MOLECULA

Breaking the Latency Floor—

Data Access at the Speed of Thought™

MOLECULA

Abstract

Latency, not unlike death and taxes, is a certainty of life that modern man has yet to avoid. Computer scientists and software engineers across industries and technologies devote countless resources to reducing latency—the time delay between an action and a response—in nearly every imaginable application.

Latency-based technological limitations are universally endured, but also present an opportunity for transformative improvement. Managing latency is vital to mission-critical applications such as power grid management, medical records, and financial transactions. It is equally important to business-critical applications such as **retail purchasing**, **online gaming**, communications, **mobile experiences**, **entertainment**, and literally any technology that relies on data access and transfer.

With lives and profits at stake, solutions for reducing latency come in many forms and are applied at all levels from the physical layer up to the application. This paper analyzes the most common causes of latency as they impact processing of analytical workloads, and it discusses several of the most frequently implemented solutions. As data demands grow universally, it is more important than ever to address the "false" *latency floor* that is preventing engineers from even approaching the true, physical *latency limit*—the point at which it is impossible to reduce lag time due to raw physical limitations.

Read on to learn about Molecula's novel approach to data access which breaks through the latency floor created by the zoo of data processing abstractions so common today. Molecula's technology reduces complexity and compresses *days, hours, or minutes* worth of processing time into ***milliseconds***.

Table of Contents

Latency: An Overview

Latency, in brief, is the time delay between an action and a response. For example, the average person experiences latency every time they click or tap a website link and wait for the requested page to begin rendering on screen. If the page takes a long time to load, it may be due to high network latency, but it is also likely due to constrained throughput. It is worthwhile to understand the relationship between these two concepts. *Latency* is response time, whereas *throughput* is how much of something you can get per unit of time. They are both important concepts, and data scientists and engineers often have to consider tradeoffs between them when solving the challenge of accessing and delivering large amounts of data in a short amount of time.

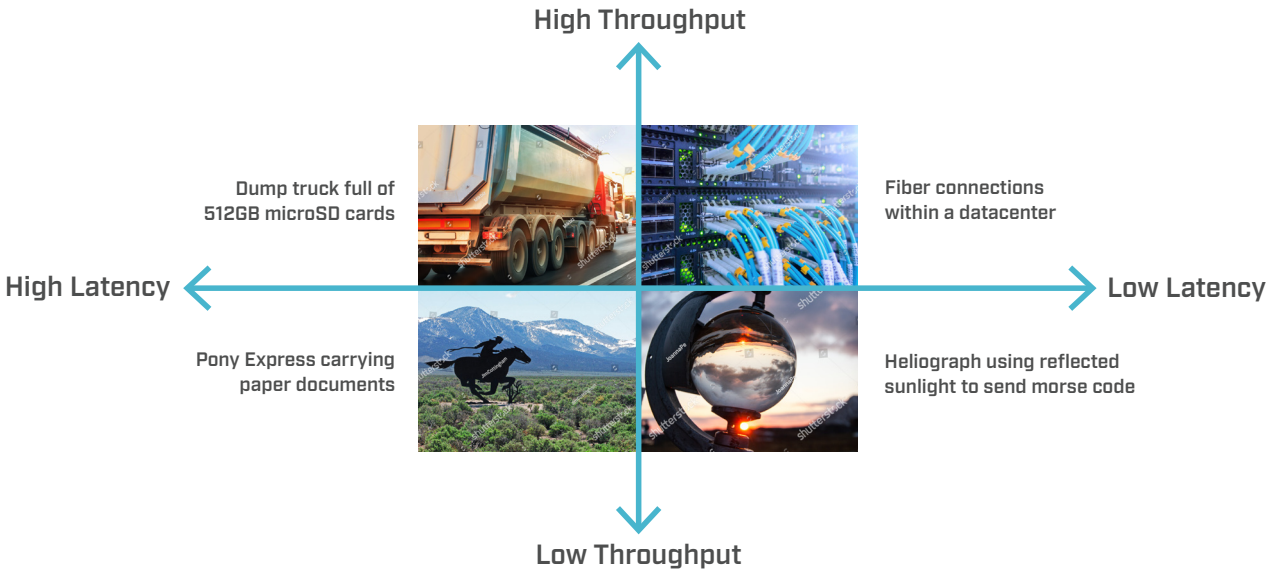


Figure 1. Throughput vs. Latency

Figure 1 above demonstrates combinations along the continuums of latency and throughput with tangible examples. While throughput can nearly always be increased (adding more cables, more dumptrucks, more ponies, etc.), latency has always had a hard floor; dump trucks and ponies can only go so fast. The "latency limit" refers to the point at which it is impossible to reduce task time due to raw physical limitations.

