+ a b | e a u

Hyper-charge Big Data Analytics Using Tableau

Jakob Mund, Product Management Senior Manager, Development, Tableau
Phillip Cheung, Product Management Manager, Development, Tableau

Contents

Introduction

As digital transformation efforts accelerate, organizations are collecting, processing, and analyzing more data—coming from more sources, in more shapes and forms—than ever before. Reliant on data and insights to inform time-critical business decisions, many organizations are struggling as their traditional data platforms and analytic systems cannot keep up with the volume, velocity, and variety of data today.

Tableau's mission is to help people see and understand data, no matter where it comes from, where it lives, or how much of it there is. In this whitepaper, we will demonstrate the Tableau platform's readiness to analyze terabytes of data—representing billions and tens of billions of rows—using Tableau Server, by focusing on three key areas:

- Bringing in large amounts of data into Tableau Server fast and efficiently
- Achieving interactive analytics at the speed of thought using Hyper, the core technology behind extracts
- Providing a natural language interface to enable all users to analyze big data

We'll dive deep into each of those topics, including providing benchmark results, discussing tool, configuration, and deployment options, and providing hardware sizing guidance.

Big data and Tableau

What is big data?

The term "big data," as it is used in this whitepaper, does not solely refer to large data sets, but is a combination of three key attributes: volume, velocity, and variety.

- **Volume** refers to the size of the data set, which can be a large number of rows, columns, or the size of values (e.g., large strings). For big data, the sheer amount of data to be analyzed is often too high to be handled by traditional analytics solutions. Many traditional systems are not built for that scale of data, resulting in unsatisfactory performance or running out of system resources.
- **Velocity** refers to the frequency with which new data is generated and introduced into the systems, regardless of whether existing data is updated or completely new data added. Velocity often imposes downstream challenges on capturing, transforming, storing and analyzing the data, and due to their fundamental design, many traditional transactional systems cannot keep up in high velocity scenarios.
- **Variety** refers to the fact that the data stored and analyzed can have a multitude of different forms. Beyond more traditional data types such as numbers, strings and timestamps—for example, representing product names, sales numbers, or transaction dates—modern analytics often includes geo-spatial data, internalization, semi-structured, or even unstructured data. This adds requirements on data analytics solutions, such as using optimized storage representations, specialized geo-joins, collation support, or support for semi-structured data such as JSON.

Data sets continue to grow due to rapid acceleration in digitization and because of the increasing number of useful data collection systems—like sensors, smartphones, cameras, microphones, networks, GPS devices, radio-frequency identification (RFID) readers, log files, web, or social media. At the same time, the demand to analyze this data in real time and at a very large scale is also increasing.

Analyzing big data

Bringing visual analytics to big data allows analysts, researchers, and business users to get valuable insights and make better decisions using data that was previously inaccessible or unusable. Businesses can use advanced data analysis techniques such as text analytics, machine learning, predictive analytics, data mining, statistics, and natural language processing to gain new insights from previously untapped data sources, whether independently or together with existing enterprise data.

Exemplary applications of big data analytics include:

- **Operational monitoring:** Collect data from your operational systems to quickly gain insights about how core business processes are doing, and rapidly react with counter-measures where required. Due to the nature of the use-case, this often involves collecting and analyzing a large amount of data that is continuously generated from several systems, while at the same, requiring frequent updates to find issues and opportunities earlier.
- **IoT data:** Characteristically, in internet-of-things (IoT) applications such as smart home, manufacturing, energy networks, a large number of independent producers (e.g., a sensor) together produce a large amount of data over time that is ever-changing. In addition, the data points produced by individual entities are often semi- or even unstructured data. Analyzing this data can allow to not only spot expensive issues with minimal delay, but using predictive methods, sometimes before they happen, e.g., as for predictive maintenance.
- **Call center analytics:** What's going on in a customer's call center is often a great barometer and influencer of market sentiment, but without a big data solution, much of the insight that a call center can provide will be overlooked or discovered too late. Big data solutions can help identify recurring problems or customer and staff behavior patterns on the fly not only by making sense of time/quality resolution metrics, but also by capturing and processing call content itself.
- **Fraud detection:** For businesses whose operations involve any type of claims or transaction processing, fraud detection is one of the most compelling big data application examples. Historically, fraud detection on the fly has proven an elusive goal. In most cases, fraud is discovered long after the fact, at which point the damage has been done and all that's left is to minimize the harm and adjust policies to prevent it from happening again. Big data platforms that can analyze claims and transactions in real time, identifying large-scale patterns across many transactions or detecting anomalous behavior from an individual user, can change the fraud detection game.

Using Tableau for big data: Key focus areas

While big data offers a lot of opportunities, there are also challenges that come when seeking to analyze it. As such, we recognize three distinct areas which are central to how Tableau performs for big data, and which will be described in more detail in this whitepaper:

- **Ingestion:** Depending on the volume, velocity and variety of data, efficiently bringing big data into any system can be very challenging. To this end, Tableau offers multiple options to ingest data from virtually any data source. By offering unique benefits in terms of functionality, performance and ease-of-use, it's critical for big data use cases to choose the right set of tools. In this paper, we will dive deep into how Tableau Prep, Tableau Desktop, the Hyper API and the Tableau Server REST API can work together to achieve the best results.
- **Interactive visual analytics:** Providing users with an analytical experience for big data at the speed of thought poses very high demands on the data source. If your data source is not able to handle big data, Tableau provides a fully integrated data engine called Hyper that offers best-of-breed performance, even when analyzing billions or tens of billions of rows. Hyper allows you to bring interactive analytics to big data with stellar performance. In this whitepaper, we will detail how to get the most out of Hyper when using Tableau.
- **Natural language analytics:** While dashboards and visual exploration are important, an increasing number of users expect to interact with their data in a more intuitive way. With Ask Data, Tableau users can get answers and insights by typing natural language questions to engage in a conversation with their data.
- **Deployment and configuration:** When using Tableau Server on-premises or installed on your cloud infrastructure of choice, server topology and configuration will affect your outcomes when working with big data. We will provide guidance for designing a Tableau Server configuration that will allow for optimal performance in both data ingestion and analysis.

Boost analytical performance using Tableau extracts and Hyper

What are extracts, and why use them (especially for big data)?

Tableau provides the option of connecting to data live or extracting data into its high performance data engine Hyper. Users can even toggle between a live and extracted connection to test performance in both modes. This flexibility is a unique option which other analytics products do not offer. However, there are some advantages to using extracts when working with big data.

In a nutshell, extracts are saved snapshots of data that you bring into Tableau Server or Online to improve performance or to take advantage of Tableau's rich analytics capabilities that may not be available or supported in your original data. When you create an extract of your data, you can optionally reduce the total amount of data by using filters and configuring other limits. After you create an extract, you can refresh it from the original data directly on the Server, either on demand or by setting up a refresh schedule. Alternatively, you could use Tableau Prep, our APIs or data integration solutions provided by Tableau partners (such as Alteryx, Informatica, Fivetran) to create and publish extract files as centralized, governed data sources. When refreshing the data, you have the option to either do a full refresh, which replaces all of the contents in the extract, or you can do an incremental refresh, which only adds rows that are new since the previous refresh.

Extracts offer a number of advantages over live connections:

1. **Performance:** Data extraction not only offers improved performance when the underlying data source is unacceptably slow, but it also can speed up the performance when the use of CustomSQL slows it down.
2. **Reduced load:** Replacing a live connection to an OLTP database—or any database—with an extract that reduces the load on the database that can result from heavy Tableau user traffic.
3. **Full set of analytical functions:** Extracts allow you to take advantage of Tableau functionality when not available or supported by the original data, such as the ability to compute distinct counts or statistical measures such as percentiles, correlations, or covariance.
4. **Consistent data:** Extracts can be used to gather a snapshot in time of your data. Whereas live connections to data stores operate on ever-changing and potentially inconsistent data, creating an extract at a point in time can provide a more consistent look into the business, e.g. after working hours, at the end of a fiscal quarter, or before launching a new product offering.

