October 16, 2019

CPU based throttling for a multi-tenancy platform

Anonymous Anonymous

CPU based throttling for a multi-tenancy platform

Abstract

A multi-tenancy system hosts many clients using shared capacity and resources. Client requests (demand) may have different resource consumption and traffic pattern. The multi-tenancy system needs to protect itself from overload. Strategies and techniques for applying CPU based throttling to ensure fair use of resources are described. To ensure a comprehensive and flexible protection, the CPU based throttling is enforced at three levels: a global level, a regional level and a local level.

At the global level, quota-based CPU throttling is applied for each client. A backend system utilizes a global in-memory counter service to aggregate and keep track of a CPU usage (service cost) of the served client requests. A client-side rate limiter controls whether to serve the client requests based on a preconfigured quota and real-time CPU usage of the clients. An adaptive probabilistic throttling algorithm is utilized to maximize throughput while capping the CPU usage under the allocated client quota.

At the regional level, strategies for rate limiting are based on a regional average CPU load. When a region becomes overloaded, the backend system starts to turn down the requests from all clients in that region based on preconfigured client priorities until the capacity frees up again. When a region is under-utilized, soft throttling is applied to make best use of the capacity. The backend system makes best efforts to serve all traffic irrespective of whether clients have exhausted their quotas or not. When none of these situations occur, the quota based probabilistic throttling algorithm is utilized to serve the demand, as done at the global level.

At the local level, server-side throttling is applied as the last line of defense. Each of the hosts rejects requests when the CPU load of the host rises above a preset threshold. It protects individual hosts from overload when in-region load balancing does not work perfectly.

Problem statement

The multi-tenancy system serves a variety of different clients having different resource requirements and traffic patterns. So, it is crucial to handle overloading gracefully to protect availability and reliability of the backend system.

In prior art, a QPS (queries per second) based throttling mechanism has been utilized to handle overloading. The QPS based throttling mechanism sets the throttling limit on an allowed number of requests from each of the clients. But the QPS based throttling mechanism does not directly translate into resource consumption for following reasons:

- Requests from different clients can have vastly different resource requirements
- Cost of serving requests from the same caller can change gradually or rapidly

As a result, there may be scenarios when the number of requests is still within the throttling limit, but the resource consumption is huge (due to a drastic increase in the real-time service cost) and causes system overload. So, applying throttling based on the number of requests per second is inadequate.

The present disclosure proposes a novel solution to overcome the above-mentioned limitations in the prior art.

System and working

The present disclosure describes a comprehensive and flexible CPU based throttling mechanism to prevent overloading and ensuring a fair use of resources in a multi-tenancy system. The multi-tenancy system hosts many clients using shared capacity and resources. Applying rate limiting directly based on resource (e.g., CPU, memory, etc.) consumption is more accurate than the QPS (queries per second) based throttling mechanism. CPU is chosen as the metric because it is more relevant to capacity planning.

To prevent excessive use and to maintain system availability, the CPU based throttling mechanism is enforced at three levels, as explained in subsequent sections.

Following section explains the CPU based throttling at a global level.

(i) Global CPU quota throttling

A backend system processes and measures CPU usages (service costs) of requests (demand). The backend system utilizes a global in-memory low-latency counter service to aggregate and maintain a near real-time client CPU usage of the past second across all hosts on the multi-tenancy system. A cost bumper component updates the counter service with the latest CPU usages of the requests. A global CPU usage limit (CPU time per second) is configured for each client. A CPU load rate limiter applies client-side throttling on the clients based on quota and the real-time service cost of each client. A quota controller is responsible for publishing any changes in client quota configuration (config) in real time.

Figure 1 shows a diagram of major components in global CPU based throttling mechanism.