The most fundamental limit to latency is the speed of light. A web page hosted in New York will never be served to a browser in San Francisco in less than about 28 milliseconds. They're about 2,500 miles apart, the speed of light is roughly 670 million miles per hour, and so the "light distance" between them is 14 milliseconds. Since the request must go out and the response be returned, the total time is 28ms—also known as the round trip time or RTT.

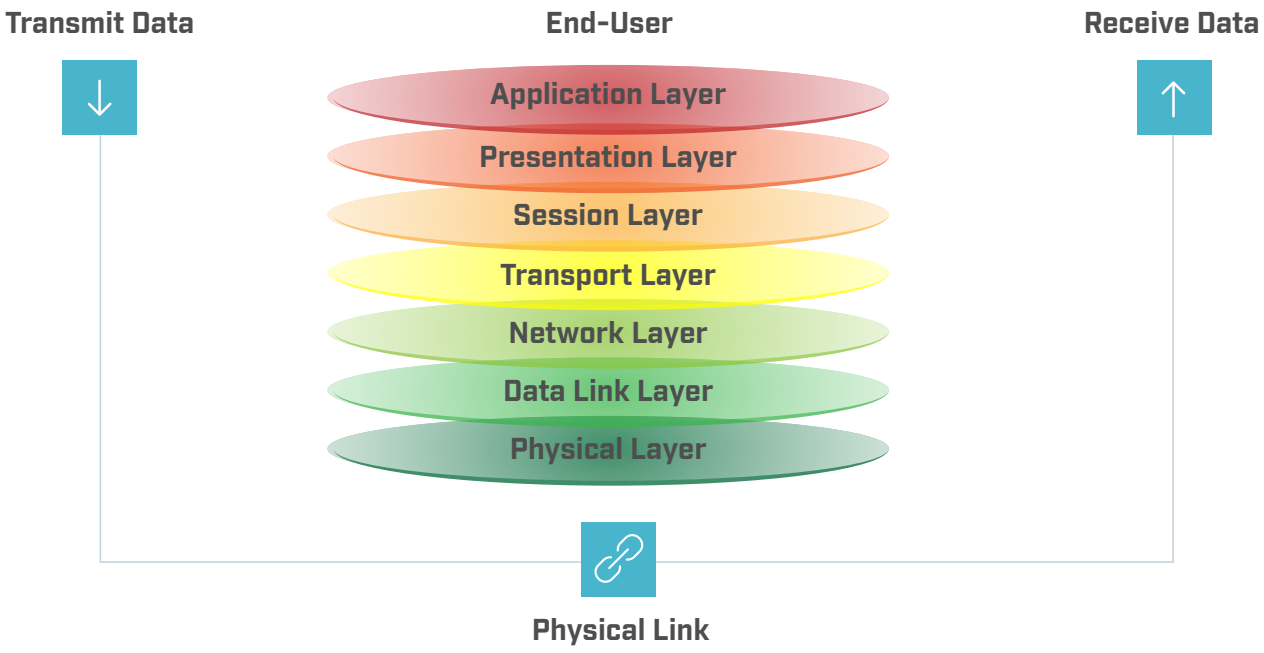
In practice, the RTT will be even larger due a variety of factors such as:

1. Delays caused by routers and other networking equipment processing the packetized information and any processing which must be done at the endpoints such as simply serializing the information and sending over the network interface.
2. The path the information traverses must weave through physical cables connecting various routers, so it is actually a longer distance than the straight line distance between any two locations.
3. The speed of light within a transmission medium is less than the speed of light in a vacuum. For example, optical fiber and copper result in roughly 30% lower speed.

