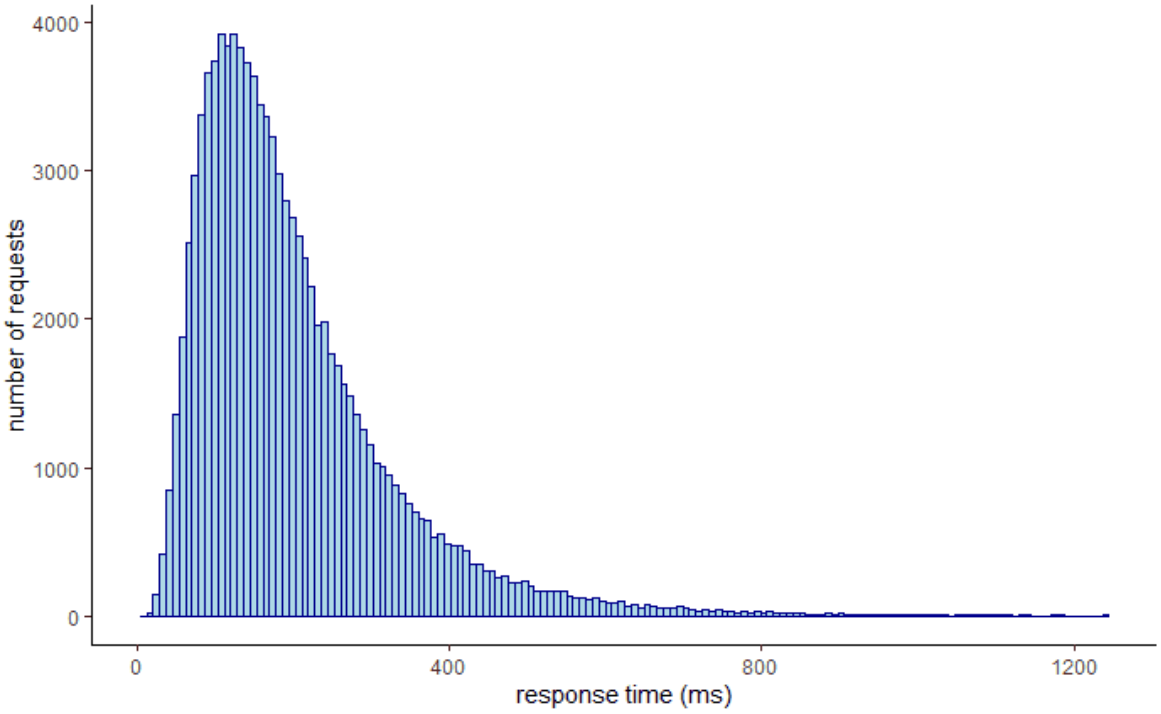
Why you should measure tail latencies

May 31, 2020

The response times of your service can cost you dearly if left unchecked. Even when a small fraction of requests experiences extreme latencies, it tends to affect your most profitable users, and not in a good way. On top of that, long response times can decrease the resilience of your service, making it more costly to operate. But, I am getting ahead of myself - to begin with, let's start by defining what response time is.

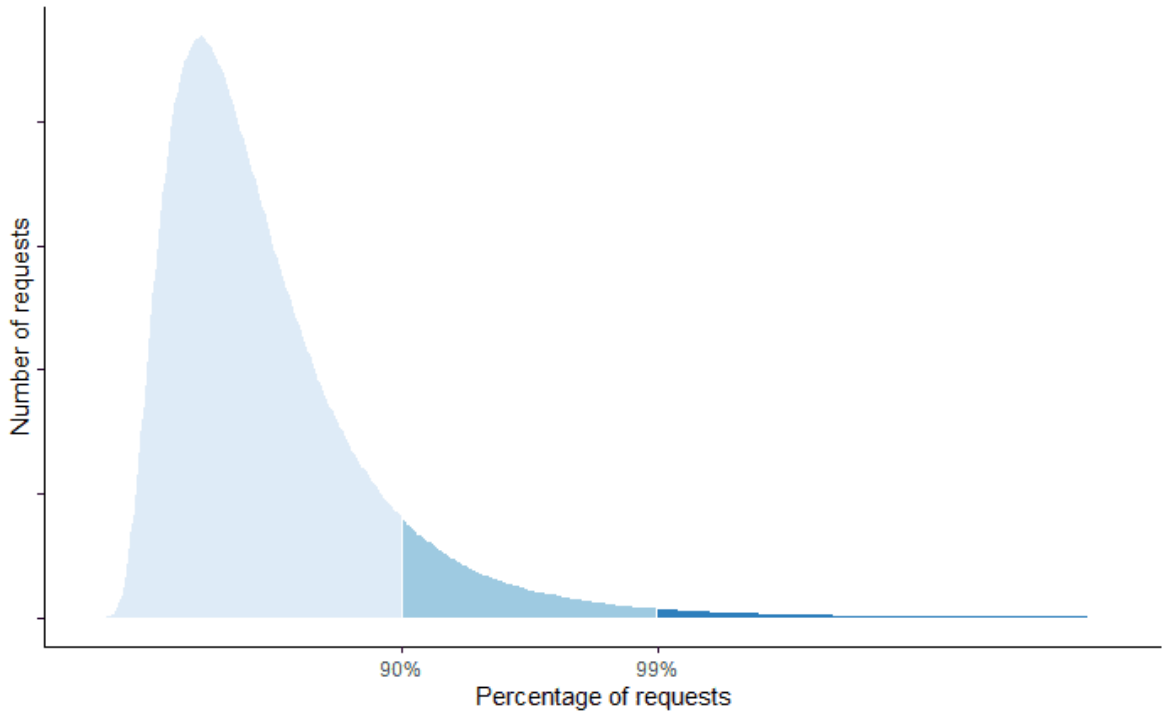
The time your service takes to respond to a request can vary quite a bit based on several factors. The presence of network timeouts, page faults, or heavy context switching can all affect it. As not every request takes the same amount of time, response times are best represented with a distribution. Typically, this distribution is right-skewed and long-tailed.



