать следующего пирога еще 90 минут, даже если вы очень голодны.

Проблема троттлинга в Go-приложениях

Root cause

- Троттлинг в Go-приложениях происходит из-за несоответствия между внутренним планировщиком Go и механизмами ограничения ресурсов в Linux (CFS Quota). Основная причина кроется в том, как рантайм Go по умолчанию определяет доступную ему мощность процессора.
- Количество CPU доступное рантайму хранится в переменной GOMAXPROCS которая по умолчанию задается как количество CPU доступное всей ноде

Решения

- Библиотека от Uber automaxprox
- начиная с версии Go 1.25 рантайм стал «container-aware», теперь GOMAXPROX выставляется относительно заданного значения limit

Отказ от Limit

Причины отказа

- Деградация Latency: Даже если среднее использование CPU низкое, микро-всплески нагрузки могут вызвать троттлинг, что увеличивает хвостовые задержки (tail latency) в 3–5 раз.
- Риск каскадных сбоев: Троттлинг вызывает образование очередей запросов, что может привести к росту потребления памяти и последующим OOM-kill (завершению процесса по нехватке памяти).
- Экономическая неэффективность: Чтобы избежать троттлинга при использовании лимитов, инженерам приходится завышать их, что требует на 25–45% больше ресурсов, чем при работе без лимитов.
- Избыточность: Механизм CPU Requests (запросов) сам по себе достаточен для гарантии ресурсов, так как планировщик Kubernetes следит, чтобы сумма всех запросов не превышала мощность узла. В случае конкуренции ядро Linux распределяет CPU пропорционально весам (shares), что является более «мягким» и эффективным способом изоляции.

Дальнейший план

- Придумать как внедрить в go механизм настройки GOMAXPROX без использования limit
- Сделать прототип
- Провести тестирование