Tableau Server Scalability

A Technical Deployment Guide for Server Administrators

Neelesh Kamkolkar
Product Manager, Data and Performance

tableau

Table of Contents

Executive Summary

Tableau is mission critical to many organizations. As many departments across these enterprises realize significant value from finding insights in their data, IT teams are working with the business to deliver Tableau as their enterprise-wide analytics platform. As enterprises start to deploy Tableau in this fashion, it is essential that enterprise architects and IT leaders understand how Tableau Server scales with data, content, and users and how it can be deployed and integrated across diverse and heterogeneous enterprise IT platforms in order to support the analytical needs of the business today and tomorrow.

This whitepaper is targeted towards enterprise architects and technical IT leaders. It provides a deep view into Tableau's architecture and how it scales with increasing workloads.

We have explored and tested how Tableau Server 10 scales and how the results compare to earlier versions of the product. In response to your requests, we expanded the scope of scalability testing to include background workloads, in addition to user scalability.

There are a number of system factors that can impact performance and scalability of Tableau Server. Some of the important system variables include workbook design, server configuration, infrastructure tuning, data environment, compute capacity, and networking. These factors are highly variable across different use profiles and deployments. The results of any given scalability test will differ as these variables are tuned or changed. In our attempt to mitigate and isolate variables, we ran our tests in a closed network lab on physical machines. Our goal was to minimize variability in measurements due to influences from external systems. We could then measure scalability metrics by modeling real usage of Tableau Server.

To this end, we started by analyzing a real production deployment of Tableau Server at its peak usage, and then modeled that usage in automated tests. This two-step approach mimics a very realistic workload. This approach also simulates realistic variances in how actual users and backgrounder workloads (mainly, data extracts and user notifications) might be exercising the system. Variances in this context include the amount of time users wait between interactions with the visualization or the number of backgrounder jobs being run on a schedule.

We modeled the variances as part of our tests to emulate real production conditions. We observed that the system scaled linearly when we added more worker nodes in a Tableau Server cluster. Our experiments pushed the server to peak load conditions beyond what we observed in the production environment. Production environment usage tends to peak for short bursts frequently throughout the workday. We measured load as expressed in throughput. Throughput is the amount of work that the server is processing in a given amount of time, or put simply: transactions per second. As shown below, we observed that throughput scaled from 4 to 18 transactions per second in our experiments. Each bar indicates an experiment that was run on the topology described in the column header (Standalone Server, Primary + 1 Worker etc).