5. **Pre-aggregations and materialization:*** Tableau provides the option to aggregate data for all visible dimensions, known as an aggregated extract. An aggregated extract is smaller and does not contain all of the row-level data but only the necessary aggregations. Accessing the values for additive aggregations in a visualization becomes near-instantaneous because all of the work to derive the values has already been done. Similarly, Tableau allows you to materialize calculated fields, resulting in all defined calculations being converted to static values upon the next full refresh. This can increase performance for CPU bound workloads, like when working with expensive string calculations (which tend to be significantly slower than numeric and/or date calculations). Note that, however, this increases the time required to update extracts, and might impact performance for workloads that are limited by disk speed or memory, as it required to fetch more data from disk and load into memory.

Hyper technology explained

Hyper is a high-performance database technology that is fully integrated into the Tableau platform. Hyper exploits the capabilities of today's hardware and cloud infrastructures and utilizes state-of-the-art query compilation and optimization to provide users with query results at the "speed of thought" on data of all shapes and sizes. To achieve this, Hyper was built from the ground up based on cutting edge research.

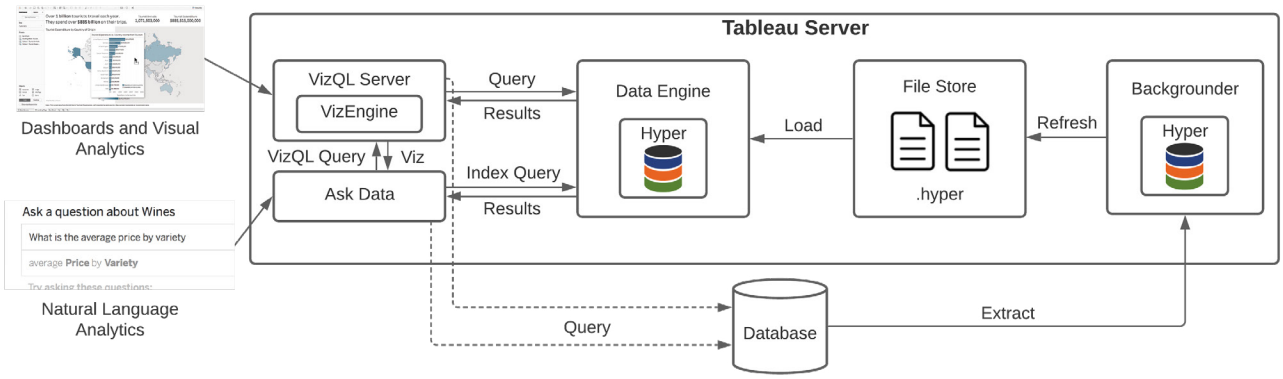
- **Morsel-driven parallelism:** Hyper was built without relying on any legacy code bases and is, thus, tailored to run on modern computing resources. It optimally uses multi-core and vector-processing capabilities with its *morsel-driven parallelism*. This fine-granular parallelization allows query processing to scale almost linearly with the number of available cores. Piecewise segmentation of complex processing tasks allows all cores to contribute equally without idle times. This makes Hyper resilient to often-encountered data skew which limits parallelism in traditional systems, where some processing units struggle with larger tasks while others are idling. In Hyper, all processing units terminate their tasks in a "photo finish," lagging at most one morsel task behind. With Hyper's adaptive, morsel-driven parallelization, the database system can easily be deployed to any scale of hardware or cloud instance—from laptops to desktops to large multi-socket servers with hundreds of cores. Therefore, when data volumes grow, the investment into new, more powerful hardware or cloud instances is fully leveraged.
- **Adaptive query compilation:** Hyper pioneered adaptive query compilation which ensures optimal processing for queries of any size: complex queries are translated into highly optimized machine code to maximize execution efficiency while "smaller" queries are interpreted for immediate response. Traditionally, database systems relied entirely on interpreting queries, one operator at a time. In conjunction with interpreter overhead, these context switches induce a severe layer of indirection. Contrary to this tedious query interpretation, Hyper compiles entire processing pipelines in a data-centric fashion in order to avoid context switches.

- **Column-store representation using data blocks:** Furthermore, Hyper achieves best-possible cache locality by storing data in columnar representations. This high locality enables clock-speed scanning of data as the processor's prefetcher can proactively load data into caches. In combination with the multi-core and vector-processing parallelism, this enables the evaluation of queries on hundreds of millions or even billions of data records interactively, i.e., within seconds or even sub-seconds. The columnar data is partitioned into so-called data blocks that are individually compressed using the most appropriate and state of the art algorithms to guarantee best possible space efficiency and fast processing. Additionally, for each data block, a small synopsis of the stored values is maintained. This allows Hyper to narrow the scan range within a data block and skip entire data blocks depending on a query's filtering predicates. Achieving high cache locality, reducing data size, and narrowing the amount of data that needs to be scanned to answer a query is essential when working with big data sets.
- **Best-in-breed query optimization:** Hyper has the most advanced logical query optimizer. By way of precise cost and sampling-based cardinality estimates, it selects the optimal evaluation plan for each individual query. Where other systems have to resort to (often failing) heuristics, Hyper can find the optimal plan even for very large join queries by using dynamic programming in combination with a clever sub-plan enumeration technique. Queries with arbitrarily complex nested sub-queries are fully unnested such that nested loop evaluations—and their quadratic processing costs—are avoided entirely. The unnested queries can then be executed using Hyper's highly optimized hash join processing technique that leverages its above-mentioned parallelization. Finding an optimal query plan is crucial as it can impact query execution speeds by many orders of magnitude.
- **Main-memory optimized:** Hyper persists data in a compressed and efficient format in .hyper files on disk. While Hyper is optimized to leverage available main memory for fastest query execution, data does not have to be present in memory. To process a query, Hyper scans through data from disk, reading only what it needs, and retaining in memory only what is needed to answer the query. In the unlikely event of running out of memory, Hyper will fall back to a processing mode that requires little memory by writing intermediate results back to disk.

In summary, Hyper provides responsive, interactive user experiences at the speed of thought on large data sets, even for the most complex analytical queries. Starting with Tableau 10.5, this technology is used as the data engine that serves all the requests going against extracts as well as federation among different data sources and is the processing engine used by Tableau Prep.

How are extracts and Hyper used to boost analytical performance?

Essentially, extracts are used to substitute queries against the original data source with queries against the Tableau Data Engine powered by Hyper technology. To serve a dashboard and it's visual exploration, or when asking your data questions in natural language using Ask Data, a number of internal Tableau Server components are involved, as illustrated in the following diagram:



The flow is as follows:

1. **User opens or interacts with a dashboard, or types in a natural language question for a published data source.**
2. **Translating user actions into queries by the VizQL Server or Ask Data:** For viewing, exploring or authoring vizzes, the VizQL server process translates user actions—such as the request to view a certain viz or changing a filter—into queries against extracts (or, in the case of live connections, against the original data source). Ask Data translates the natural language questions into a VizQL query first, which is then further translated into a query that can be issued against the Data Engine (Hyper) or the remote data source. Before using Ask Data, an index must be built, and Ask Data will use the Data Server to either query against the Data Engine/Hyper, or a remote data source.
3. **Execute analytical queries using Hyper:** All requests against the data engine are served by Hyper. Usually, for analytics, this involves analyzing row-level data and providing aggregated results. With the technological advantages described above, Hyper is able to answer complex queries on billions of rows at the speed of thought.
4. **On-demand loading:** Typically, Hyper depends on the necessary extract data being loaded on the same machine by using the File Store. The File Store is responsible for giving access to and managing how the .hyper files are stored and replicated in a Tableau Server cluster. Alternatively, with the Tableau Server Management Add-on, Tableau Server can be configured to use a network attached storage (NAS) instead. This enables multiple users and heterogeneous client devices to retrieve data from a centralized disk capacity, reducing backup time and network data transfer.

5. **Viz rendering:** With the query results provided by Hyper, the VizQL Server will then layout and render the visualizations for the user, independent if this is for a dashboard or a question asked using Ask Data. For some visualizations, the VizQL Server will use a sub-process called VizEngine to perform additional computations that are required to render the final viz—for example, if they are on a visualization-level or not supported by the original data source or in Hyper, such as certain table calculations or geographical joins.

