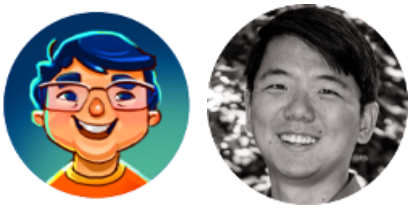
Professor Beekums Blog



Follow Professor Beekums

2017-08-19

Performance Vs Scalability

One thing that tripped me up early on in my career was the difference between performance and scalability. At first I thought they were exactly the same. I was quite surprised when my first project to scale a system actually made my code run slower… in my dev environment at least.

Let's get definitions out of the way. Scalability is being able to handle large amounts of users/data/traffic. Performance is about speed. While the speed of light may be constant, software system speed is far from.

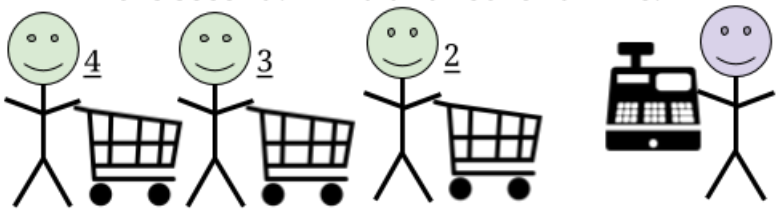
A useful analogy is checking out at the supermarket. Let's say a cashier can checkout one shopper every minute. Assuming no more than one shopper goes to the cashier every minute, then a shopper can checkout in one minute.

What happens when 2 shoppers attempt to go to the cashier every minute? One shopper will be checked out in 1 minute,

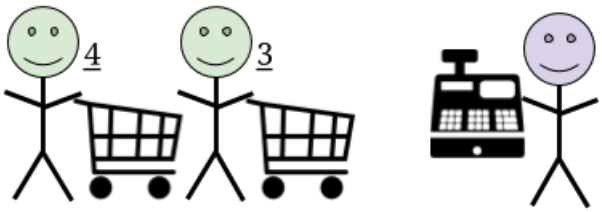
but the second shopper will take 2 minutes.

A third and fourth shopper will come while the second shopper starts checking out. The third shopper will have to wait 1 minute for the second shopper to finish and then spend 1 minute checking out so the third shopper will also take 2 minutes. But the fourth shopper needs to wait 3 minutes.

12:01 First shopper done. Start checking out the second. Third and fourth arrive.

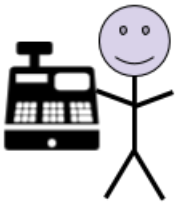


12:02 Second shopper done.



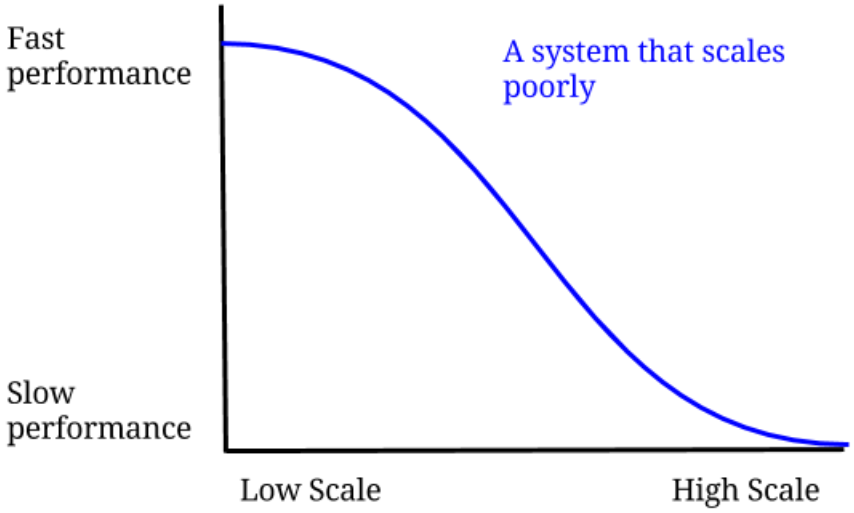
12:03 Third shopper done.

12:04 All shoppers done.