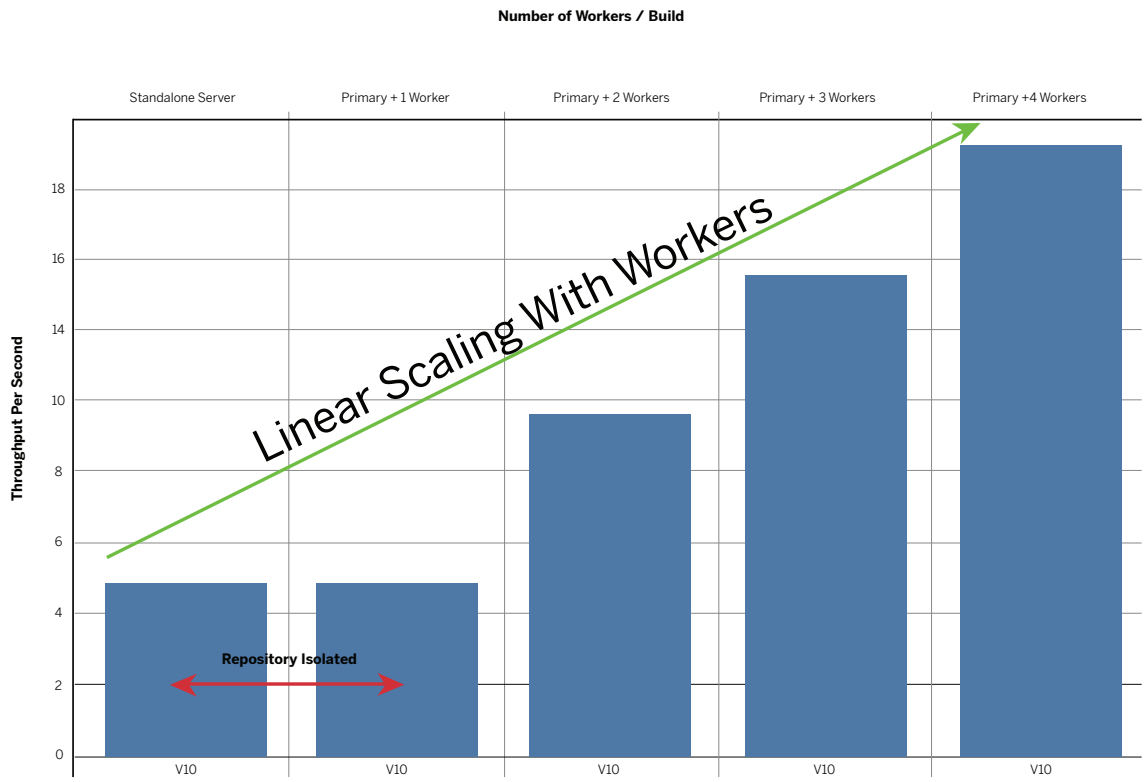

Figure 1: Scenario throughput per second

We identified the maximum number of active user loads we could push through a single server with 8 physical cores. An active user load is an automated set of operations acting against a Tableau Server. Later in this whitepaper, we describe in detail what operations are included in an active user load. After we measured the maximum number of active users that Tableau Server with 8 physical cores could support, we then used that active user number as the scale unit to test whether we can linearly increase load by adding homogenous 8 core worker nodes to the cluster.

The result: we observed that a single 8 core server can support up to 112 active users under realistic modeled sustained peak loads. When we scaled that baseline linearly—adding four homogeneous worker nodes with 8 cores each and a primary controller node with repository and base install only —we were able to support 448 active users. These are simultaneous users doing work on the system, so if you extrapolated that not all users are always using the system, you could expect to support anywhere from 500 to 10,000 users with a cluster configuration with 8-32 cores. As we'll see, such an extrapolation depends upon various parameters. All other variables being equal, server sizing in context of user numbers will rely on analytics use and data conditions. What our testing has shown is that Tableau architecture will allow you to continue to linearly scale your user base by adding more worker nodes to the Tableau Server cluster.

In our testing, we pushed the server to find where it began to return errors or time out. This stress testing provided an upper ceiling that defines the scalability and performance limits of the server.

While quantifying these limits, we then continued to push the design beyond production burst load characterization to sustained peak loads. Therefore, in practice, where a more stable state is the norm, the platform should support more usage than suggested in this paper. You can be confident that Tableau Server architecture is scalable to meet your current needs and your future growth as it scales linearly.

Each of the topologies and use cases are described in the table below. The Risk Profile column maps to variables such as whether the topology may be exposed to more or less risk based on various factors like automatic fail over, exposure to hardware failure, and available headroom for peak bursts.

Deployment Configuration	Use Case	Risk Profile
Standalone Server	Simple single server deployment	High risk for availability, high resource contention, no head room for peak load performance challenges
1 Primary + 1 Worker	Two node deployment	High risk for availability, lower resource contention with repository, no headroom for peak usage
1 Primary + 2 Workers	Three node deployment	Moderate risk for availability, lower resource contention, improved horizontal scale, very little headroom for peak usage
1 Primary + 3 Workers	Four node deployment	Moderate risk for availability, lower resource contention, improved scale, moderate headroom for peak usage
1 Primary + 4 Workers	Five node deployment	Lower risk for availability, improved scaling, available headroom for peak usage [1]

Table 1: Deployment topologies and use cases

[1]The available head room depends on many factors and is not guaranteed

It's important to understand these numbers in the context of the methodology and testing. To this end, the rest of the white paper focuses on the architecture and what makes Tableau architecture unique compared to traditional BI technologies, as well as our methodology and scalability results analysis.

VizQL breaks an age-old paradigm

If you are used to traditional business intelligence (BI) solutions or if you are new to Tableau, it may

help to understand some core differences with how Tableau works when compared to traditional BI. We have challenged the fundamental premise of the "query first, visualize next" approach to traditional BI. Instead, we believe data reveals insights when explored and questioned in real-time. For over a decade, Tableau has included patented technology called VizQL™ that combines query and visualization into a single platform. This powerful pairing of functionality allows end users to ask limitless questions of their data—querying, filtering, and analyzing – as they are visualizing their data. VizQL™ is a foundational language in Tableau that expresses users question and actions, translating them into queries that can be run against any data set in the enterprise or in the cloud. This technology has matured over more than a decade of engineering investment and continues to evolve to enable the next generation of data sources and analytic requirements.

Unlike traditional BI reports that are designed and developed with a pre-defined, static set of requirements, Tableau visualizations are built for interactivity and collaboration. Users can ask any questions of their data without the need to write complex SQL Queries or joins. Users don't have to wait through another cycle of the traditional software development process to get their questions answered. Instead, they can iterate on existing visualizations and continue their analysis to answer questions about their business or project.

With traditional BI, you may be used to load-testing static reports that meet a specific service level agreement (SLA), where queries are designed to run against targeted systems with pre-built optimizations. A static report has a fixed scope, a fixed set of queries, and is often optimized by a developer, one at a time, over many weeks. While it is relatively easy to define and set an SLA for a static report, changing the report resets the entire development cycle, a limitation that is not typically factored into the SLA.