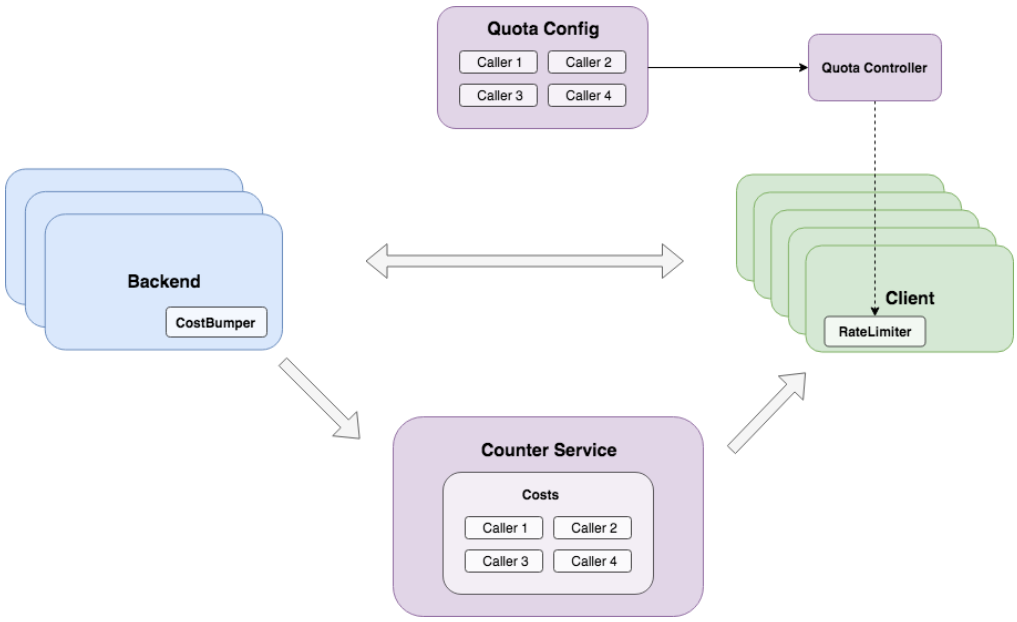

Figure 1: Comprehensive and flexible CPU based throttling mechanism

The global CPU quota throttling provides an accurate resource accounting and auditing system, which is a fundamental feature required for the multi-tenancy system. The requests are throttled only from the clients that are misbehaving, and other clients can still access the multi-tenancy system.

When the demand from the client exceeds the quota, the client-side throttling is applied to cap the CPU usage at the quota level and avoid oscillation. A probabilistic throttling algorithm is utilized to achieve it.

Probabilistic throttling algorithm

The probabilistic throttling algorithm performs proportional control of a probability of dropping requests based on the current CPU usage by the client.

The demand if no throttling was done can be derived this way:

$Usage = Demand * Probability\ of\ Serve$

$Demand = Usage\ /\ Probability\ of\ Serve$

$Demand = Usage\ /\ (1 - Probability\ of\ Drop)$

Probability of dropping request at time t:

$P(t) = Max\ (0,\ 1 - quota\ /\ demand)$

$$P(t) = Max(0, 1 - \frac{quota \times (1 - P(t-1))}{usage})$$

$usage$ = the CPU usage by the client in the current time interval

$P(t-1)$ = the probability of dropping requests in the previous time window

Under normal conditions, the client CPU usage is less than the quota and no request would be dropped. When the client CPU usage exceeds the quota, the client starts dropping requests. The probability of dropping request would increase as attempts of the request grow.

This algorithm works very well in practice. When the demand exceeds the quota, this algorithm caps the CPU usage at the quota level while maintaining a stable request serving rate.

Figure 2 clearly shows that the probabilistic throttling algorithm, as explained above caps the CPU usage of an offending client at the quota level.

Figure 2: The usage capped at the quota using the adjustments and the iterations of the probabilistic throttling algorithm

Following section explains the CPU based throttling at a regional level.

(ii) Regional CPU load throttling

An upper limit and a lower limit of a regional average CPU load is configured for each region. The upper limit is a safe net to prevent a regional meltdown. Once the upper limit is hit, the requests are throttled based on client priority irrespective of whether they have exhausted their quotas or not, as shown in Figure 3. However, this kind of extreme situation should not occur if cross-region load balancing works

perfectly. When a region becomes very busy, the requests can be distributed to other under-utilized regions, thus the upper limit is hit very rarely.