Even after taking the above into account, the actual latency of serving a web page is usually significantly larger than the full RTT because a Transmission Control Protocol (TCP)—and probably a Transport Layer Security (TLS)—connection must both be established. This can require multiple round trips to execute the various handshakes involved at the protocol level.

Layers of Latency

In addition to the fundamental physical causes of latency, there are obstacles to faster response times at every other layer of the network. Modern networks universally utilize the Open Systems Interconnection model (OSI model) seven layer approach where each layer builds new abstractions upon the last, and each has different responsibilities. The typical layer stack includes: physical, data link, network, transport, session, presentation, and application layers.



While there are many reasons this layered approach has been so universally adopted, each and every layer of abstraction incurs some cost and contributes to latency. One case in point is TCP and TLS, mentioned briefly above, which operate at the transport and session layers respectively. Among other things, TCP enables reliable, in-order delivery of data while TLS provides security by encrypting traffic. Both protocols incur latency costs in the form of extra processing at the endpoints, extra data for headers, and, most impactfully, additional round trips across the network.

All this being said, the most grievous offender in terms of added latency is often not in the network. Many times, processing at the endpoint of a request overwhelms other sources of latency to an almost comical degree. This is particularly true in the case of analytical data processing where queries routinely take hours or even days.

Who Feels the Greatest Latency Pain?

A growing number of users find themselves needing access to data that is so large, so rapidly changing, and so complex that it's difficult or impossible to feasibly utilize. When latency is an issue for all the reasons previously discussed, imagine how the problem is compounded by massive, exponentially-growing datasets. Analysis of large datasets, whether for fraud detection, marketing strategies, business intelligence, scientific research, risk calculation, or any number of other applications is limited not by human intelligence nor potential for incredible benefit, but by a struggle for affordable, real-time data access.

Traditional relational databases are infamous for taking hours or even days to process a single query of a large dataset. In addition to being frustrating and expensive, by the time query results come in, the data is often out of date.

> The time and resources it takes to perform actions based on query results such as a seemingly-simple follow-on query can make the payoff not worth the expense—if it's even technically possible at all.

Researchers, marketers, data scientists, business analysts, and AI are all made markedly more effective by reducing data access latency.

Estimating the Latency Floor

Latency Limit vs. Latency Floor

Before diving into measuring latency with respect to analytical data access, it is helpful to think of latency in two parts. The first part is the latency that is dictated by physics—the speed of light and the distance separating two communicating entities will apply equally to all systems. We'll call this the latency limit. The second part of latency is that which is inherent to a particular system, but not bound by the laws of physics.

The latency *floor* of a system is the absolute best latency you can expect to achieve when you've fully explored all of the parameters of the system.

This is all very abstract, so let's walk through an example.

An Estimation Example

In order to discuss the latency floor, we must first carefully define a system—which parts are fixed, and which parts are the parameters? A system might be defined as running a particular query on a particular data set in Elasticsearch v7.6, running on c4.8xlarge instances on AWS, with a particular version of the JVM with particular settings, etc. In this case, maybe the only parameter of the system being adjusted is the number of servers it's using. This parameter can be scaled up while the latency is observed until the optimal value is discovered. At some point, adding more servers won't improve latency, resulting in the *latency floor* for this system. The definition of the "system" could then be relaxed to allow tuning of JVM settings or the Elasticsearch version, and ultimately the whole parameter space can be explored (in theory) to find the latency floor. As long as the physical distance between the client querying and the ES cluster serving the query remains largely the same, the *latency limit* won't really change.

For a narrowly defined system it's easy to determine the latency floor, but in practice the systems we're interested in are much less constrained. If you work for a large company that's looking to start a new data analytics project, your parameter space could be huge. Which cloud vendor will you choose? Will you use a managed service or deploy a traditional database? If you deploy it yourself, what instance types will you choose? One method of estimating the latency floor for a broadly defined system would be to apply reasoning from the basic capabilities of the system's components. This would give a lower bound on what the latency floor could be. For example, you

might reasonably assume that your system will be composed of servers which are connected by a network which has a certain amount of throughput and average latency between nodes. Each server could have a CPU which runs at a particular clock frequency and can process a certain number of instructions per cycle. Each server has some number of memory channels, each of which support a certain data rate.