Tableau visualizations, on the other hand, regenerate and submit new queries on behalf of the user's exploratory actions. Local browser caching and optimizations in VizQL that enable quick retrieval of data can help the user stay in the flow of analytics instead of waiting for the results of a query.

VizQL as a language is encompassed into our VizQL server process and it works in concert with many other distributed server processes to provide a scalable, available, secure, and reliable business analytics platform.

Tableau Architecture

Tableau's flexible architecture is designed for scale up and scale out. It allows Tableau to be run as an enterprise standard platform or a platform for powering cloud analytics. Tableau Server is capable of supporting the most complex enterprise production infrastructure requirements as well as simple departmental or workgroup level deployments.

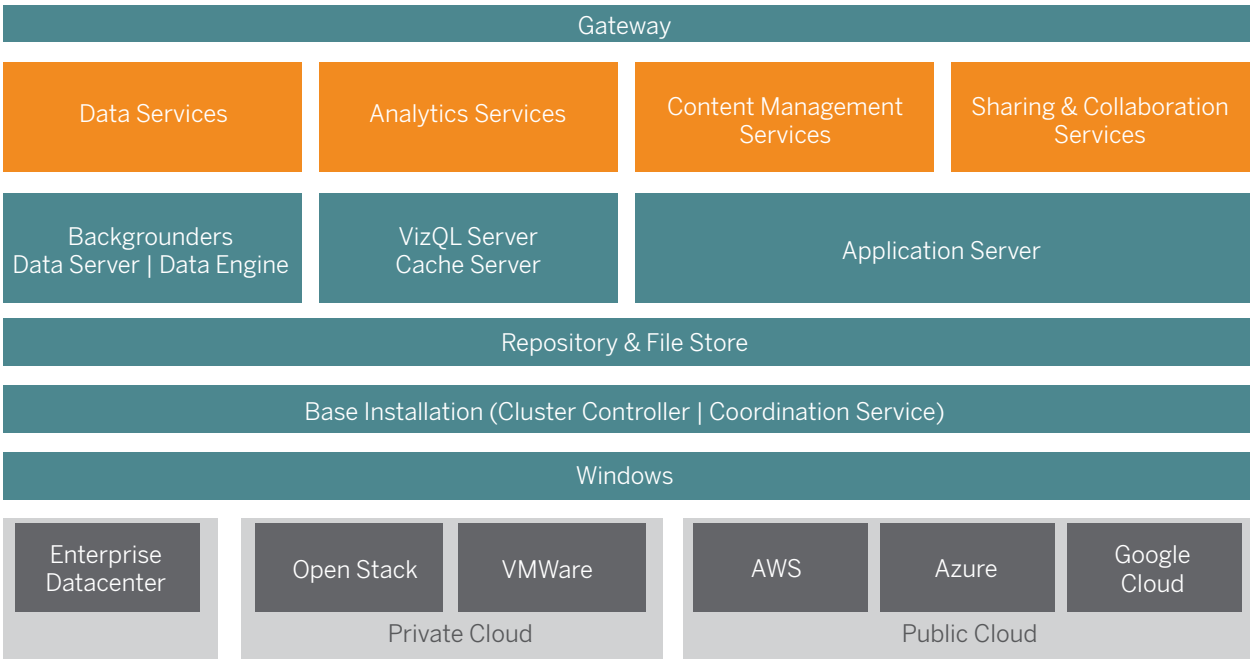

Figure 2: Tableau Server architecture

Tableau Server follows a simple installation and configuration process. Once installed, multiple server processes (shown in blue in Figure 2) work together to provide services at various tiers.

The Gateway process is the component that redirects traffic from all Tableau clients to the available server nodes in a cluster.

Data Services is a logical grouping of services that provide data freshness, shared meta data management, governed data sources, and in-memory data. The underlying processes that power Data Services are the Backgrounder, Data Server and Data Engine processes.

Analytics Services, composed of the VizQL and Cache Server processes, provide user-facing visualization and analytics services and caching services.

Content Management and Sharing and Collaboration Service are powered by the Application Server process. Core Tableau Server functionality such as user login, content management (projects, sites, security permissions, etc.) and administration activities are provided by the Application Server process.

All of the above services use and rely on the Repository process, which contains structured relational data like permissions, workbooks, data extracts, user info, and metadata. The File Store process enables data extract file redundancy across the cluster and ensures extracts are locally available on all cluster nodes. Under heavier loads, extract files are available locally across the cluster for faster processing and rendering.

Tableau's architecture is flexible, allowing you to run the platform anywhere. You can install Tableau Server on-premises, in your private cloud or data center, on Amazon EC2™, on Google Cloud Platform™, or on Microsoft Azure™. Tableau analytics platform can also run atop virtualization platforms such as VMware ESXi™ or Microsoft Hyper-V™. We recommend you follow the best practices for each virtualization platform to ensure the best performance from Tableau Server.