To make best use of the resources, a lower limit is set, which is a key to soft throttling. If the value of the regional average CPU load falls below the lower limit, the global CPU quota throttling is disabled, and best efforts are put forth to serve the requests from all the clients, as shown in the Figure 3. The requests from all the clients are served irrespective of whether the clients have run out of the quotas or not. This mechanism is called soft throttling.

However, if the value of the regional average CPU load lies between the upper limit and the lower limit, the global CPU quota throttling is utilized to determine whether the requests can be served, as shown in the Figure 3.

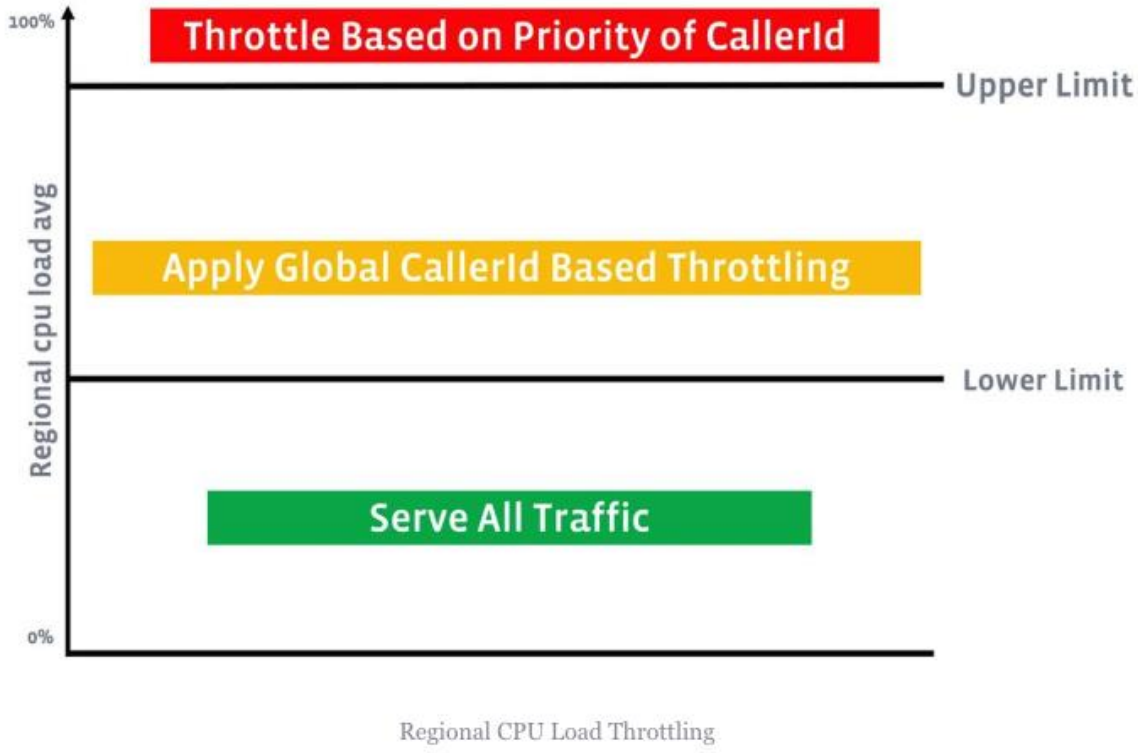

Figure 3: The regional CPU load throttling

Figure 4(a) clearly shows that the demand gets served immediately after the soft throttling is enabled. The system makes best efforts to serve as much traffic as possible. However, as the regional average CPU load climbs up and exceeds the lower limit, the global CPU quota throttling gets enabled and it results in throttling errors, as plotted in Figure 4(b), corresponding to the clients for which the demand exceeds their quotas.