Note that, in order to use extracts, the data must first be loaded from an external data store into the File Store. This can be achieved in various ways that will be explored further down in this whitepaper. Most commonly, the Backgrounder processes on Tableau Server will fetch data—either ad-hoc or according to a schedule—from the original data source, and use a local Hyper instance to update the .hyper extract file in the File Store.

What performance can be achieved with big data and Hyper extracts?

While the actual performance from extracts depends on a large variety of factors, we conducted a series of experiments to provide size-specific reference points for the fast query response times of the Hyper technology. In addition, we compared the results with common alternatives for big data. Note that there are many variables that can impact performance, including data model complexity, hardware sizing, concurrency, configuration, and workbook design, and the actual results will vary from case to case.

We used the following testing environment:

- **Infrastructure:** We tested Hyper on different sizes of AWS EC2 instances ranging from m5d.8xlarge (32 cores, 128 GB memory) up to i3en.metal (96 vCPUs, 768 GB memory).
- **Data set:** We choose the industry-standard TPC-H and TPC-DS benchmarks as our synthetic test data sets. Both benchmarks model a decision support system, and allow you to apply scale factors to obtain different order of magnitudes in data size. We tested started from 1GB of uncompressed data (about 10 million rows) up to 10 TB (about 100 billion rows). The following table provides additional details on the tested data-sets and workloads:

Data Set	Size (raw CSV)	# Tables	# Rows*	# Columns	Description
TPC-H (industry-standard synthetic benchmark)	1 GB 10 GB 100 GB 1 TB 10 TB	8 = 1 fact + 7 dimension	6M 60M 600M 6B 60B	61	TPC-H models a wholesale retailer database. The supporting schema contains vital business information, such as customer, order, and product data.
TPC-DS (industry-standard synthetic benchmark)	1 GB 10 GB 100 GB 1 TB 10 TB	24 = 7 fact + 17 dimension	2.8M 28M 280M 2.8B 28B	>300	TPC-DS models a retail product supplier database. The supporting schema contains vital business information, such as customer, order, and product data.

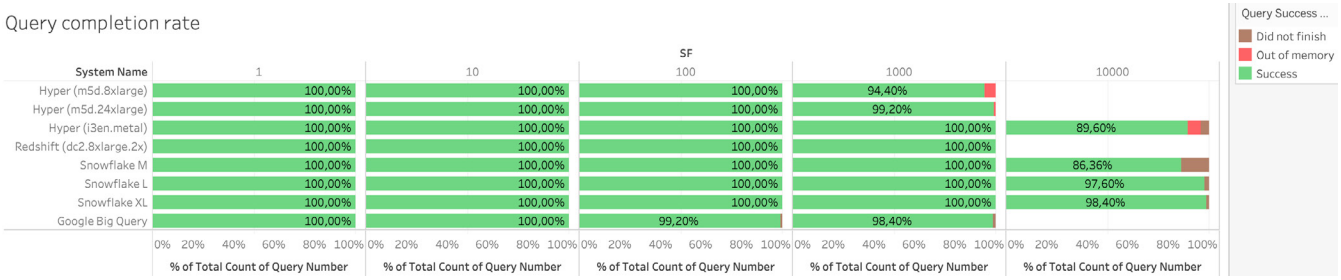
**Number of rows in largest fact table: line_item for TPC-H and store_sales for TPC-DS.*

- **Testing procedure and workloads:** We tested query processing performance in Hyper and several market-leading alternatives (as a live connection to the benchmark data) using the SQL queries as defined by the TPC-H and TPC-DS benchmarks, excluding data manipulation statements as those are not issued when analyzing the data. To load the data, we created .hyper extract files for the above data sets for Hyper respectively loaded the data directly into the tested alternative systems. Note that we tested query performance only; rendering, layouting, and query pre- and post-processing for visualizations are not part of this benchmark. The Hyper process was executed and monitored in isolation to minimize noise, and all tests and measurements were fully automated.
- **Third-party comparisons:** In addition to Hyper, we tested and compared query performance with the following systems:
- **Snowflake:** We used a newly-provisioned Snowflake instance with warehouses of size M, L, and XL to query the data with the default configuration settings.
- **Google BigQuery:** We used Google BigQuery on demand. We did not provision any dedicated instances.
- **Amazon Redshift:** We used a newly-provisioned two-node cluster (minimal size) consisting of two dc2.8xlarge instances, with the default configuration. Technically, this configuration should be most similar to Hyper running on a single m5d.16xlarge instance.

Query success

Querying large data volumes might quickly become infeasible, depending on the query generated against the data source. In order to assess the feasibility of querying large datasets, we measured the query success as the percentage of queries that finished divided by all queries issued. For those that did not finish, they either run into a timeout of 15 minutes, or out of memory (for Hyper only).

Results:



Key findings:

- For Hyper, query success rate was mainly determined by having sufficient main memory. While Hyper will use the disk to finish queries that exceed memory, this feature was disabled for this experiment as spooling will not improve success rate for the set query timeout.
- For dedicated systems, query success was determined by running into a timeout. While those can usually be increased, any query performance exceeding the default values of several minutes (e.g., 500 seconds for Snowflake) can be argued to be inadequate for analyzing large data volumes.
- For the tested datasets and workloads, 128 GB RAM were sufficient for querying extract from ingesting up to 100GB of CSV data (several hundreds of millions of rows). For data sets in the 1-10 TB range, 384 GB are recommended but cannot guarantee success for any query.
- In production, when analyzing very large datasets, spooling-fallback messages in Hyper's log files indicate that Hyper is using the disk to finish a query. If those are present, consider adding more memory to the nodes running Hyper or pre-aggregate your data.
- For Tableau Server deployments with a high number of concurrent users, make sure that the nodes are not oversubscribed. A deployment optimized for query-heavy environments is advised (see below or here). If necessary, add more Hyper nodes to redistribute overall load.

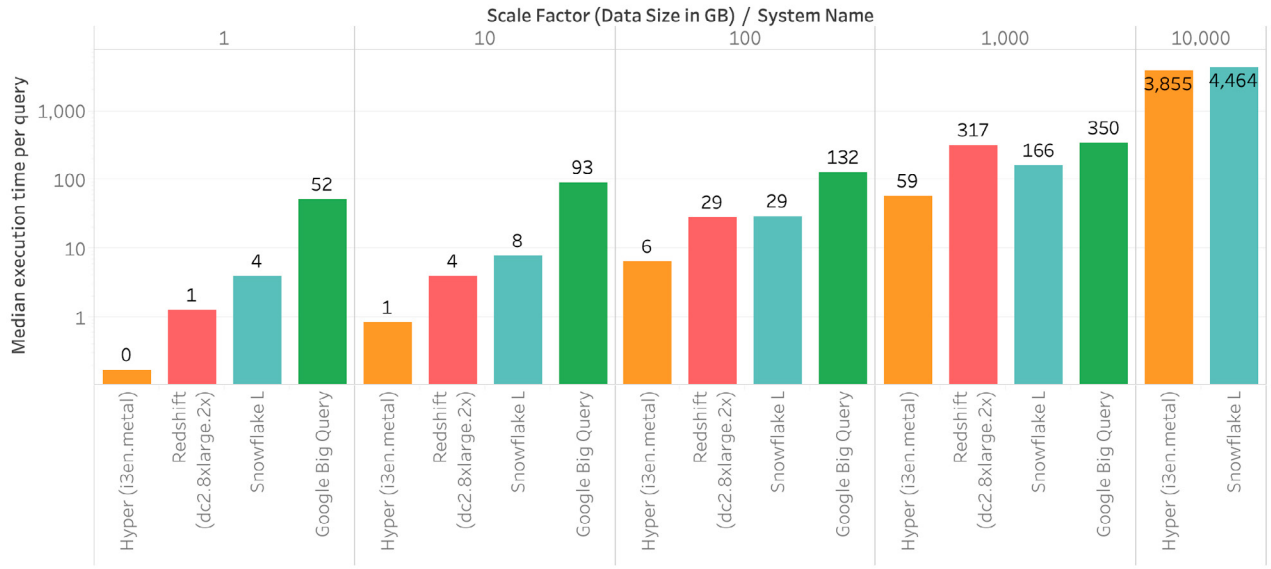
Query performance

To assess query performance, we used the query execution time as reported by Hyper or the dedicated systems. For each system, an initial warm-up run was performed to fetch the data from the cold store and warm caches. Then, we re-executed all tests by consecutively running each query three times, measuring the median execution time for each query.