The distribution of response times is usually log-normal

You could use the average, or the arithmetic mean, to summarize the distribution with a single value. But how representative is the average really? While the average has its use, it's not helpful if you want to know the proportion of your users experiencing a particular response time. All it takes is one crazy outlier to skew the average. For example, if 100 users are hitting your service with a single request, 99 of which have a response time of 1 second and one of 10 min, the average is nearly 7 seconds. Even though 99% of the users experience a response time of 1 second, the average is 7 times higher than that!

A better way to represent the distribution of response times is with percentiles. A percentile is the value below which a percentage of the response times fall. For example, if the 99th percentile is 1 second, then 99 % of requests have a response time below 1 second. The upper percentiles of a response time distribution, like the 99th and 99.9th percentiles, are also called tail latencies.



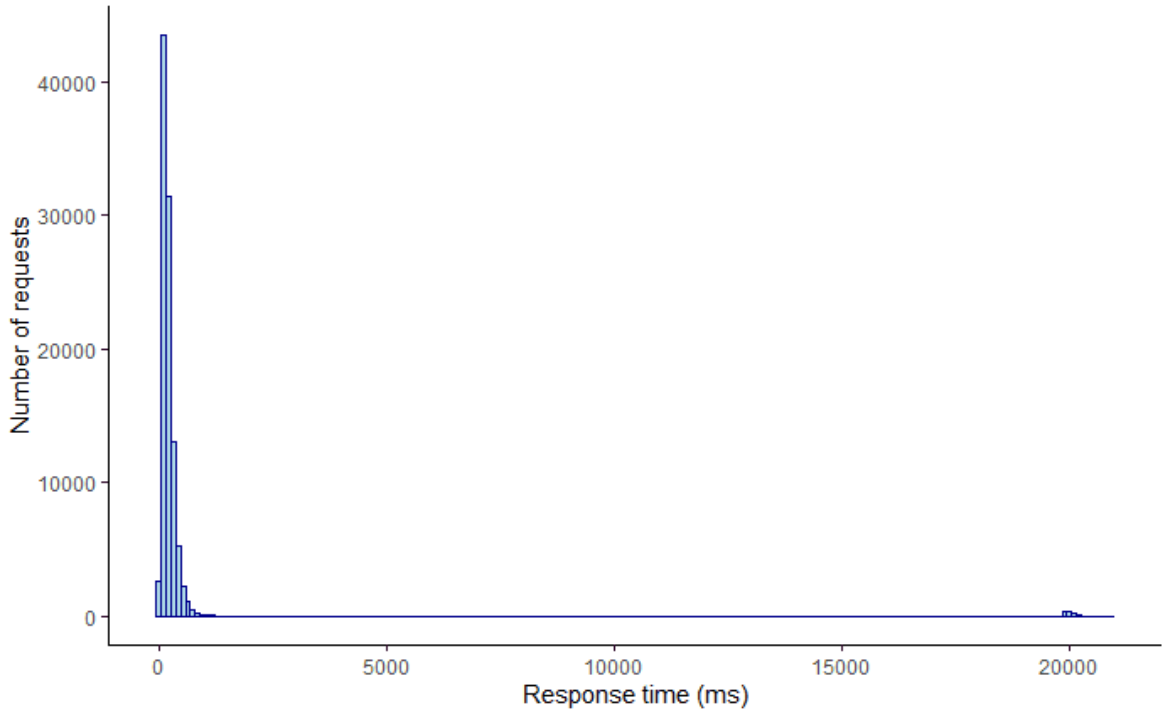
The 90th and 99th percentiles in all their glory

Even though a small fraction of requests experiences these extreme latencies, it tends to affect your most profitable users. These users tend to be the ones that make the highest number of requests and thus have a higher chance of experiencing tail latencies. Several studies have proven that high latencies affect revenues. A mere 100-

millisecond delay in load time can hurt conversion rates by 7 percent.