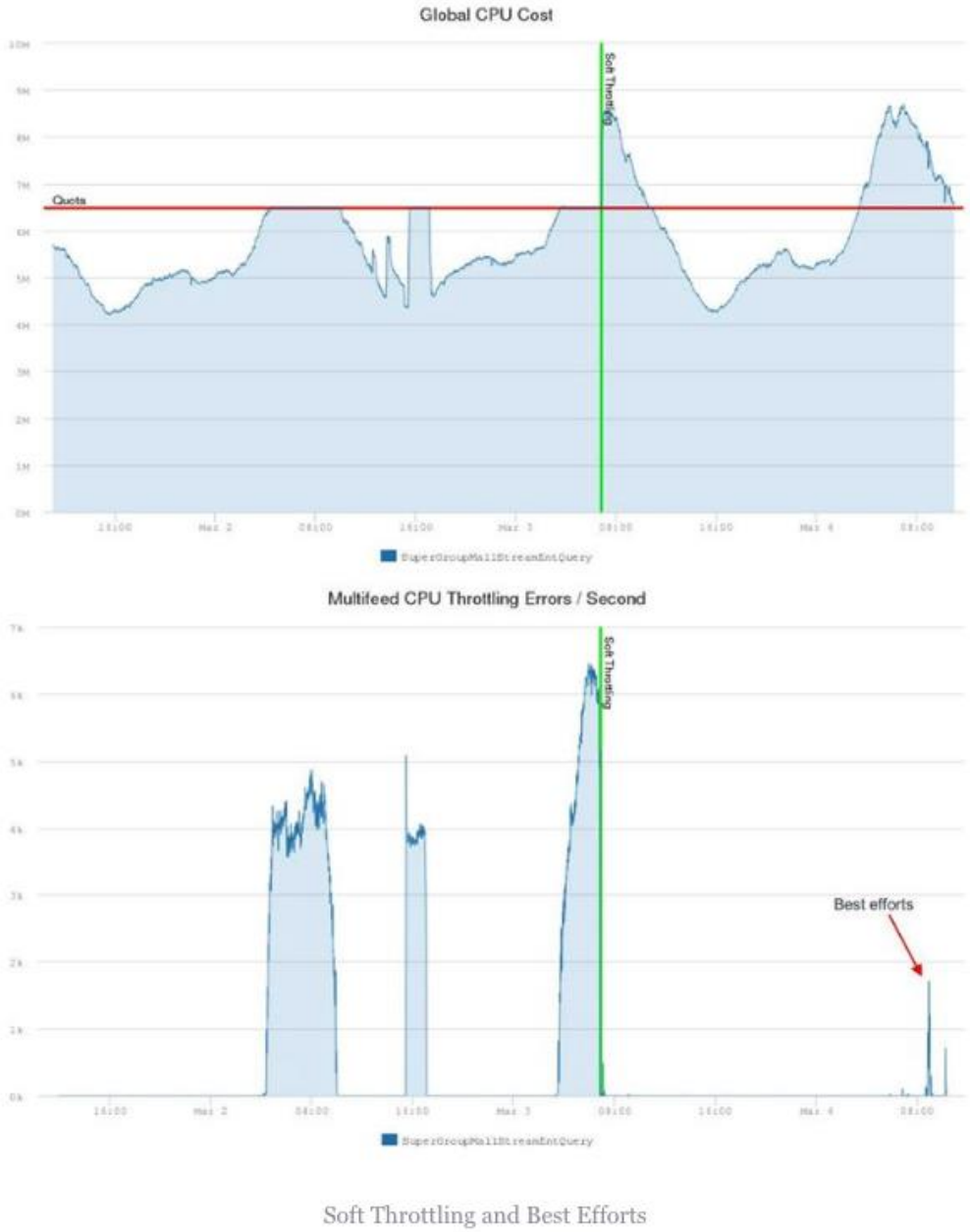

Figure 4(a): Capping and serving the demand in the regional CPU load throttling

Figure 4(b): Throttling errors/second

Following section explains the CPU based throttling at a local level.