For details on individual server processes, please review the Tableau Server administration guide.

In-Memory and Live - Unified Architecture

Tableau provides true heterogeneous platform support, connecting to over 50 of the most popular data sources from dozens of vendors. The architecture provides flexibility for you to choose whether you want fast in-memory analytics or drive analytics by connecting directly to live data stores. Whichever path you choose, it is also easy to switch between live connectivity and in-memory analytics for varying business needs or data sources..

Tableau supports in-memory analytics by extracting data into a proprietary column store called a Tableau Data Extract (TDE) which is loaded into memory-mapped files for fast access. Extracts can be created by users on Tableau Desktop or from external business processes using Tableau APIs. Tableau Server architecture provides built-in support to ensure that the extracts your users create are updated and fresh on an optimized schedule that you control.

Regardless of how you connect to data and bring it into Tableau Server, you must ensure you have sufficient memory resources for data analysis, caching, extract refreshes, and other related operations.

Unlike pure in-memory tools, Tableau's memory footprint spans your entire cluster and isn't accumulated on a single server. Rather, memory use is distributed across the cluster via cache servers for externally shared query cache and other processes that store session level caches. The impact to memory of increasing loads is thus spread across the cluster according on your workloads.

Testing Approach & Methodology

Unlike most traditional load testing projects, where the target application is treated as a black box, Tableau Server should be load-tested with sufficient understanding of its architecture. We built Tableau Server to run anywhere: on-premise or in the cloud. It's designed to service teams and organizations of any size. Therefore, the default install of Tableau Server should work well for most standard deployments. As you look to scale and deploy Tableau Server across your enterprise, you must understand how different workloads are processed and how a few simple configuration tweaks may improve results for your deployment scenario.

For Version 10 scalability goals, we set out to answer the following questions:

1. Does Tableau Server 10 scale linearly by adding more hardware for two common scenarios:
   - Scaling end user workload
   - Scaling Backgrounder workload

2. When is it a good time to move the Backgrounder process(es) to a dedicated worker node?

3. How does Tableau Server 10 performance compare with previous releases?

Previous versions of this whitepaper focused specifically on end user scalability, but many customers asked us about how to scale the Backgrounder workload. The Backgrounder workloads control how fresh the data is (extract refreshes) and how widely analytics is consumed (subscription notifications).

Methodology

Improving performance and scale were primary goals for the version 10 release of Tableau Server. As a result, developing a production-ready, enterprise-class testing methodology was a core requirement as well. We began running the methodology described in this section on pre-release versions of v10. During the iterative process of agile development, we discovered and fixed nearly two dozen scale and performance-specific bugs exposed by the methodology. We continued to test after the v10 release and fixed bugs into the 10.0.1 maintenance release (MR). This whitepaper refers generally to the v10 release of Tableau Server throughout. However, the results of testing are based on the 10.0.1 MR. To realize the best scale and performance benefits of v10, you should run the 10.0.1 MR or later in your organization.

We have continually evolved our scalability practices to gather and test for workloads that represent realistic customer scenarios. While there are many variables that inform scalability of a deployment, the important factors to consider as you plan your deployment are:

- User Impact – self-service usage and user adoption: How many users will be using analytics? How often will users employ analytics to make an informed decision? How complex are visualizations that users are creating?

- Data Impact - freshness, size and location: How big is your data? Where is the data located? How fresh does the data need to be to accurately inform business decisions?

