# Impact of Latency
# On Database Performance

A White Paper From


For more information, see our web site at
**http://www.goldstarsoftware.com**

# Impact of Latency on Database Performance

**Last Updated: 02/15/2021**

One of the most common questions we get is related to application performance. In other words, users want to know why their system is running so slowly, and (if possible) how to make it run faster. This sounds like a valid question, right? Well, it's actually quite complicated, and we need to break it down into a number of factors.

Let's start with a simple definition – what is "performance"? Most people look at what I would call *perceived application performance*. This can be determined by monitoring the time it takes from when a user issues a command (for example, via a mouse click) until the time that a response is displayed on the screen. In most cases, a stopwatch is the best monitoring tool, but if you're unlucky, you might need a clock (or even a calendar).

But, what actually goes into this total time needed for a process? In reality, this is split into the sum of the time it takes to complete every step of the task. Let's look at the various steps with respect to a database application that is reading ONE record from the database. This process then includes:

1. Queueing of the mouse-click event in the local OS
2. Receiving of the event by the application
3. Application think time needed to process the event locally and decide that a record is needed from the database
4. Transmission time to send the request across the network to the server
5. Server-side processing time, or the time for the server to locate the database record requested
6. Transmission time to send the reply back to the workstation
7. Application processing time to receive the data back and deal with it as needed
8. Screen display time – i.e. building the graphical display of the response.

Whew. Of course, most applications don't request just one record – many will request several hundred records for each "user requested process", so we may repeat steps 3-7 several hundred times, therefore we must add the time on each and every one of these requests together to get the total *perceived application performance* time.

So, when a system is perceived by the user to be "slow", how can we speed it up? There are a few common solutions often considered:

- Get a Faster Workstation
- Get a Faster Server
- Get a Faster Network

Let's look at these one at a time.

## *Get a Faster Workstation*

This one seems like a no-brainer. My desktop computer running the application, so it must be the slow piece. Let's upgrade it! You now spend $1500 for the latest box with more memory and hard disk space than you need – but it still doesn't help.

While getting a faster computer or one with a faster video card will help with items 1, 2, 3, 7, and 8, most of these gains will only shave nanoseconds off of a process that already takes microseconds (us). An exception to this would be the case of an application that is doing data analysis or reporting calculations, in which case it may really benefit more from a faster CPU.

## *Get a Faster Server*

After the new workstations don't help, you may realize that the database server is running their database engine itself, so any slowness must be the fault of the server. Again, buying bigger, faster, better hardware is easy to justify with very little research, and replacing the workstations didn't help, so replacing the server seems the next obvious step. Since any server-centric database application environment **will** be limited in performance by the throughput of the server, getting a faster server (or more cores, or more memory) should be a benefit, right?

There's actually two common misconceptions here.

- **CPU Performance**: The first mistake is looking at benchmarks when determining CPU performance. Many of the common benchmarks are running multi-threaded applications to simulate workload. These respond really well when you have a CPU with 32 cores all running in parallel. However, that 32-core CPU may be actually running at a clock speed of only 2.0GHz. While multi-core CPU's excel at doing multiple things in parallel, your one report process that is taking 35 minutes to complete will only be using one core at a time, and then running at only 2GHz. This is when you find out that your old server with only 2 cores running at 3.0GHz was actually around 50% faster than the new box!
- **Server Cost**: The second mistake is assuming that a high-end server is required. Buying a box with 64 cores running at 3.2GHz, 512GB of RAM, and really fast SSD's in a RAID-10 array is going to be quite expensive. However, that extra money doesn't translate into any extra speed for your 10-person network with a 3GB database.

Let's go back to the factors that are included in performance. Note that any gain due to a new server is limited because this impacts only step 5 above. Let's say that you buy a faster server, and that you have been able to decrease the server-side processing time from 60us to 40us – a 33% gain. That's wonderful! Now all of your reports run 33% faster, right?

Nope. Assuming a report has to read 100,000 database records to complete. With each record now being processed 20us faster, the total gain after spending all of the money on the new hardware (and possibly software) is actually 20us * 100,000 = 2 seconds. Ugh.