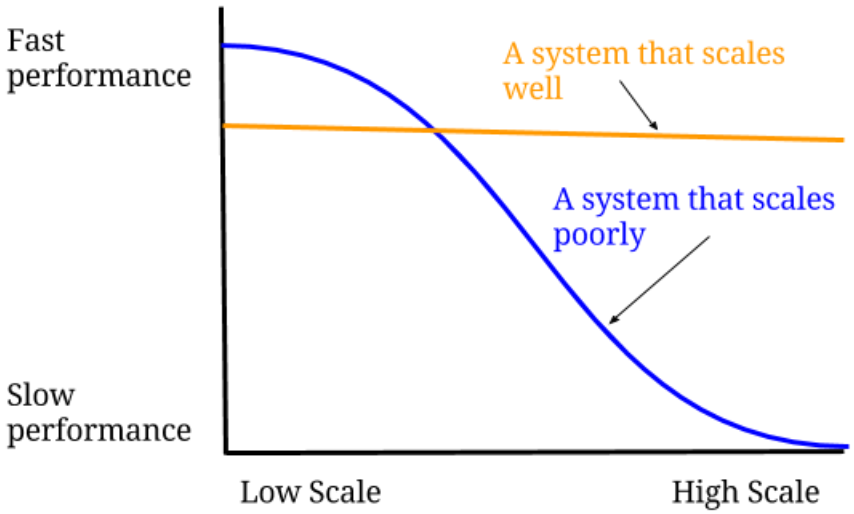


The fifth and sixth shopper are also arriving just as the third shopper starts. Fifth shopper waits 3 minutes, sixth shopper waits 4 minutes. The longer this keeps up, the longer shoppers will have to wait.

The performance we care about is not how fast the cashier can checkout a single person. It is how long the average shopper is waiting. Performance for a single cashier works for us well enough when there aren't very many people. The performance drops quickly when we add shoppers at a faster rate than a single cashier can handle.



Scalability is making sure that the performance is constant no matter now many people are coming in. This may mean creating things that lower performance when there aren't very many people (low scale). Ideally our business is successful and we're more concerned about performance with a greater number of people (high scale).



One tactic used is to add caching. In some cases you may need to process a lot of data for a user. But if that data needs to be processed exactly the same many times, then it may be worth just saving the final result.

Using our supermarket example: let's say there are a number of regulars who order the same thing every day (they're hungry people). Instead of having them scanning the same items day after day, we could give them a special receipt. Scanning this one receipt will automatically scan their entire shopping cart. Instead of taking a minute to

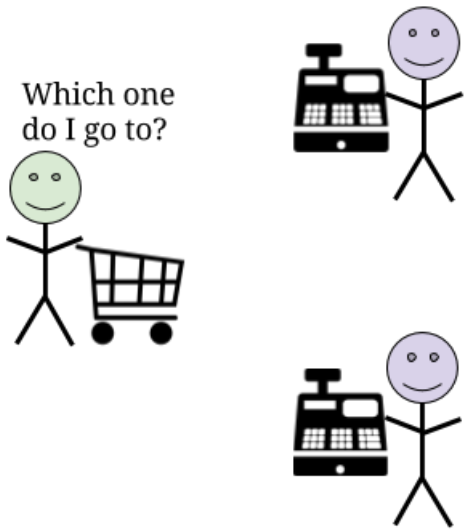
check them out, our cashier may only need 10 seconds to check the receipt. That increases the number of people a single cashier can handle to 6 a minute instead of 1 a minute.

However, the receipt may fail for a number of reasons:

- The shopper is not consistent with what they buy (data that changes often)
- The supermarket gets a lot of first time shoppers (large variety of data)
- The receipt data is lost (cache data loss)

In these situations we have “cache misses” where our receipt doesn’t work. That means we have to spend time checking the receipt and then scanning our shopper’s shopping cart anyway. Instead of 10 seconds or a minute, our cashier now spends 1 minute and 10 seconds to check out a shopper.

The next thing we could do is add more cashiers. If 1 cashier can handle 1 person a minute, then 6 cashiers can handle 6 people a minute. The performance for a single shopper may still go down because they need to decide which cashier to go to. They may spend 5 seconds making that decision so their checkout is 1 minute and 5 seconds instead of 1 minute.