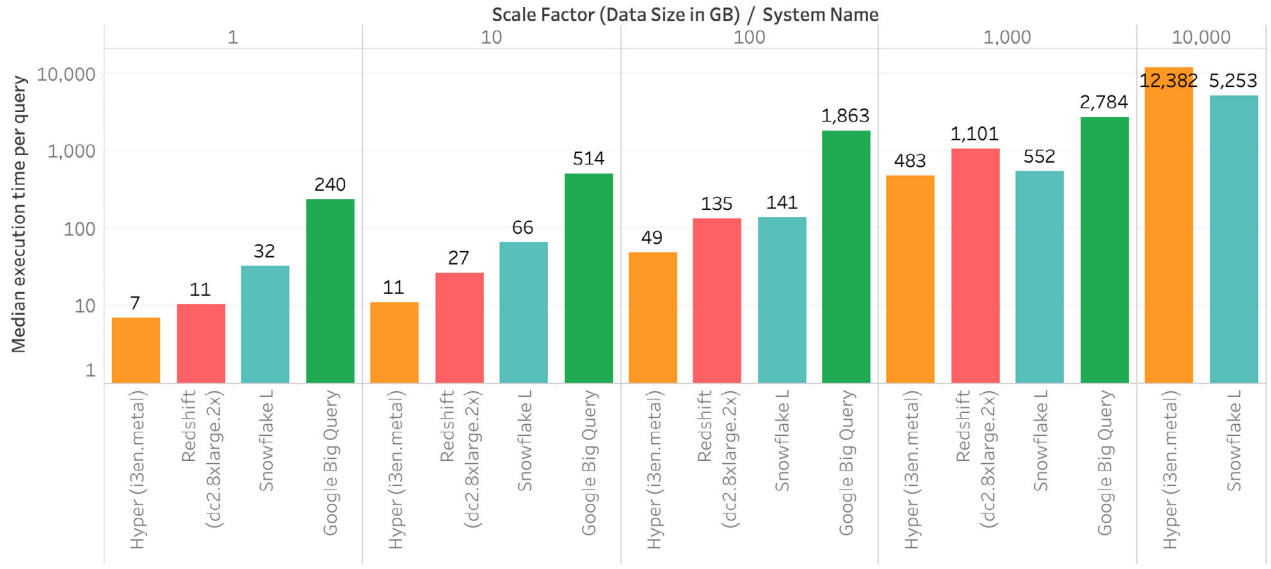
Results:

The following charts show the total execution time for the TPC-H and TPC-DS datasets, per data size and for each system. Please note the logarithmic scale reflected in the y-axis.

Sum of TPC-H benchmark execution time in seconds per scale factor and system



Sum of TPC-DS benchmark execution time in seconds per scale factor and system



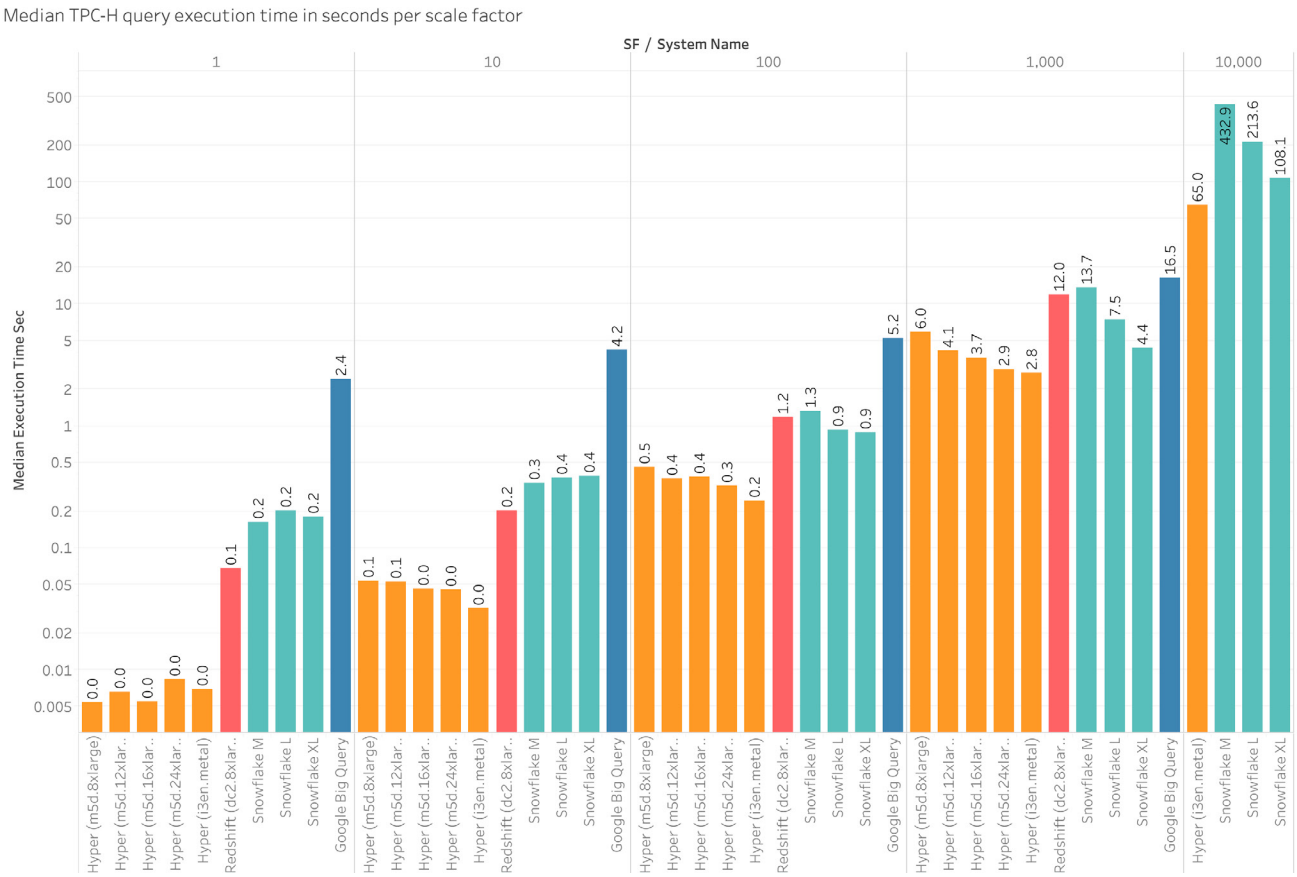
Key findings:

- In our testing, Hyper was able to provide second or sub-second query times for data sizes up to 1 TB of uncompressed data (billions of rows), given adequate hardware sizing.
- Hyper can offer comparative or significantly better performance than the other systems tested, given similar or smaller instance sizing.

Scalability and hardware sizing

To better understand hardware sizing, we tested the same data sets and workloads on different AWS EC2 instances sizes ranging from m5d.8xlarge (32 cores, 128 GB memory) up to the i3en.metal (96 vCPUs, 768 GB memory). All machines were equipped with NVMe SSDs.