Your data set will have a certain size, and you can make some assumptions about how much of it your queries will have to scan on average. If we assume perfect sharding, the query will have to be fanned out to every node, so we can figure out the latency cost of doing this and getting the results back. We can reason about how much of the data set actually needs to be read to process the query, and taking the aggregate memory (or disk) bandwidth across the cluster, reason about how long it will take to do that. We can further consider how much processing needs to be done on the data which is read, particularly if there are O(n*log(n)) operations like sorting, or quadratic operations, and get an estimate of how long this will take based on ops per clock and number of processors available. If the result set is expected to be large, we can use the network throughput to estimate how long it will take to deliver back to the client.

This type of analysis can deliver a very optimistic lower bound on the latency floor, but it still has very little to do with the latency limit. The laws of physics are not limiting latency at this point, as it is being driven by the assumptions we're making about what hardware we have access to and, more importantly, how the data has to be processed to serve the queries.

It's pretty important to have some understanding of the latency floor for a system which you are evaluating. These days, many systems scale well, but scalability is usually talking about a throughput *ceiling*.

The latency *floor* directly limits what kinds of use cases you can tackle. BI, edge computing, artificial intelligence, and ML applications all have specific data access requirements. For example, you can't drive a friendly user interface with a system where the latency floor is measured in seconds. You can't power a self-driving car with a system where the latency floor is in the 100s of milliseconds. You can't go to space on a system where the latency floor is in the 10s of milliseconds.

Latency-Reduction Strategies

Given the analysis above, there are a number of ways that we could go about trying to reduce the latency floor and open up new use cases.

Reduce Physical Friction

As previously discussed, physical distance connected by physical wires bears inherent friction. What if the wires were eliminated? It would be relatively cheap to blanket the majority of Earth's population with high-bandwidth internet access using just a few geostationary satellites. Satellite internet services are in fact used in rural areas and other conventionally inaccessible locations. While satellite access eliminates physical wires, the altitude of geostationary orbit (over 22,000 miles) means that the absolute floor for any communication is nearly half a second due to the fact that a single round trip between two Earth-based entities must go up to the satellite and back down twice.

Only recently have space launches become affordable enough to allow us to consider large constellations of low-Earth orbit satellites to enable low latency satellite internet. These schemes are far more complex and require thousands of satellites for full coverage since the motion of the satellites over Earth's surface is quite fast and the amount of the surface that any one satellite can "see" is greatly decreased.

The below diagram shows the limited visibility of low Earth orbit vs. geostationary and the relative distances involved.