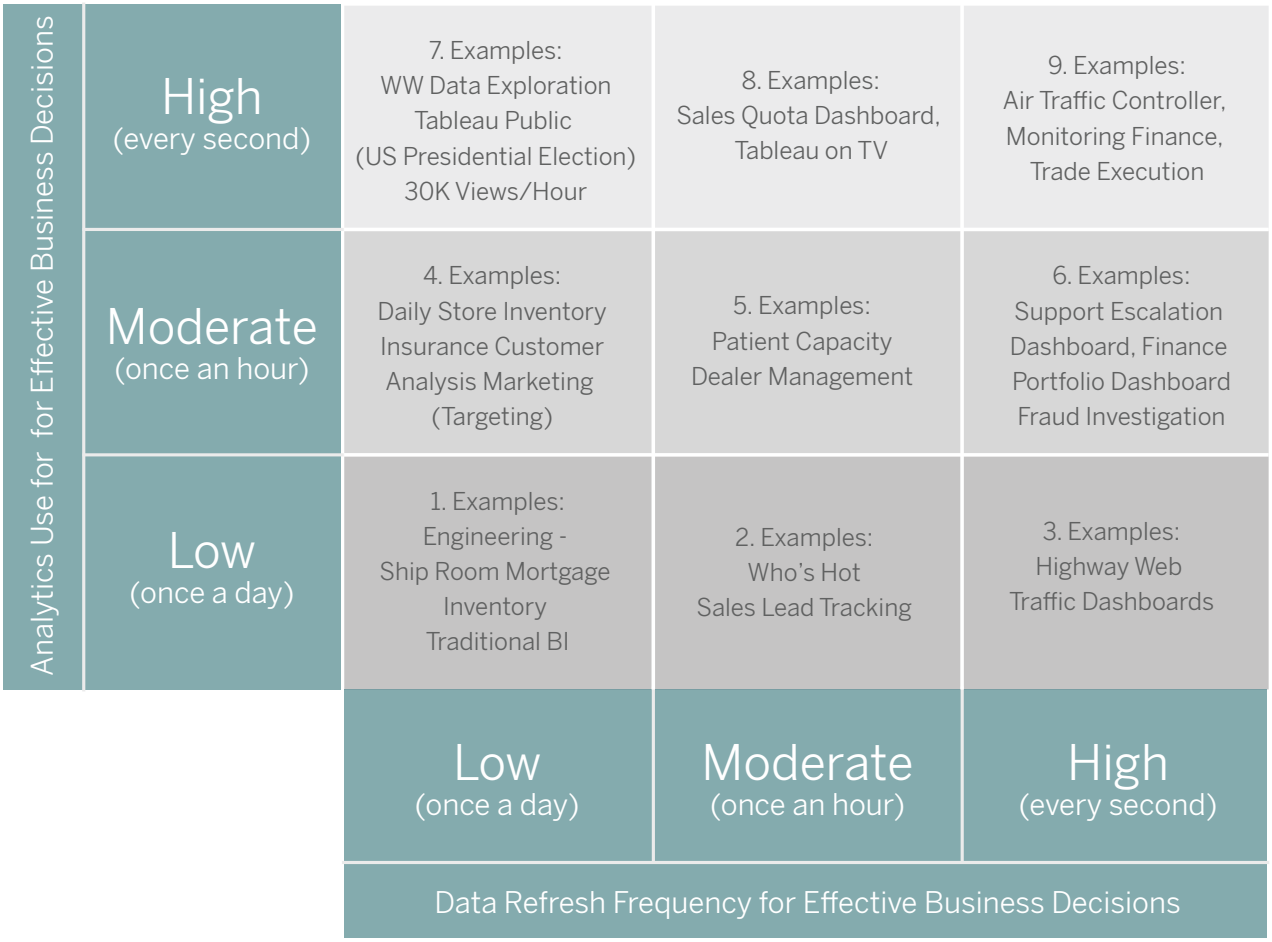

Figure 3: Analytics use and data refresh frequency matrix

To test for both of these considerations, we needed to incorporate tests where we incrementally loaded a server with more end users while increasing Backgrounder workloads. This methodology allowed us to study the impact of the Backgrounder workloads on end users' quality of service. To model low, moderate, and heavy use of analytics and the impact of isolating the Backgrounder workloads, we ran over 400 iterations of tests in our labs across various workloads. We studied system scalability and we also fixed bugs that only manifested under heavy loads.

To make the tests relevant in the real world, we first had to gather the appropriate set of workloads to run these tests.

Real workload characterization

For v10, we observed and characterized how a real production Tableau Server was being used during periods of peak utilization.

To determine which workbooks to model and the workload characteristics for our testing, we analyzed Tableau Server log files from a production environment of 3000 users. We identified the visualizations and workbooks that were heavily used. We calculated the usage distribution across these workbooks and then analyzed usage characteristics. For example, we investigated the time gap between requests (aka think time). Below are the specific steps we took to model the workload from the production server.

Test Modeling Steps

1. Get the production server logs from peak usage period

2. Identify the top N workbook-views by weighted average time taken on server.

3. Weighted Average = Average Response Time * Number of requests

4. Calculate the relative weights among the top N workbook-views

5. For each selected workbook-view
   a. Find percentage of visualization loads with: refresh=y (this was a way to find out how many users were actively refreshing their data to get fresh data)
   b. Find the top N interactions by weight
   c. Calculate average think time between interactions

6. For model verification:
   a. Find average time between bootstraps TBB (time between tests) for the workbook-view
   b Find the top N Backgrounder tasks by weighted average time-taken on server

7. Categorize the Backgrounder traffic by different tasks (e.g., subscriptions and extract refreshes)
   a. Find the average time between top Backgrounder tasks and include that in the model
   b. Find the number of subscriptions across various schedules
   c. Find the size and type of extract refreshes (published extract to data server, workbook extract)

We used all of the above data to model a realistic workload mix that represented how real users were using the production server during peak usage. Finally, we generated a workload-based model for Backgrounder extract refreshes and subscriptions from our production log analysis.

The workload mixes are summarized in the following table.