However, we don’t care too much about reducing the performance slightly when there are only a small number of shoppers. We care about making sure we don’t greatly

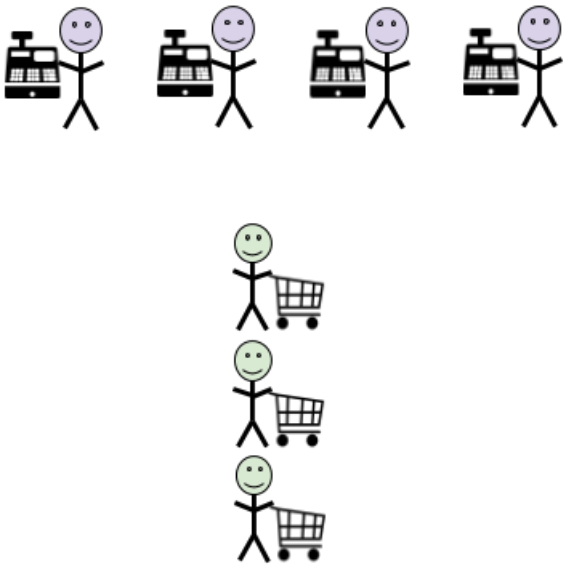
reduce performance when there are a large number of shoppers so we will add more cashiers.

We may still encounter issues with lots of cashiers though. There's always the chance we will always have more shoppers than we expect and they will overload our cashiers. People will start lining up behind each cashier.

What if there is one shopper who is taking a really long time? The shoppers behind that person will be frustrated watching other lines move faster. What if there isn't enough space for the lines behind each cashier? Do the lines start merging? Do we have shoppers get frustrated when it appears that someone cuts in front of them? Do fights start breaking out at our supermarket?

What happens if one of our cashiers falls ill and there is no replacement? Does every shopper in line now have to get on the back of another line? Or will they just leave without buying anything?

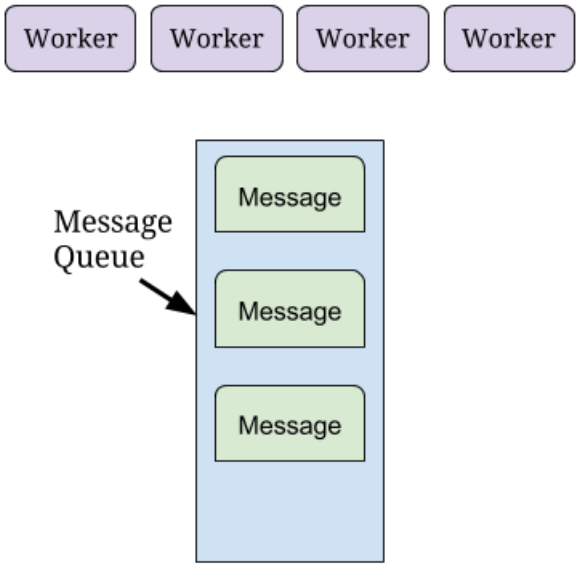
We can solve this by creating a single line or queue for every shopper. Now every shopper will start checking out in the order they enter the queue. They will also not have to worry about being stuck behind a slow shopper or a cashier falling ill. With a single queue, shoppers just go to the next available cashier.



It may take longer. Our shopper has to find their way to the end of the queue. They may have to walk farther from the

queue's exit to the next available cashier. But this is a small price to pay for an orderly and speedy checkout process.

The software analogy would be message queues. A message represents our shopper and the things they have to buy. Messages are added to a message queue just like shoppers go to the back of the line. Software processes called workers take messages from this queue and process them much like cashiers checkout shoppers.



There are many other techniques to improve scalability. I hope this has illustrated what some of those techniques look like. Also the result will be more complicated code that runs slower on a developer machine so it may seem like a pointless effort at first. The effort will pay off though when you have a successful high scale business.

Hi there! I hope you enjoyed this post.

If you did, I'd appreciate it if you took the time to check out my product: **Dynomantle**

Dynomantle will help you deal with thousands of bookmarks, notes littered everywhere, lost important emails, newsletter subscriptions, and more! If you suffer from information overload, give Dynomantle a try.

Signup for free!