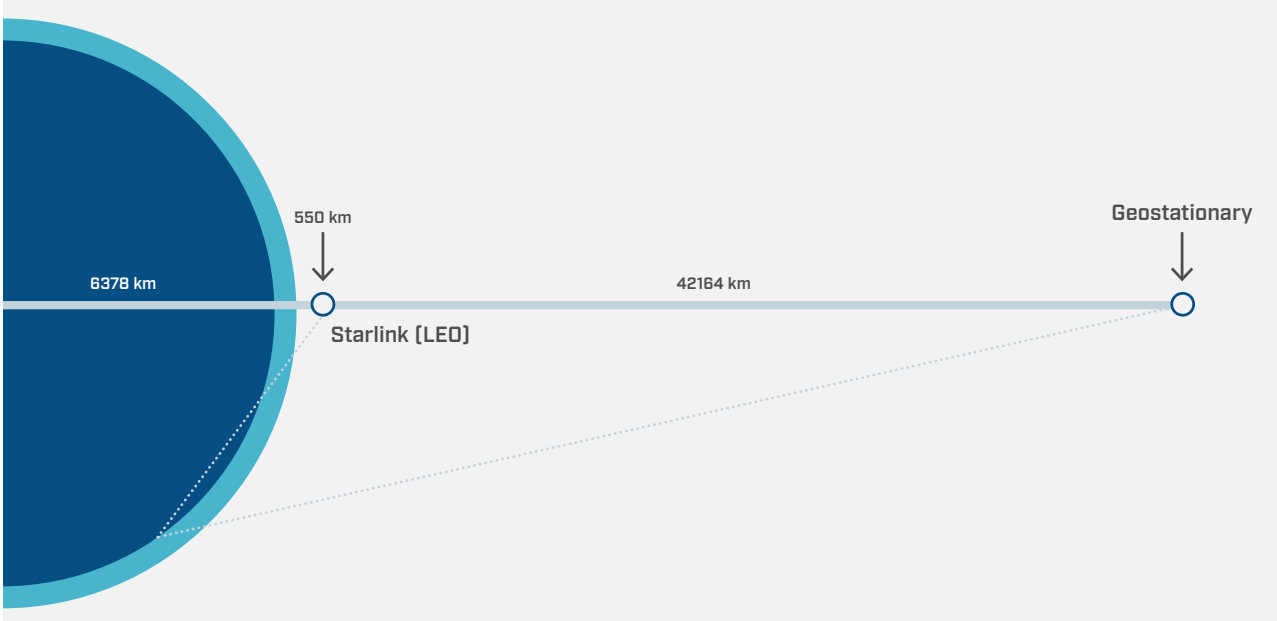


Figure 3. Line-of-sight earth surface visibility for low-earth vs. Geostationary orbits.

While decreasing the physical latency limit would be helpful, in the realm of analytics and data processing, it is a relatively minor gain. It could represent an improvement of a few dozen milliseconds to communicate with the other side of the planet, but this is negligible if your query is taking an hour. You'll feel that kind of improvement a lot more if your starting point is in the hundreds of milliseconds, but we'll have to look at other strategies to get there.

Scale Up

The "scaling up" approach refers to buying a bigger machine to house the database. While buying bigger machines definitely improves latency to a point, most demanding applications will hit that point sooner if not later. One machine won't support more than about 100 cores and a few terabytes of memory. Even if the required data set fits in memory, the amount of I/O and processing which needs to be done to serve a complex query may still take hours. For example, scaling from a machine with one core to a machine with 100 cores would result in a 100x performance increase in the absolute best case scenario.

The 24-hour query would be reduced down to 15 minutes. While that's a big improvement, it is neither sufficient nor acceptable to most end users.

Scale Out

If the problem can't be solved with a bigger machine, another solution would be to spread the workload over many machines. This "scaling out" approach works pretty well. As the data is spread over more and more machines, each machine only needs to process a smaller chunk of data. All these machines can save time since they work in parallel. However, there is overhead associated with fanning a request out to many hundreds or thousands of machines, and there is overhead on each of those machines in processing the request, returning its results, and eventually those results need to be aggregated into a single answer.

Now we return back to our fundamental limits. A thousand machines don't fit into a small space; there is necessarily distance between them, not to mention networking equipment. For large numbers of machines, fanning out a query and reducing the results may involve several network hops. Additionally, the more machines that are involved, the greater the chance that some will have failures or performance hiccups adding to overall request latency. If one machine fails to return results, that portion of the query must be reprocessed. Sometimes it is necessary to speculatively execute a query in multiple places to mitigate failure, but this compounds the problem by requiring the provisioning of even more hardware.

Scaling out can nearly always provide more throughput, but the effect on latency, even when the bulk of the latency is due to data processing, is a bit more subtle. Every time more servers are added to the processing of a request, the latency *limit* gets raised, not lowered, and depending on how much processing there is to do, the latency *floor* will start to increase as well.