And that's when the frustration sets in. After spending all this money for new desktops and a really fast server, users are still complaining about their performance!

## *Get a Faster Network*

Most people give up after the first two and just blame the application at that point. To some extent, it is the application design that is the problem, so this is technically true. However, the application developers say that on their own servers, these reports (using your test data) take only 30 seconds, so any problem must be in your environment.

What is the real difference between your environment and the developer's environment? The developer is likely using a single, fast test server for running your workload – and everything is in local memory. They are not running across the network at all!

If you look carefully, at the list of factors impacting performance, steps 4 and 6 involve data transmission over the network. Together, these two factors comprise the *round trip time* (RTT) of your network. This is the total amount of time it takes for the request to leave the workstation, traverse the network, and arrive at the server, and then for the reply to leave the server, traverse the network, and arrive back at the workstation.

Using a tool like Wireshark, it is possible to actually capture the network traffic with accurate timestamps. If you grab a network capture from the workstation and the server together, then you can directly compute the RTT by looking at the total time from the workstation's perspective for a given request to get a reply, and then subtract out the total time for that same request to get a reply from the server (which is the server's processing time). So, if a workstation sends a request to the server and gets back a reply in 240us, and you find that the server's processing time on the request was 40us, then you can determine that the round trip network time was 240 – 40 = 200us. (If you've read all the text above and are savvy at math, you might now realize that losing 200us in the network pales by comparison to the 20us you saved when you bought that new server.)

Now, what about a slower network connection – one with a 400us RTT? What about a Wifi connection with an extremely variable connection, with some RTTs clocking in at 1000us or longer. Let's extend that further to your wide area network, where your round trip time may measure 24 milliseconds (ms), or 24000us, or even higher. As you can see, the 20us gains on the server are lost 1000-fold with each time you have to send a request across the network.

## *Why Can't I Get a Faster Network Connection?*

The next logical question is whether a faster network will help. However, in the classic sense of the word, what we consider "network speed" is, in reality, network **bandwidth**. This is actually a measure of network **capacity**, since we are limited by physics when trying to speed up the electrons that transmit the signal in the wiring.

Let me explain it another way: Let's say that you are out of oranges at home and need to go to the supermarket. You get in your car, start it up, drive 2 miles, go inside, pick out a bag of 12 oranges, check out, drive home, shut off the car, and go back inside. In this case, the total time of your journey may be 10 minutes, and your bandwidth (the amount of oranges you can carry) is 12.

Now, let's look at bandwidth -- how many oranges can you actually carry. Assuming you have a decent size car, let's assume that your back seat can hold 6 cases of 100 oranges each, or 600 oranges in total. Your round trip time hasn't changed much. You might argue that it'll take *slightly* longer to get the oranges from the store to the car, but if you have help from store employees (moving orange boxes in parallel), it won't add *that* much extra time. Your round trip time may now be 10.5 minutes. However, you just bought 600 oranges, which is your

maximum bandwidth for the car. If you need to buy 6000 oranges, you'll need 10 round trips to the store, each taking 10.5 minutes, or 105 minutes in total. Or, you could spend extra money on more bandwidth by renting a truck to go to the store – where you can then pack all 6000 oranges in a single load, reducing your total processing time considerably.

However, nobody needs 6000 oranges at a time. What if you want to have just ONE orange for breakfast, so you go to the store and buy just the ONE you want to eat this morning? Note that your *available bandwidth* in the car may be 600 oranges, but your actual payload is only 1 orange. Overall, it still takes 10 minutes round trip time (latency) to get your orange in the morning. But, after you eat, you decide you are still hungry and go back to the store for another orange. There's another 10 minutes lost in latency. However, when you get home, you find that the orange is rotten inside, you have to go back to get another one -- another 10 minutes. Note how each round trip is quite costly overall due to the small payload. The *latency* of going to the store is high, so it took you 30 minutes to eat breakfast – even though you hold the world's record for fastest orange peeling and eating each orange is really quick.