Results:



Key Findings:

- **CPU / Core Count:** For large data sets, query performance in Hyper scales very well with the number of cores. Doubling from 32 vCPU (8xlarge) to 64 vCPU (16xlarge) reduced average execution times from 6.0s to 3.7s, and from 48 vCPU (12xlarge) to 96 vCPU (24xlarge) from 4.1s down to 2.9s.
- **Memory:** As described above, 128 GB RAM were sufficient for querying extracts of up to 100 GB uncompressed / hundreds of millions of rows. For data sets in the 10+ terabyte range, 384 GB are recommended but cannot guarantee success for any query. Memory consumption heavily depends on the workloads, hence we recommend testing workbooks on big data by looking for spooling-fallback messages in Hyper's log files. If those are present, consider adding more memory to the nodes running Hyper or pre-aggregate your data.
- **Disk:** With growing data size, and given sufficient CPU and memory, I/O speed can quickly become the bottleneck for performance. This is especially true for workbooks that require scanning all the data, e.g., when using distinct count. As a rule of thumb for big data, we strongly recommend using SSDs with high throughput (e.g., NVMe SSDs). In terms of disk size, we recommend to size disks to at least 3x of the data's size on disk to allow for spikes in temporary disk space requirements when refreshing the data. Note that this rule of thumb is meant to be applied to the sum total extract data used in dashboards or refreshed at a given point in time, and not all data brought into Tableau.

Asking the big questions: Using Tableau Ask Data with big data

What is Ask Data?

Ask Data allows to analyze data using natural language by simply typing in a question and instantly getting a response, right in Tableau. You can continuously refine your question as you explore your data, then share the findings with others as a Tableau workbook. Answers come in the form of rich data visualizations with no need to understand the structure of your data, thereby helping users get to insights faster.

Ask Data interprets the intent behind ambiguous and vague questions to give you accurate results. You can easily customize the underlying semantic model to your organization by adding synonyms, definitions, and calculations.

How does Ask Data work?

Ask Data works with all existing published data sources, both live connections and extracts, in Tableau Server and Tableau Online with no additional setup required.

Before Ask Data can be used, the data source has to be indexed. More specifically, Tableau analyzes and creates an index for each column individually that is later used to interpret a natural language question into a query that is issued—in case of extracts—against Hyper or directly against the data source for live connections. Internally, Tableau uses our Data Server service to retrieve the relevant data and metadata for each column, and then stores it in ElasticSearch (ES) as an in-memory, inverted index.

Once the data source is indexed, Ask Data will use the ES index to provide metadata, recommendations and translate natural language questions into queries that can be issued against either Hyper or the live data source. The time it takes to translate a question into a query is called the query interpretation time. Once the query is generated and issued against the data source, the performance of the underlying data source will determine how long it takes to answer the query. The results of the query are then visualized in Tableau, allowing users to continue the dialog with their data by asking additional questions or building a workbook from those results.

Using Ask Data with big data

While Ask Data works the same indifferent of the data size in principle, there are some things specific to big data to keep in mind:

- **Indexing row restrictions:** Ask Data only indexes the top 10k values for an individual field. We do this because in practice we've found that it's common for high cardinality fields to have overlapping values, or values which are highly ambiguous and conflict with syntax used to ask analytical questions. Restricting indexing reduces likelihood of spurious conflicts, while ensuring frequently appearing values are easily recognizable.
- **Data Server timeout:** Ask Data is dependent on the Data Server when it indexes a data source. There's a default timeout of 1min per field from Ask Data. For queries on very large data sets, the timeout needs to be increased. In our testing, we found this to be necessary for data sources of 100 GB (100s of millions of rows) and larger.
- **ElasticSearch memory:** Ask Data allows us to change the memory that is allocated to ElasticSearch. While the default value of 256 MB is usually sufficient, increasing memory can substantially improve indexing and query interpretation time.
- **Field details modal (value list):** The field details modal lets users see field metadata and add synonyms to a field or its values. A maximum of 10k values are shown, even if a field has more distinct values associated with it.
- **Categorical filter widget:** There are some places in Ask Data experience where the system will try to fetch all records from the data source and not from the 10K indexed values. Depending on the performance of the underlying data source and the Data Server timeout value, this request might timeout and if unable to handle that many records.

Ask Data performance

We used the following testing environment:

- **Infrastructure:** To test Ask Data, we installed Tableau Server 2020.2 on c5.4xlarge AWS EC2 instances, providing 16 vCPUs and 32 GB RAM.
- **Data set:** We choose the same industry-standard TPC-H and TPC-DS benchmarks as with the Hyper testing. Both benchmarks model a decision support system, and allow to apply scale factors to obtain different order of magnitudes. We tested started from 1GB / <10 million rows up to 10 TB / <100 billion rows.
- **Testing procedure and workloads:** We measured both the time to index the data source and the time to interpret a natural language question, i.e., the process of turning a natural language string as provided by the user into an analytical expression. The time it takes the data source to execute the query, i.e., produce a data result which can be visualized, was out of scope for this benchmark as this depends on the performance of the underlying data source. Besides the default configuration (1 min data server timeout, 256 MB ES memory), we also tested with no timeout (timeout=0) and 1 GB memory for ES.

Ask Data results:

Ask Data performance (1 GB ElasticSearch memory)				
	Indexing time	Query interpretation time	ElasticSearch size	Data Server timeout
1GB	58s	1.004s	51.7 MB	1 min
10GB	1m 4s	1.003s	62.7 MB	1 min
100 GB	7m 33s	1.001s	12 MB*	10 min
1 TB	1h 53m	1.002s	1 GB	10 min

**Partial indexing only; six text enriched fields with more than a million rows were not able to be indexed at all*

Key findings:

- Using the default configuration, Ask Data performed well, indexing data sources between 1–10 GB in less than 10 minutes, and interpreting the vast majority of queries in ~1 second.
- For data sets 100 GB and larger, indexing exceeded the default Data Server query timeout, and while Ask Data was still performant and functional, values were unable to be indexed.

- Increasing the Data Server timeout value to 10 minutes allowed Ask Data to index all data sizes, with time increasing semi-linearly with data source size. The recommended timeout setting will depend on both the size of the data set and the performance of the underlying data source. In practice, we recommend to incrementally increasing timeouts until indexing completes.
- Query interpretation performance remained constant and snappy, providing "speed of thought" answers at all tested data sizes.
- Increasing ES memory from 256 MB to 1 GB reduced indexing time by 13-43%. For larger data sources with frequent updates, we recommend increasing ES memory. Note that the effect on query interpretation performance was minor.

	256 MB ES RAM	1 GB ES RAM	Performance improvement
1 TB data source(ES size = 1 GB)			
Indexing time	3h 17m 0s	1h 53m 25s	42.64%
Query interpretation time	1.059s	1.002s	5.38%

Summary

Benchmarks show that Ask Data is able to handle large data sizes up to 1 TB of data. Natural language questions are transformed within a second into an analytical query, no matter the size of data, and the interactive performance becomes mainly determined by the speed of the underlying data source to execute the query. Preparation of the data source itself increases with the size of data, and while it can take a considerable amount of time for billions of rows, the performance and reliability of indexing can be significantly improved by increasing ElasticSearch memory and increasing the data server timeout, respectively.

Preparing and loading big data into Tableau

Tableau enables customers to choose from a number of different data integration approaches:

1. **Live** query against a dedicated data management system. With this approach, analytical query performance largely depends on the performance of the underlying data management system.
2. **Extract** creation over one or more underlying data sets. The raw or prepared data is copied or "pulled" into Tableau Server and subsequent analysis is later performed in Hyper.
3. **External** population of an extract using either a generated Hyper file or supported plain text format like CSV. In this pattern, a data preparation tool or direct use of APIs are used to "pushed" to Tableau Server.

Tableau offers multiple avenues for getting data prepared and delivered to Tableau Server for analysis. This includes the products and tools—such as Tableau Desktop, Tableau Prep, or the Hyper API—to prepare and ingest data into a publishable format, as well as different ways to publish the resulting data to Tableau Server. Each of these solutions provide unique advantages and performance characteristics.

Tableau content types and publishing methods

To understand the different ways for getting data into Tableau and their advantages and disadvantages, let's first take a look at the options of which content types can be published to Tableau Server, how they can be packaged into files, and how they can be delivered to Tableau Server.