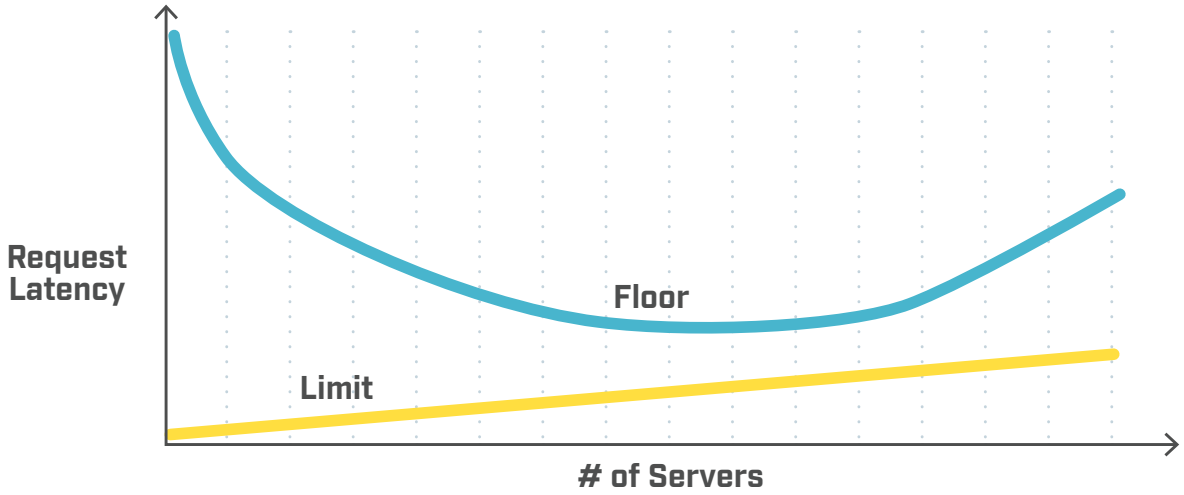


Figure 4. When scaling out, the latency limit consistently increases, while the latency floor decreases up to a point, but ultimately trends upward.

It's worth noting that these solutions are not either/or. Scaling out, for example, will always be a part of the solution when it comes to big data. However, more can—and needs—to be done to drive down latency.

Pre-Process

The next often-used strategy is pre-processing the data. This includes techniques such as data marts and OLAP cubes. When data is pre-processed, it can be queried and explored very quickly as long as the specific needs have been articulated and are supported by the processed version of the data set.

Pre-processing typically involves aggregating data. The data set is shrunk to a more manageable size, but the trade-off is a loss of data resolution so granular views are not accessible. Technically, the latency is still there, it is just moved to a new location within the process. The typical life cycle begins with a business unit or data science department making a request to IT for some data set that is queryable in a certain way. IT builds a processing pipeline to get the data into a cube or whatever form the business is asking for, and then runs it. In savvy organizations this whole process might take just 12 hours. In a worse case it might take months and rack up millions of dollars in costs. In either case, there is still an unacceptable amount of latency in accessing the data—and a significant cost in personnel and infrastructure associated with the whole process.

Get Smart

This strategy has been evolving in parallel with the previously mentioned ones over the past few decades.

> "Getting smart" means storing the data in the most efficient format possible for the job. One might argue that this is just pre-processing, but there are some important differences.

The first difference is that no information is lost; the original data set can be completely reconstructed. Second, data can be updated in place and in near real time. When updates are made, the whole data set does not need to be reprocessed in order to update it. Finally, the data can still be queried in a flexible, ad-hoc manner because it is not built specifically for only certain queries as it is with pre-processing.

The very beginning of "get smart" goes back to some of the first databases and the notion of indexes. In many databases, indexes are created as auxiliary data structures which help to look up data for particular purposes quickly. An index might help answer queries with sorted data or might avoid additional I/O by storing pointers to certain sections of the data based on the query parameters.

Indexes are helpful, but the real performance gains come when you start playing with how the data itself is stored. Some of the first columnar databases came along in the early 2000's. These stored data column-by-column instead of row-by-row and were a great advance for analytical workloads. Many analytical queries only deal with a subset of the columns in the data, so a columnar format makes it easy to do sequential I/O on only the columns of interest rather than having to perform full table scans.

Another benefit of the columnar format is that it tends to put like data with like which makes the data far more compressible. Compressed data means even less I/O, and in some cases intelligent algorithms can operate on the compressed data without first decompressing it.

Putting it all Together: The Future of Latency

Many of the aforementioned techniques for reducing latency are combined in an effort to drive down the latency floor. The latest, more popular big data solutions are using a combination of "get smart" with "scale out" techniques to achieve reasonably speedy performance. Columnar formats like Parquet and ORC, or even in-memory columnar formats like Arrow can be paired with scale-out processing technologies like Apache Spark to yield some formidable data processing power.

All that being said, it is still extremely difficult to push into sub-second latencies for analytical queries on huge data sets. Shrinking a query which previously took days down to only a few seconds may sound like a successful ending to the latency story. Simply put, it is not.