Can you fix this problem by buying more bandwidth? If you rented a truck that can carry 6000 oranges at a time and then went to buy just one, would it reduce your time lost in transit? Nope -- not at all. Buying bandwidth is buying capacity, and capacity is rarely a limiting factor in database performance where many small requests are being made.

The right solution, of course, is to adjust the payload to match your accordingly. You change your application – er, breakfast – design to buy a "reasonable" number of oranges (12) per trip. You don't need to spend the money on a rental truck to get the oranges you need, as your capacity (600) still outmatches the payload (12). Granted, sometimes you will have a week where you don't eat all 12. Some oranges will go bad and need to be thrown out, but in general, grabbing 12 at a time just makes a lot more sense.

## *Making Orange Juice*

So how does all this fruit really apply to database applications? Most database applications need to retrieve many different records from the server. However, the *next* record the application needs is often dependent on the *previous* records that have been read. For example, you may read an Invoice record, and then you need to read the Customer Name for that Invoice. Later, you need to read all 12 Line Item records for that invoice. You don't know which records you need until you read the one before it, so the program has to go back & forth quite a bit.

When you look more closely at this application design, you realize that the latency (round trip time, RTT, or whatever we choose to call it) is going to be your biggest factor here. Unlike reading a Word document where you can make a request for 1MB of data and have it stream over to you (taking advantage of bandwidth), you can only request one record at a time, and then wait for it to come back before making the next request.

We could buy more bandwidth, but remember that bandwidth is NOT link *speed* -- it is link *capacity*. The round trip time is the time it takes for the packet to go from the workstation to the server. In a WAN situation, this could include the time it takes to leave the workstation,

travel the local LAN wire, go through all local switches, pass the local firewall, get encrypted for a VPN channel, get routed to the Internet, travel across the uplink to the ISP, be routed by the ISP to the target network (which may involve many "hops", each involving additional router delays and data transmission delays caused by distance), be passed down the local line at the target location, get routed into the VPN device, get decrypted, get placed on the local LAN segment, get passed through any local segments and switches, and get to the server. Then, the server can find the data and sent it back via the reverse route (which is just as long). I should also note that if you are using Wifi on either side, this will add even more latency. (Pro tip: Stick with hard-wired connections only.)

## *Factors Impacting Network Latency*

What really goes into this latency time? Each router hop will add latency, so reducing hops is usually the first/best solution. If you do a TRACERT to the target server and you see 12 hops, then you have to realize that you are adding latency for **each** hop (which may be dependent on load on each device, in addition to bandwidth between devices). If you are going through three hops just to get out of your own network, then reducing that is easy. Reducing hops on your ISP's network is usually much harder. You also have no control over how busy the routers are (which can add more time) or how congested the links are (even at their utility-scale bandwidth). However, you could pay for a direct fiber optic line to be run from one location to the other, which will cut your hop count by quite a bit -- assuming you can afford the monthly charges.

Switches, while being much faster than routers, also add latency, even if they are “cut-through” switches. Even an enterprise-class switch like a Cisco Catalyst 3850 has a minimum latency of 5 microseconds, and the cheap switch you might pick up from a local big box store may be considerably higher. Check your switch specs (from the vendor) to get a feeling for how much time you can expect to lose in each switch.

Next up is total distance. People don't think about cable length being important, but it actually is. The electron wave (i.e. the physical manifestation of your data) travels down a copper cable at approximately 2/3rd the speed of light. How fast is a fiber cable? Also 2/3rd the speed of light, because light is slowed down by the glass and internal reflections, too. While this is still really fast (186000 miles/second), it is also finite and real. If your total physical cable length is 186 miles, then you are losing 1ms in propagation delay alone. Note that a dedicated fiber line is also a way around this -- if you can select a shorter route, then you can reduce propagation delay accordingly.