Content types for publishing:

Tableau Server currently supports three types of analytics content: workbooks, datasources, and flows. Each content type can have data files associated with them and, depending on the file type, those files within Tableau Server are stored differently:

- For workbooks and datasources, **Hyper files** are pushed into and stored in the "File Store", which is either
  - The local filesystem on File Store node(s) for Tableau Server, by default
  - A file share that is globally accessible from a Tableau Server instance, when using the external file store feature offered as part of the Server Management Add-on
- For workbooks and datasources, all **other files** (CSV, Excel, JSON, spatial files etc.) are stored in Postgres repository **inside of twbx/tdsx**
- For **Tableau Prep flows**, all the input files are stored in the "File Store" (as described for Hyper files above)

Content delivery methods for publishing:

Most, if not all, file upload interfaces for Tableau Server only support single file uploads. As mentioned above, content types supported by Tableau are workbooks, datasources, and flows and, at minimum, they each have at least one file describing/backing them: TWB, TDS, and TFL. If the published resource is not based on an external data source, the clients must prepare a single file prior to publishing that contains the workbook / datasource or flow description as well as the data. Depending on the content type being published, TWB**X**, TDS**X,** or TFL**X** is that single file which is a zip file containing content definition and all its supplemental files such as the .hyper extract files or other files.

Both Tableau Desktop and Prep Builder operate on packaged workbooks/datasources or flows and don't publish individual data files but content packages. Therefore, for example, in order to publish a 100 GB Hyper file with Tableau Desktop (as a datasource), it must first create an TDSX archive containing said .hyper extract file plus the data-source description (.tds). Such compression is a resource intensive operation that require both compute power as well as additional temporary storage.

In contrast, Tableau Server REST API—and tabcmd as a simple command-line interface for this—do allow uploads of individual (unpackaged) .hyper extract files. Files uploaded in this manner do not require additional client-side compression, and Tableau is able to create a simple datasource wrapper at the final stages of the publishing operation.

Choosing the right tool for data preparation and ingestion

Tableau provides several tools to ingest and optionally prepare data, either for local consumption or to be subsequently published to Tableau Server for consumption by a broader audience.

- **Use Tableau Desktop:** Tableau Desktop provides a simple interface to load data from a variety of data sources, by offering out-of-the-box connectivity with more than 80 native connectors plus additional connectors offered through the Extension Gallery. To bring the data into Tableau and not rely on live connections, users can easily switch to extract the data. Also, specifically important for high volumes of data, Desktop provides easy access to aggregation, extract filters and only extract visible dimensions to reduce data sizes. Tableau data modeling capabilities allows to separate the logic model of the data used for analysis from the physical model, providing control over which tables are joined during ingestion time vs. during analysis time in order to optimize performance.

- **Prepare your data as you go using Tableau Prep Builder:** Tableau Prep Builder offers specialized capabilities and an interface that focuses on preparing your data in an intuitive, self-service fashion. Tableau Prep ships with 58 connectors out of the box to allow connecting to a variety data sources easily. It offers smart features to fix common data prep challenges, and employs fuzzy clustering to turn repetitive tasks, like grouping by pronunciation, into one-click operations. Advanced filtering and aggregation can be used to reduce the volume of your data by precisely select the data and aggregation level you want to load into Tableau.
- **Use the Hyper API to create the most performant and flexible solutions where necessary:** The Hyper API allows to create and locally modify .hyper files. It is available for C++, Python, Java and C#/.NET, and for Windows, Linux and MacOS. The Hyper API is build directly on Hyper's technology and offers the best raw performance when creating a .hyper file. Out-of-the-box connectivity is limited to CSV and .hyper files, but any data accessible through your code can be ingested into Hyper. Therefore, it is often combined with built-in language or third-party library support to access data to create the .hyper file. For consumption, the resulting .hyper file can be either directly opened in Tableau Desktop or Prep Builder for local consumption, or published to a Tableau Server instance via the Server's REST API or the tabcmd CLI tool (see below).

Comparison chart:

	Ingestion & local consumption			
	Out-of-the-box connectivity	**Ingestion speed**	**Data transformations**	**Data modeling capabilities**
Prep Builder	>50 connectors	High	Advanced	Essential*
Tableau Desktop	>80 connectors, extensible	High	Essential	Advanced
Hyper API	CSV, .hyper files	Highest	Anything (that code can do)	Essential*

**The Hyper extract file (.hyper) created through Prep Builder/Hyper API can be combined with Tableau Desktop to create a rich data model.*

Performance comparison:

To assess raw ingestion speed —without any additional operations—we measured the wall clock time for ingesting raw data (from CSV) into a Hyper extract file (.hyper). CSV input was chosen to minimize dependency on the performance of external systems. The output was written into a single, multi-table .hyper file in case of Hyper API and one single-table .hyper file per table using Tableau Prep Builder. Assessing the performance of joining data or performing other operations in Tableau Prep Builder was out of scope for our evaluation. The following table shows the results the TPC-H dataset ranging from 1 GB to 10 TB in uncompressed size:

Product	Input	Output	1 GB ~10M rows	10 GB ~100M rows	100 GB ~1B rows	1 TB ~10B rows	10 TB ~100B rows
Hyper API	CSV	Hyper file (multi-table .hyper)	10s	1min 28s	11min 51s	1h 51min	21h 6min
Prep Builder	CSV	Hyper files (one .hyper per table)	1min	4min 30s	41min 25s	5h 26min	Failed
Tableau Desktop	CSV	Hyper file (multi-table .hyper file)	1min 45s**	6min 30s**	1h	N/A**	N/A**

***For data sizes smaller than the tested 100 GB, results reported are interpolated expecting linearity. For larger data sizes, we marked the results as "N/A."*

Key findings:

- Prep Builder was able to successfully ingest CSV and generate a local Hyper file for data sets up to 1 TB in size. It ingested CSV at a speed of ~40 MB/s, corresponding to ~400k rows/s for the TPC-H data set.
- Hyper API was able to ingest all sizes tested (up to 10 TB), offering stellar performance of a constant ~150 MB/s, corresponding to ~1.4M rows/s for the TPC-H data set.
- Ingestion is currently not parallelized and speed is largely independent of instance type. For Tableau Prep Builder, ingesting TPC-H at 1 TB size required to upgrade from a m5d.8xlarge instance (32 cores, 128 GB memory) to the larger m5d.12xlarge instance (64 cores, 256 GB memory).
- We have also tested using Desktop to ingest the CSV data into Tableau. Internally, Desktop creates a so-called shadow extract for the CSV data. Performance-wise, we've measured Desktop to be slower than Prep Builder, taking 1 hour to ingest 100 GB of CSV compared to 41 minutes for Prep Builder.

Choosing the right publishing method

To share a locally created data source backed by an extract (.tdsx or .hyper), it must be published to a Tableau Server instance. As for loading and ingestion, Tableau offers different ways to publish such a data source, each offering its own benefits:

- **Tableau Desktop:** When signed in to a Tableau Server, Tableau Desktop allows to publish data—either embedded in a workbook (.twbx) or as a published data-source (.tdsx) directly through the UI. As both those formats are a compressed (zipped) format, publishing via Tableau Desktop will include compressing and decompressing any bundled .hyper files on the client and server, respectively. While incremental updates of extracts are possible on Tableau Desktop, re-publishing the workbook or data source will result in publishing the full extracts again, and is not recommended. Instead, refresh the extract directly on Tableau Server. If the data is reachable from the Tableau Desktop machine but not from Tableau Server, e.g., because it's a local file or due to network security policies, Tableau Bridge running on the Desktop machine can be used to push the data to Tableau Server during an extract refresh.
- **REST API / tabcmd:** Tableau Server's REST API includes endpoints for publishing which allow to publish workbooks (.twb and .twbx), data-sources (.tds and .tdsx) as well as Hyper extract files (.hyper). In the last case, the Server will generate a published data-source descriptor (.tdsx) as part of the API call, resulting in a published data-source on Tableau Server. However, this step does not include any client or server-side compression, often resulting in a net reduction of publishing time given fairly decent network speed. Uploads and their payload can be chunked into multiple requests, allowing to continue uploading if a network or any other error occurs, which is especially important for big data. The REST API also allows to append the data being published to an existing data source that has the same name, given both the data source on the server and the data source being published are Hyper extracts (.hyper files) with matching schemas. Both initial publishing and appending to a Hyper extract file (.hyper) require to be single table only; multi-table publishing is not supported.
- **Prep Builder:** Besides choosing local outputs in terms of a .csv or .hyper file, Prep Builder also allows to directly publish the output of a flow to Tableau Server as a published data source. Starting with version 2020.2.1, flow inputs and outputs can be configured to refresh incrementally so that only the new rows are retrieved and processed when the flow runs, saving time and resources. For Tableau Server instances with the Data Management Add-on, flows scheduled via Tableau Prep Conductor can automatically create or update published data sources on Tableau Server. Alternatively, Prep Builder's local output formats can be combined with Tableau Desktop or the REST API / tabcmd for publishing.