New capabilities beget new applications. What was once a single analyst painstakingly building a quarterly report for the CFO, tweaking her SQL, letting it run overnight, and praying for correct results in the morning, is now an entire marketing department curiously exploring a new user-friendly GUI. The interface lets them slice and dice by every conceivable metric, zooming in and out on different segments of the population, hunting for those cliques and personas which have both the means and the need to buy their product. They can test ideas and assumptions, iterate and explore in seconds what previously would have taken days, significant manpower, and a cumbersome process.

With big data analytics now being exposed in a UI that's being served to a broader and less technical audience, a single page might generate dozens of backend queries to populate a dashboard with invaluable insights. Suddenly a query returning in seconds feels sluggish—it now needs to be milliseconds!

In addition to the growing population of less technical end-users, there has been an explosion in AI technologies and IoT applications that consume unlimited amounts of data and need it faster than ever.

AI engines have the ability to make use of previously unfathomable amounts of data and turn it into favorable outcomes in infrastructure, medical, security, marketing, sales, and research applications. The future of our success relies on finding faster ways to ever greater amounts of data.

Molecula: Breaking the Latency Floor

There is a more efficient way to scale. Molecula breaks through the latency floor with an entirely new paradigm for continuous, real-time data analysis. Molecula's approach to solving latency in big data access eliminates the need to pre-aggregate, federate, copy, cache or move source data. Molecula extracts features from source data without creating copies or moving the data itself, providing scale, performance, and increased control. All of this translates into faster data, more data, and easier-to-access data.

Molecula's Methodology

Molecula's feature store is an overlay to conventional big data systems that automatically extracts features, not data, from each of the underlying data sources or data lakes and stores them into one centralized access point. The feature store maintains up-to-the-millisecond data updates with little to no upfront data preparation. This is achieved by reducing the dimensionality of the original data, effectively collapsing conventional data models (such as relational or star schemas) into a highly-optimized format that is natively predisposed for machine-scale analytics and AI.

When Molecula ingests data it splits the values and the features apart, but, crucially, it retains both of them, so it can respond to queries while also being able to recreate the original data set from the information it stores.

In the quest to keep getting smarter, Molecula builds on the best techniques available. Columnar storage is smart because it breaks data apart in a way that makes it more amenable to analytical workloads. Molecula takes this idea to the extreme. After breaking data out by column, it is broken down by each unique value within the column, then the values themselves are separated from the data describing which records actually have those values.

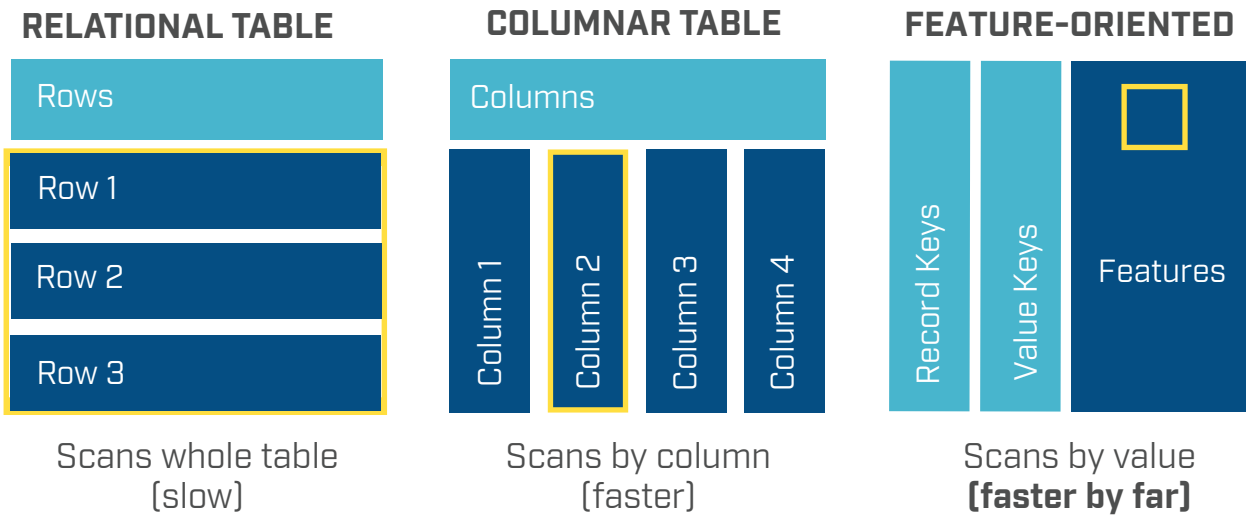


Figure 5. Comparison of traditional database formats versus Molecula's feature-oriented approach to breaking down and storing data