(iii) Local CPU load throttling

Even within the region, CPU load varies among the hosts. Figure 5 shows a plot of CPU load variance over a period of time in the region. Here the difference between the P90 of the CPU load of the hosts

and the average CPU load of the hosts is used to show the CPU load variance. As a last line of defense, each of the hosts throttles the requests when the CPU load rises above a preset threshold. Once this happens, retries of the dropped requests can be routed to a spare host based on a customized retry logic (which is implemented in a client library, not described here). This reduces crashes caused by overloading and improve in-region load balancing to some extent.

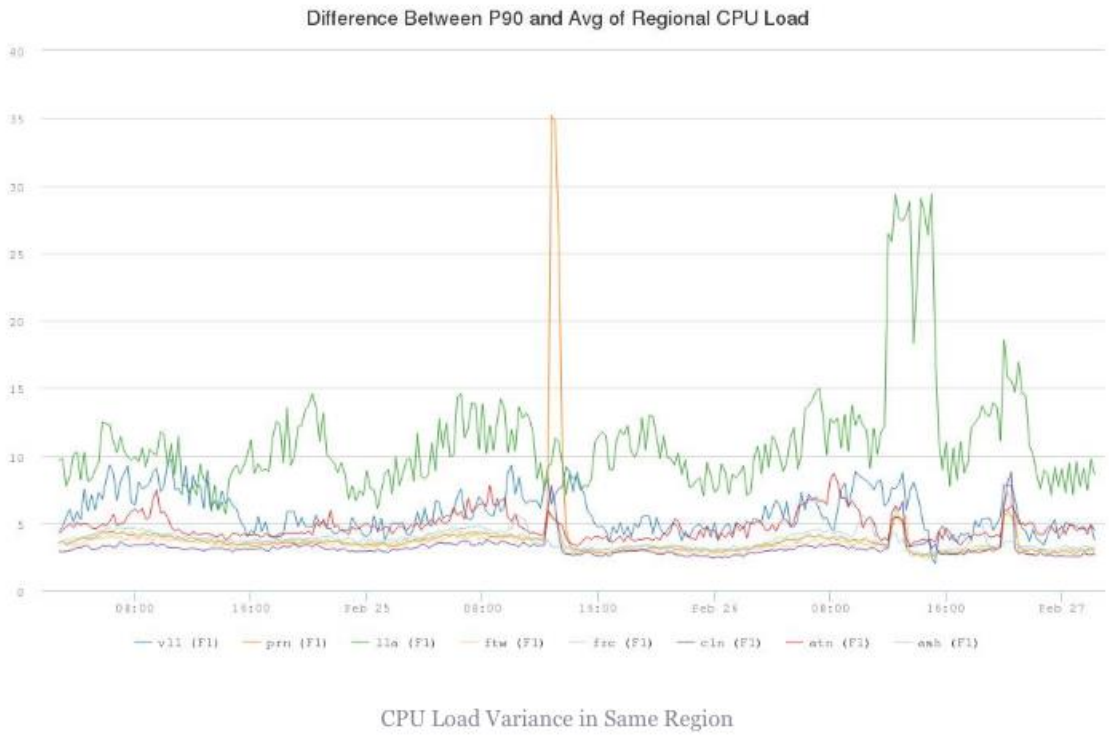

Figure 5: The CPU load variance in the region

Additional embodiments

In an additional embodiment, an improvement may be done in the probabilistic throttling algorithm by using a configurable query window (for example, 2 seconds) for a previous average throttling rate. This is needed because traffic pattern of some of the clients may change very rapidly whereas the traffic pattern from other clients may be stable. Using longer query window stabilizes the average throttling rate and provides a more accurate estimation of the demand.

Conclusion

With the advent of virtualization, infrastructure sharing, and data centers operated by third parties, multi-tenant architecture has been gaining popularity. A large number of clients can be served efficiently at the same time with shared capacity. The key to leverage the benefits of the multi-tenant architecture lies in sharing resources among different tenants in a reliable manner. The present disclosure introduces a

comprehensive and flexible CPU based throttling mechanism for keeping multi-tenant platform available and robust. This is achieved by handling overloading gracefully.