Comparison chart:

	Publishing Large Datasets to Tableau Server			
	Publishing methods	**Supported publish formats**	**Publishing performance**	**Supported data models**
Tableau Desktop	Full replace	.TDS(X), .TWB(X), .HYPER*	High	Full data model support (single + multi-table)
Tableau Prep Builder	Full replace, incremental append	.TDS(X), .TFL(X), .HYPER**	Medium	Single-table only
REST API / tabcmd	Full replace, incremental append	.TDS(X), .TWB(X), .HYPER**	Highest	Single-table only

**Publishing a Hyper extract file (.hyper) requires to create a data source (.tdsx) in Tableau Desktop prior to publishing.*

***Tableau Server will automatically generate a published data source (.tdsx) when publishing a Hyper extract file (.hyper).*

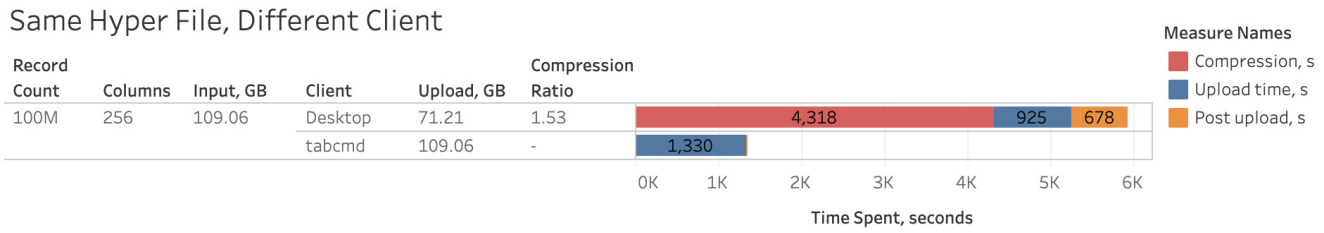
Performance considerations

We conducted scale tests for publishing. Text processing and file upload process scales linearly with file size growth. Besides disk space limitations, we have not observed obstacles that might not allow us to push content size beyond TBs of data.

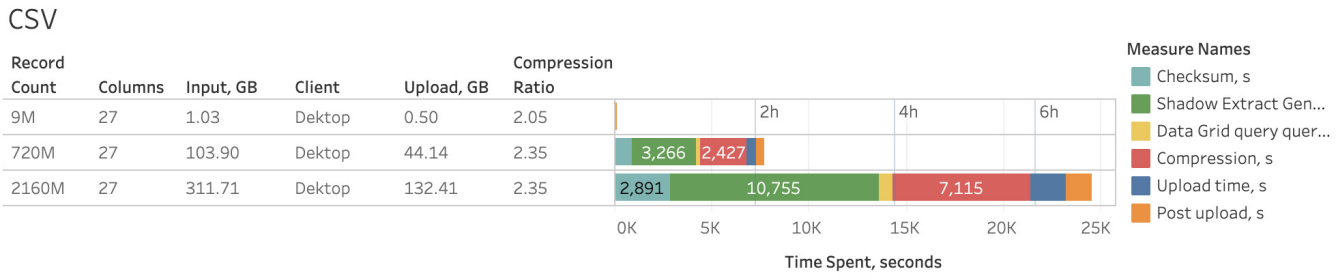
- **Ingesting and publishing directly from Prep Builder:** Tableau Prep Builder allows to publish as part of executing the flow. In our testing, Prep Builder was able to successfully publish a data source for data sets up to 100 GB:

			Data size (uncompressed, TPC-H)		
Product	**Input**	**Output**	**1 GB ~10M rows**	**10 GB ~100M rows**	**100 GB ~1B rows**
Prep Builder	CSV	Published data source (joined single table)	2min 30s	6min	1h 12min

- **File compression for packaged workbooks/data sources:** For larger file sizes, compression takes sizable portion of the whole publish process, and is CPU-bound while only utilizing one to two cores on the machine. In our testing of publishing a 109 GB extract, compression on Tableau Desktop did benefit from 30% upload size reduction, however end-to-end publishing time quadrupled compared to tabcmd:



- **CSV file handling:** Below is a visualization of CSV file publishing experiments via Tableau Desktop. Text files require non-trivial preprocessing in order to get data into the state where it can be analyzed and this preprocessing becomes more pronounced as the data size grows. The following steps must be executed every time CSV or Excel file is used for analysis: Checksum generation (single-threaded) and shadow extract generation (heavily multi-threaded, memory intensive).



Deployment and configuration options

How to design your topology for heavy extract use

When Hyper was first released in with Tableau 10.5 in 2018, the focus was on using Hyper as an embedded data engine for improved extract query speed. Since then, Hyper continues to grow in its capability to handle progressively larger query volumes. As extract use grows, we recommend customers use a topology called HoSS (Hyper on Standalone Server) that allows Tableau Server to scale out infrastructure, specifically for extract query loads. There are several key benefits to using a HoSS topology:

1. Dedicated HoSS nodes will reduce resource contention between extract query and other, resource intensive workloads such as those processed by VizQL Server.
2. Extract queries to HoSS nodes can be intelligently load balanced so that no one node is over or under utilized.
3. You have more control to scale out the parts of Server that most need more resources. If VizQL Server, Hyper, and Backgrounder are all running on the same node and slow extract queries is the problem, it will be difficult to isolate improvements by adding a second node with all three processes. With a HoSS deployment, you can add more nodes that will specifically improve extract query workloads.
4. A HoSS topology can help improve availability and uptime. In the event there is a hardware or network issue and one of the HoSS nodes is unavailable, VizQL Server can attempt to route request to other HoSS nodes.

To move to a HoSS topology you will need to identify which nodes will be dedicated to running Hyper. You may be able to move services to other nodes or you can add net new hardware. Once you have your hardware identified, you can add the nodes using TSM and assign the File Store and Data Engine roles. Below is a topology diagram that shows HoSS running on node 5 and 6.

Process	Node 1	Node 2	Node 3	Node 4	Node 5 (HoSS)	Node 6 (HoSS)
Cluster Controller	✓	✓	✓	✓	✓	✓
Gateway	✓	✓				
Application Server	✓	✓				
VizQL Server	✓ ✓	✓ ✓				
Cache Server	✓ ✓	✓ ✓				
Search & Browse	✓	✓				
Backgrounder			✓ ✓	✓ ✓		
Data Server	✓ ✓	✓ ✓				
Data Engine	✓	✓			✓	✓
File Store	✓	✓			✓	✓
Repository	✓	✓				

Hardware guidance

To get the most out of your HoSS deployment you will need to experiment with various hardware sizes and configurations to see what best fits your peak load performance objectives. Understanding how Hyper uses resources to process queries will help you start your hardware selection and reason between different configurations. Hyper is a high performance database technology and the key resources that impact performance are memory, cores, and storage I/O:

- **Memory:** When an extract-based viz query is processed for a user or background process, Tableau Server selects which HoSS node will process the query. That HoSS node will copy the extract from local storage, most often the server hard disk, into memory. This copying process can take time and optimizing this part is covered in the Storage I/O section. Having more available system memory allows the Operating System better manage memory usage for Tableau. HoSS uses system memory to store the result set of executed queries. If the result set is still valid and the Operation System has not evicted it from memory, the result set in memory can be reused which will be much faster than retrieving the data from disk again. Tableau Server's minimum hardware recommendation is 32 GB of memory but if you are expecting a high volume of extract based viz loads, you should consider 64 GB or 128 GB. Depending on the hardware cost, instead of scaling up to 128 GB of memory, it might be more cost effective to scale out to an additional 64 GB HoSS node.