Workload Name	Description	How to compare to previous versions
Real Production Server Workload	This workload was based on analyzing and characterizing usage of a production Tableau Server that services 3000 users in an organization as a mission critical application managed by IT.	Results data from only this whitepaper are comparable between 9.3 and 10.0 as specified here. The results are not comparable to previous whitepapers we've published—including the 9.0 whitepaper—since testing methodology was significantly different.
Real Production Server Workload + New Features	This workload combined the above workload with workbooks that exercised the new features in v10.	Not comparable across 9.3 and 10.0 because 9.3 does not have the v10 features to exercise. The best use of the results in this whitepaper should be to inform v10 scaling characteristics.
Backgrounder Mix	This workload is based on production workload analysis and models real workbooks and schedules mirroring the production environment.	New mix introduced in 10.0 scalability testing whitepaper Not comparable to any prior whitepaper results.

Table 2: Workload mix descriptions

We then took each of these mixes and ran them independently in an isolated scalability lab on physical machines with increasingly higher end user and backgrounder loads. Once the cluster was at capacity, we continued scaling the load after adding one worker node at a time. We observed how the system behaved during each of these experiments. Through each run we recorded the key performance indicators, like response time, throughput, and error rates. We also recorded system metrics and application server metrics using JMX. For each of the runs, we correlated the data and analyzed how the system behaved under increasing workloads. Along this process, we also found and fixed scalability bugs as part of our agile development process.

Backgrounder Methodology

The Backgrounder server processes system-level and user-level background jobs. System-level jobs, such as routine repository maintenance tasks, are performed by the Backgrounder. User-level jobs are those that a user may have submitted for the system to run on behalf of the user. For example, users can publish extracts to the server and then configure a recurring data refresh for the extract according to a schedule. This set of operations creates a refresh job. The Backgrounder is the process that will review the jobs list and execute the jobs on behalf of the user. This scenario is critical to effective self-service because the user does not have to wait for an administrative department to refresh the data. However, if the administration team managing the Tableau Server does not plan for capacity for this type of load,

quality of service issues may result for the end user. How much you need to optimize for Backgrounder services is a critical component of server sizing and planning. You should consider whether to separate the Background services on another computer to isolate the workloads.

There are several simple best practices that are discussed toward the end of the paper that allow you to separate workload processing by time of day. However, if you are running the Backgrounder on the same computer as the Analytical Services, you may see an impact on the end user quality of service under heavy loads due to resource sharing constraints on the server.

For this reason, we wanted to study what impact the Backgrounder has on end user scaling when it is co-located on the same computer as the Analytics Services. In addition, we wanted to quantify how the Backgrounder scales with increasing loads when it was isolated on its own hardware.

To study this, we used a computer with four physical cores to run the Backgrounder in isolation. We did not run any other Tableau Server processes on the same computer. We ran the same production modeled workload on versions 9.3 and 10.0 of Tableau Server. The workload included extract refreshes and subscriptions so we could focus on user-level jobs. The workload included 400 subscriptions across 8 schedules. We studied the success of subscription notifications and also the amount of time it took for Tableau to complete all the subscriptions.

Standardized Isolated Environment

We ran the scalability tests in our performance lab on identical physical machines with the following specifications.

Server Type	Dell PowerEdge R620
Operating System	Microsoft Windows Server 2012 R2 Standard 64 Bit
CPU	2.6 GHz 1x8 physical cores, hyper-threading enabled (16 logical cores)
Memory	64 GB

Table 3: Hardware specification of each node in the testing environment

Although this table lists physical cores, we recommend that you don't disable hyper-threading. For consistency, with the exception of the reference here, we refer to physical core counts in this whitepaper and always assume hyper-threading is enabled on physical machines.

Deployment Topology

Clusters are made up of one or more primary (controller) nodes and one or more worker nodes. In our testing, the worker nodes shared the same process configuration profile:

Process	Primary tsperf-212.perf.dev.tsi.lan	Worker 1 tsperf-213.perf.dev.tsi.lan	Worker 2 tsperf-215.perf.dev.tsi.lan	Worker 3 tsperf-216.perf.dev.tsi.lan	Worker 4 tsperf-217.perf.dev.tsi.lan
Cluster Controller	✓	✓	✓	✓	✓
Gateway	✓	✓	✓	✓	✓
Application Server		✓	✓	✓	✓
VizQL Server		✓ ✓	✓ ✓	✓ ✓	✓ ✓
Cache Server		✓ ✓	✓ ✓	✓ ✓	✓ ✓
Search & Browse		✓	✓	✓	✓
Backgrounder		✓	✓	✓	✓
Data Server		✓ ✓	✓ ✓	✓ ✓	✓ ✓
Data Engine		✓	✓	✓	✓
File Store		⟳ Synchronizing	⟳ Synchronizing	⟳ Synchronizing	⟳ Synchronizing
Repository	✓				

Refresh Status ✓ Active ⟳ Busy ✓ Passive ⚠ Unlicensed ✕ Down ☐ Status unavilable

Figure 4: Process configuration

The worker process configuration shown here is the default configuration. You may get more or less scalability depending on the number and types of processes you configure for your environment and usage scenario.