This way of breaking down the data has many advantages for analytical workloads and data storage in general. The obvious advantages are extensions of the columnar advantages. It is only necessary to read the data needed for a particular query. For columnar data stores, only data for the particular columns relevant to the query rather than the whole table is scanned. In Molecula, only data relevant to the particular features of the particular columns relevant to the query is scanned.

In columnar stores, data in columns can often be compressed more efficiently because the values are closely related. With Molecula, the majority of the data becomes a feature map that describes which records have a particular feature. These feature maps remain independent of the features themselves which are compressed using the same highly optimized approach (a variant of Roaring Bitmaps). Roaring Bitmaps are a form of homomorphic compression which can read and write features without decompressing. They are a type of succinct data structure.

This feature extraction approach has some other benefits as well. When breaking data out by feature, it becomes very natural to efficiently represent "set" types where a record can have multiple features for a particular feature. Traditional databases either have to use multiple tables and join across them or use special column types which aren't represented as efficiently. In this way, Molecula can actually simplify the database schema while simultaneously storing the data more efficiently.

Separating access to a field into a "feature map" and "features" as Molecula does is unique. Since the data is broken out by feature, it's possible to share the pattern of associations between records and features without sharing the values themselves (or vice-versa). This is a form of anonymization that can happen completely automatically with no overhead because a user is simply choosing not to expose certain parts of the data—it's already stored separately.

Applications of Molecula

Molecula is primarily focused on opening up new use cases for clients by shattering the latency floor compared to legacy systems. However, IT departments using Molecula often find ways to replace OLAP Cubes, Analytical Data Lakes, and other redundant systems with Molecula.

When this happens, **cost savings can be between 10-100x** compared to the systems being replaced. This is true for the reduction of hardware footprint and for the data movement and network costs that are typically associated with information era systems

For example, in the situations where Molecula replaces Elasticsearch, there has been a 10x reduction in data footprint, a 1000x improvement in performance, and the ability to do all of this without the typical pre-aggregation or pre-processing.

Conclusion

Data scientists and engineers around the world employ numerous strategies to reduce latency with varying degrees of success. Delivering vast historical data sets or extensive volumes of streaming data stored across multiple silos and geographies in real time is only fundamentally limited by the speed of light. However, in reality, we're not even close to worrying about reaching the physical limitations. The vast majority of query latency is bound up in the software reading and processing data. The huge gap between the limitations of physics and the current state-of-the-art is a world-changing opportunity.

Scaling up and scaling out are important parts of a solution, but they're not the ultimate answer, as both raise the latency limit, and neither create new efficiency—they push the problem down the road. Pre-processing is an entirely false path. Each instance of pre-processing solves a particular set of problems while creating whole new realms of inefficiency. We have to keep getting smarter.

Molecula is a fundamental advancement in low-latency data queries because of the way the data is stored and processed. Many systems have solved the problem of scale, but Molecu-la lowers the latency floor to the point that completely new use cases are now possible—real-time analysis and data at the speed of thought. Molecula's unique ability to dimensionally reduce data and store it in a feature store allows organizations to securely access, query and improve control over data at unprecedented speeds with a fraction of the hardware, eliminating the need to pre-aggregate, federate, copy, cache or move the original data.
Software engineers, data engineers, and machine learning engineers who are tasked with delivering data access to people or applications that need to query, segment, analyze, and make decisions on data in real time all stand to benefit from Molecula's technology.

Breaking through the latency floor is mining for time. Every moment that is recaptured by reduced latency can be correlated with increased value, whether it is a better user experience, a more accurate prediction, a real-time report, or a research breakthrough. The new value to be created is only limited by the laws of the universe.