Of course, if you don’t believe me, the proof is in the math. Let’s even take a real-world example. I had a company upgrade their environment to brand new servers in a brand new data center, with a dedicated fiber optic line between the two sites running at 10 gigabit “speed”. Of course, user applications got slower. When pressed for details, it turned out that the new data center in a protected building some 10 miles away, there were all sorts of VPNs and VLANs in use, and the total network round trip time went from 0.2ms for the local users to around 7ms for users running remote to the data center. Although 7ms sounds fast, you have to understand that 7ms is *35 times slower* than what they had before!

## *Solving the Latency Problem*

Unfortunately, the limitations of the physical world restrict us from eliminating much of the latency in the typical connection. Until someone perfects ansible communications through quantum entanglement, the speed of light will always be a limiting factor. However, here are a few more realistic things to consider:

1. **Pay for more bandwidth**. Again, this will ONLY help if the link is already saturated, but it's where most people start, unfortunately. Don't make this mistake unless you have the data to prove that link capacity is the issue!
2. **Pay for a dedicated fiber link**. Many ISP's will offer a direct site-to-site link to you, though the monthly cost may cause you to lose consciousness for a short time.
3. **Reduce Network Devices.** If you are dealing with a local network only, then you'll have control over the number of switches, routers, media converters, and other devices that look at each packet. Reducing the number of devices between the client and the server will definitely help.
4. **Reduce Network Distance.** Again, under local control this is easy – use the shortest possible cables, and locate the users physically close to the servers to minimize latency caused by distance. (Often you can reduce network devices AND distance at the same time.)
5. **Leverage a Cache.** Some database environments, like Zen/PSQL, support a client-side cache that can store a small amount of data within client memory and eliminate the need to go across the wire for *every* packet. Even saving 25% of your network round trips can result in a notable performance gain. Unfortunately, not all applications are compatible with the Client Cache Engine (CCE), so this may or may not be available to you.
6. **Redesign the Application to Optimize Data Reads.** If you are in control of the application, you can redesign it to make more efficient use of the database. For example, instead of read a list of invoices one at a time for a report, you can read 100 invoices at a time with a single, larger request. This can save 99% of your round trip times! For Btrieve-based applications, look at Extended Operations. Or, use SQL queries with a large client-side buffer and client-side cursors to retrieve the data very rapidly.
7. **Redesign the Application to Eliminate Data Reads.** This is similar to the above, but even better. Let's say you are reading 10,000 invoices for a report. For each invoice, you need to also read the Customer Name. Of those 10,000 invoices, though, they are only being sent to about 1500 unique customers. As you read the first one, you save the Customer ID along with the customer name that comes back from the subsequent lookup. The next time you need to look up a customer, you can quickly check the list of ID's and names that you've already read and find it, eliminating the need to go back across the wire yet again. Yes, this takes CPU time and memory, but you did just replace your workstations with really fast boxes, right?
8. **Change Where You Run the Application.** Instead of running the application on the workstations, set up a computer or terminal server directly in the data center, connected to the same core switch as the database server, and run your applications remotely from that machine. Now, the only things crossing the network are keystrokes and mouse events from the user, and screenshots streaming back from the terminal

server. All of the latency is limited (because the client is close to the server), and the delays on the screenshots are minimal.

## *Side-Bar for Physics Majors: Wait -- What About Wifi?*

In the discussion above, we noted that the propagation speed of the electron wave in a copper cable was about 2/3rd the speed of light. Further, the transmission of light in a fiber optic cable is about the same – 2/3rd the speed of light. Wouldn't WiFi communications actually be FASTER, since the radio waves propagate at the speed of light through the atmosphere?

Indeed, this is true! However, it doesn't help. The radio spectrum is a *shared* medium, and WiFi adapters can't simply blurt out what they want to transmit any time they want. They must either first announce their intentions with a request-to-send (RTS) and obtain a clear-to-send (CTS) back before transmitting, or they have to somehow detect when a collision occurs or when a packet is lost due to multiple devices communicating at the same time. Due to the algorithms involved in this, along with the overhead in the wireless-to-wired media conversion, WiFi ends up FAR slower than a wired connection.

If you still have questions, contact Goldstar Software and let us work with you to help!