To avoid losing users, you have to keep upper percentile response times in check. One way to do that is to enforce a service-level objective (SLO). A service level objective sets a target level of reliability for a given metric. For example, your SLO could state that 99% of requests over a rolling time window of a week should be serviced within 1 second. If your service receives 1M requests in a week, then at most 10K requests can be over 1 second; these 10K requests are also referred to as the error budget.

As it turns out, keeping tail latencies under control doesn't just make your users happy, but also significantly improves your service's resilience while reducing operational costs. Why? Because if you are forced to guard against the worst-case response times, you happen to improve the average case as well. Tail latencies that are left unchecked can quickly bring a service to its knees. Let's say your service is using 2K threads to serve 10K requests per second. By Little's Law, the average response time of a thread is 200 ms. Suddenly, a network switch becomes congested, and as it happens, 1% of requests are being served from a node behind that switch. That 1% of requests, or 100 requests per second out of the 10K, starts taking 20 seconds to complete.

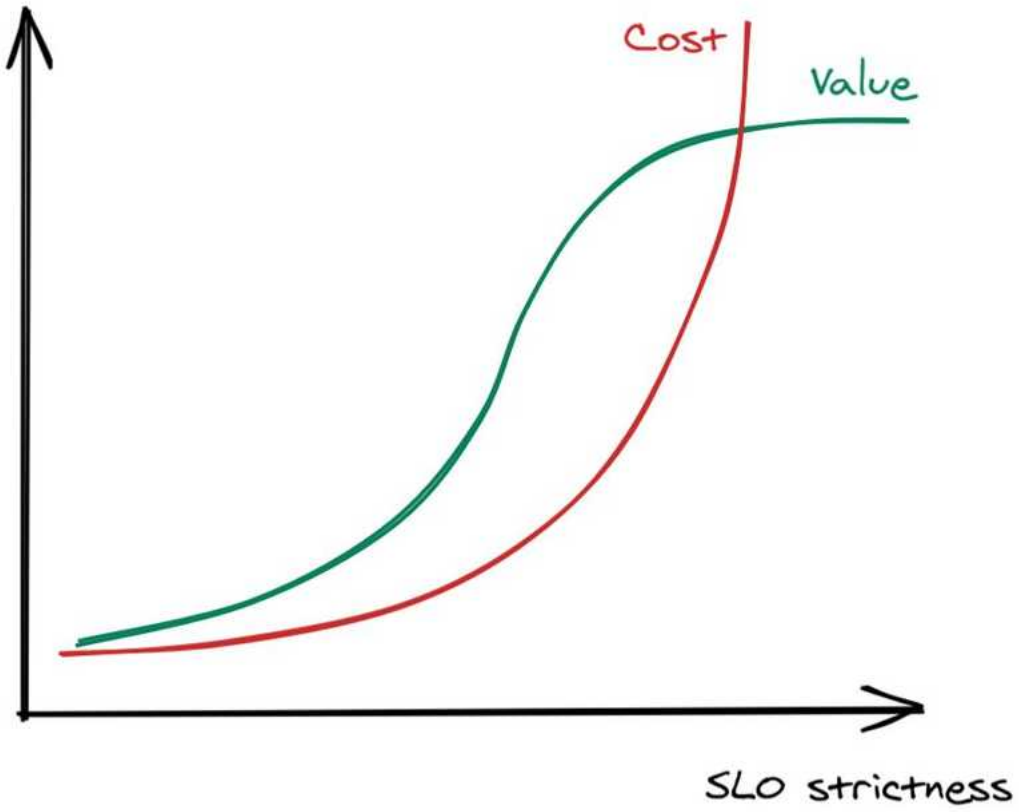


1 % of requests are taking much longer to process

How many more threads does your service need to deal with the small fraction of requests having a high response time? Well, if 100 requests per second take 20 seconds to process, then 2K additional threads are needed to deal just with the slow requests. So the number of threads of your service needs to double to keep up with the load!

If the service level objective is based on tail latencies, then to guarantee it, your service needs to have self-healing mechanisms in place. There is no other way around it. In our previous example, failing fast or adding a health endpoint behind a load balancer might do the trick. Over time, these mechanisms reduce your system's costs just as much as the operational burden to run it.

How strict should a SLOs be though? There typically comes the point when making the error budget tighter becomes very expensive in terms of engineering time, and it doesn't make much sense to go after it as the cost to benefit ratio is too high to justify the investment.



SLO trade-off between costs and benefits

The sweet spot depends on your service, but in general, anything above 3 nines is tough to achieve and provides diminishing returns.

Remember this

High latencies impact your revenues and costs if left unchecked. You can't fix what you don't measure - a good first step in the right direction is to enforce SLOs based on percentiles of response times. To guarantee the SLOs, you are going to be forced to build self-healing mechanisms to guard against the worst-case tail latencies. As a nice side effect, these mechanisms are going to improve the average response time as well and make your service less costly to operate.