- **Cores:** When processing an extract based query, the number of cores is an important hardware resource that can impact performance and scalability. CPU cores are responsible for executing a query and having more available cores will result in faster execution time. Generally speaking, doubling the number of cores will reduce the query execution time in half. For example, a 10 second query utilizing 4 cores will take 5 seconds if you have 8 available cores. The current Tableau Server recommended spec is to have 8 cores but if you deployment utilizes extracts, consider 16 or 32 core machines. An important thing to note is that if memory and I/O are your bottlenecks then increasing available cores will not improve your query performance.

- **Storage I/O:** Hyper is designed to leverage available performance of your Extract storage device to speed up query processing. We recommend picking fast disk storage like Solid State Drives (SSD) with high read/write speeds. Currently, SSDs that utilize NVMe storage protocol offer the fastest available speeds.

One final note to point out is that sizing resources for HoSS nodes only impacts the extract query performance. When loading a viz, there are many other process involved that make up total VizQL load request time. The VizQL Server process, for example, is generally responsible for taking the data from Hyper and rendering the visualization.

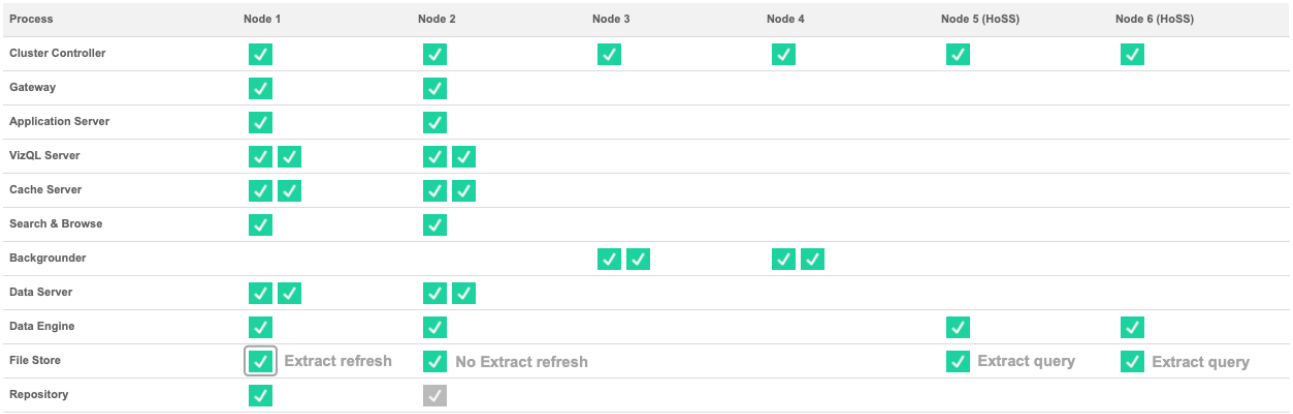
Tuning HoSS

Now that you have your HoSS deployment, there are additional features you can use to optimize performance:

- **Extract Query Load Balancing:** Hyper logs a server health metric on the amount of resources Hyper is consuming and also takes into account load from other Tableau processes that may be running on the same server node. Based on this information, extract queries will be sent to the node that has the most available resources to process the query. In addition to evaluating system resources, the load balancer will also determine if an extract already exists in memory on node and will factor that into the routing decision. This results in more efficient memory and disk utilization where extracts are not duplicated in memory across nodes. This feature is turned on by default in version 2020.2 and later. See the Extract Query Load Balancing help article for more details.
- **File Store node roles:** With File Store node roles, Server Admins have more flexibility and control over which nodes should be preferred for running extract queries and extract refreshes. As mentioned in our topology section above, HoSS nodes are dedicated to processing Extract Queries and run only the File Store and Data Engine processes. Below is an example of the same six node topology that with two dedicated HoSS nodes.

Process	Node 1	Node 2	Node 3	Node 4	Node 5 (HoSS)	Node 6 (HoSS)
Cluster Controller	✓	✓	✓	✓	✓	✓
Gateway	✓	✓				
Application Server	✓	✓				
VizQL Server	✓ ✓	✓ ✓				
Cache Server	✓ ✓	✓ ✓				
Search & Browse	✓	✓				
Backgrounder			✓ ✓	✓ ✓		
Data Server	✓ ✓	✓ ✓				
Data Engine	✓	✓			✓	✓
File Store					✓	✓
Repository	✓	✓				

By adding File Store to Node 1 and setting the node role to `extract-refreshes` and setting Node 2 to the `no-extract-refreshes`, incremental extract refresh jobs will be processed in Node 1 instead of Node 2. This reduces the amount of network traffic because Node 2 does not need to send the extract to Node 1 to be written to the File Store.



Additionally, since Node 1 also has the Repository and File Store processes, all of the data needed to perform a backup exist on Node 1 which can improve backup speed. Node 5 and 6 should be assigned the extract-refreshes role to ensure they only process queries for viz loads, subscriptions, and metric alerts. File Store Node Roles is available with the Server Management Add-on since the 2019.4 release. See the File Store node roles help article for more details.

- **Extract Query Interactive node role:** For HoSS nodes assigned the Extract Refresh node role, Server Admins can isolate and segment Interactive and Scheduled extract query and assign them to run on specific HoSS Extract-Refresh nodes. This is useful for times when there are a lot of users loading dashboards and during high volume subscription times. For example there are 1000 subscriptions scheduled for the 8 AM Monday mornings. At the same time many users are also loading dashboards at the beginning of their day. The combined volume of subscription and user queries can result in users experiencing slower viz load time. With the Extract Query Interactive node role, you can designate specific HoSS nodes to only accept queries for Interactive users (the ones who are looking at their screens waiting). These Interactive HoSS nodes would be protected from the high volume of competing subscription jobs and provide more consistent query times. Additionally, Server Admins can use this node role to better plan for Server growth as they can add HoSS nodes for Interactive and Scheduled workloads independently. The Extract Query Interactive node role is available since the 2020.4 release.

- **External File Store:** Many customers have invested in high performance storage devices like Network Attached Storage (NAS). The External File Store feature allows you to use these types of devices for File Store instead of using the local disk on a Server node. By having storage on a centralized location, you can significantly reduce the amount of network traffic spent on replicating data between File Store nodes. For example, when a 1 GB extract is refreshed using local File Store, the 1 GB of data is replicated across the network to all nodes that are running the File Store process. Using External File Store, the 1 GB extract only needs to be copied to the NAS storage once and all File Store nodes can access that single copy. The centralization of storage also reduces the amount of local storage needed on File Store nodes. Additionally Backup times leverage Snapshot technology to significantly reduce the time to complete a Server backup. While you don't need a HoSS topology to gain the benefits of External File Store, the additional workload management features with File Store node role and the Extract Query Interactive node role can be used together. External File Store is available with the Server Management Add-on since the 2020.1 release. See the Tableau Server External File Store help article for more details.

Conclusion

In this whitepaper, we presented a deep dive into:

- How to get data into Tableau Server in the most efficient way while making sure all requirements are satisfied
- Why and how to use Hyper and extracts to analyze big data at the speed of thought
- How to use Ask Data to provide natural language to big data to everyone in your organization

While all these aspects are important, the combination of all three is when Tableau really shines. Using the fastest ingestion technique helps you to bring big data in fast, which allows Hyper to power dashboards and interactive analytics as well as natural language queries from Ask Data.

Following the guidance provided in this whitepaper, organizations using Tableau do not have to make a trade-off between visual, self-service analytics and big data. Analyzing billions or tens of billions of rows not only becomes feasible, but interactive at the speed of thought—without sacrificing analytical power, relying on a myriad of vendors, tools, and data staging environments, or investing an unjustifiable amount of money.

Besides the areas discussed in this whitepaper, the following resources might be helpful:

- Best practices for Designing Efficient Dashboards
- Tableau Online Scalability: Overview and Proof Points
- Live Connections instead of Extracts: Delivering rapid-fire analytics with Snowflake and Tableau
- Managing Data Analytics at Scale: 5 Steps to Self-Service Analytics that Scales