Primary nodes are configured with a base installation that includes Cluster Controller, Gateway, and Repository processes. It's worth noting that when deployed in a core licensing scheme, this type of primary node configuration is not included in the core count.

We scaled the workload using load generators (TabJolt) to simulate the user workload described previously. TabJolt is a "point-and-run" load and performance-testing tool specifically designed to work with Tableau Server 9.0 or later. The figure below shows a logical view of the test execution.

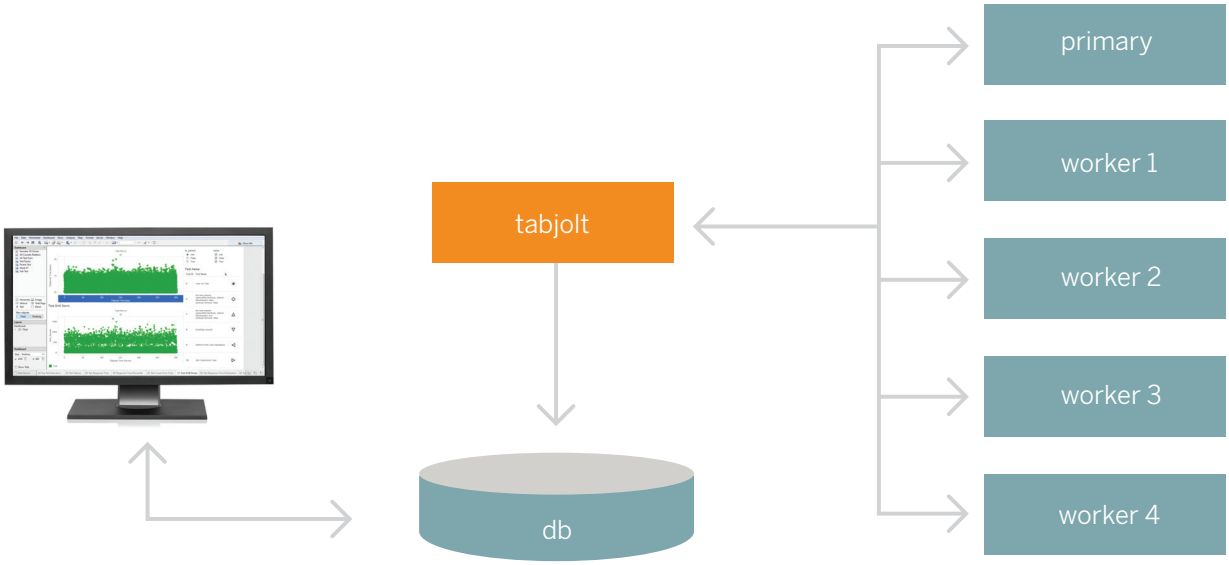

Figure 5: The logical view of the test environment

We collected data from each of the test iterations to analyze, but before we jump into the results, let's understand some of the metrics and the definitions.

Measurement & Reporting

We measured a number of metrics to understand hardware performance and scalability, including system metrics for CPU, memory, disk. We also measured performance and scalability metrics such as response times, throughput, error rates, run duration, and others.

To understand the data discussed in this whitepaper, let's quickly review some definitions.

Scenario

Scenario is the top level user activity on the server. In the prior versions of the paper, we were focused on visualization load and interact times when server had guest access enabled. In this release of the paper, we have expanded the workloads to include Application Server Services (like login) and other services. The end users go through a series of steps based on the production workload modeling which we simulated using a customized version of TabJolt.

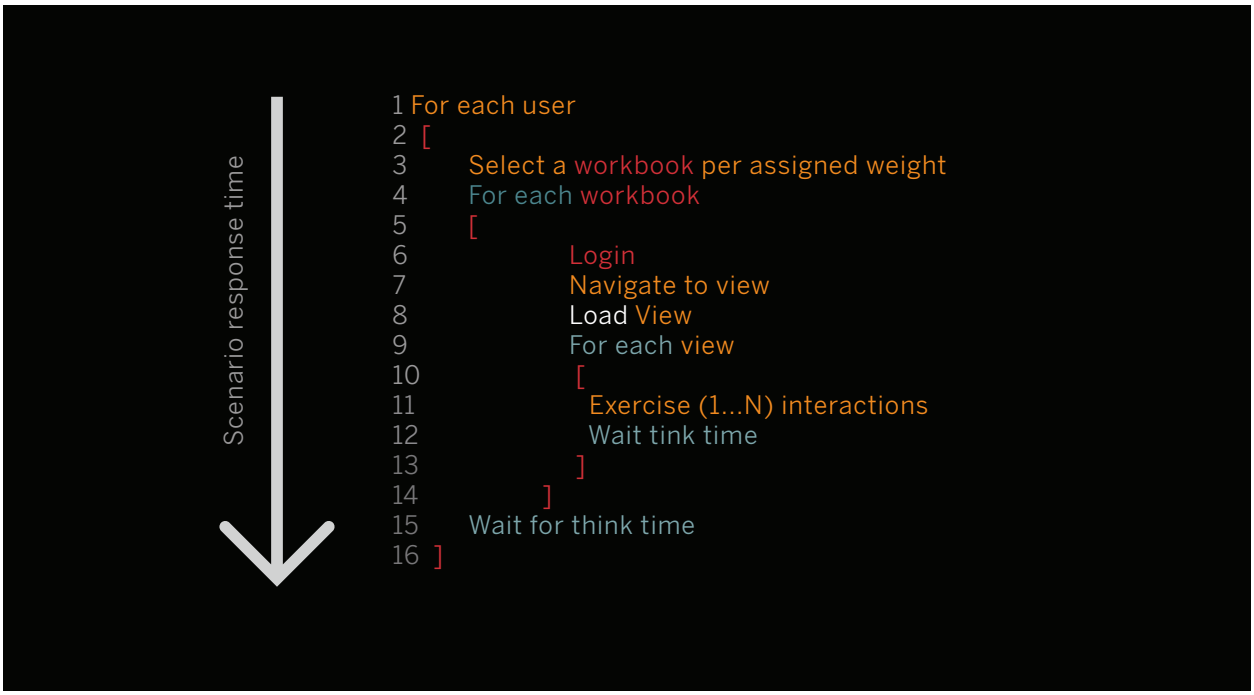

Figure 6: Scenario testing algorithm

The reported response times in this paper are larger than more simple load and interaction models because our test models include the time required to complete the scenario. The response time is measured and reported for a transaction from the client's perspective. This means the scenario describes the actual end user's perspective, including environmental network variables such as latency.

Response time

Response time is measured as the number of seconds it takes the server to respond to the end user request. Consider this example: a user signs in to server, navigates to a visualization, changes a filter on that visualization, waits for the visualization to update and render, then analyses the visualization (think time). The total "wall clock" time from the time the user logins to server to the end of the user think time is reported as a response time for that iteration of the scenario.

Scenario throughput

Scenario throughput is the number of successful scenarios completed per second (TPS). We calculated the scenario TPS by running scenarios for an hour on a given topology at the computational limit of the system. The total number of scenarios divided by the number of seconds in an hour (3600) is the TPS.

For example running our scenarios on an 8 core Tableau Server 10 for the duration of the test run completed 16,372 scenarios. This translates to a TPS of about 4.5 (16,372/3600). These experiments pushed the server to far higher loads than what we observed in production, where the TPS was < 1.

The significant increase in experimental data is because we are pushing the server to its limits trying to understand ultimate performance capacity for the chosen hardware and scale units.

However, production deployments experiences load bursts during peak usage and users are generally a lot slower than an automatic test execution framework. Keeping in mind the difference, Tableau Public, for example, delivers massive scales of approximately ~12 visualization loads (referred to as impressions) per second which translates to over 7+million impressions of Tableau Visualizations per week.

Active users

Active users is a metric that measures the number of users that are simultaneously using Tableau Server in a peak one-hour time window. Scenarios now include log-in, visualization load, user interactions, search, and other actions. We define active users as users that are performing any of these sorts of actions during a peak one-hour time window.

To find out how many active users our installation of server would support, we started by determining how many users a single server could sustain without degradation in response time, more than a 2% error rate (Tableau Server HTTP errors), or greater than 80% CPU utilization.

Based on our production workload characterization, we assigned weights to specific workbooks. We then spawned a thread (a virtual user) which would select a random workbook based on the weights assigned to it, and then complete the entire scenario described earlier. At the end of the scenario, the thread would wait for a specified think time. At the end of this period, the thread would complete its iteration and exit. During the test, we measured the scenario throughput, response time, error rates, CPU and memory usage amongst other metrics. We kept increasing the number of active threads as long as the CPU remained below 80%, the error rate was below 2% and the response time had not degraded. When one of these thresholds was breached, we considered that the sweet spot for the number of active users for that topology would reasonably support.

To validate that server scales linearly, we found the sweet spot for a single machine, then linearly increased the number of virtual users to the next increment and validated that our preset conditions were still met under the new load.

Results

Now that we've seen how we perform test execution, the deployment we used, and the metrics, let's review the results.

Tableau Server 10 Scales Linearly

Our first question was how does Tableau Server 10 scale? With increasing user loads, we observed that Tableau Server 10 scales linearly with load by adding more worker nodes to the cluster. The figure below shows the number of user scenarios that were completed per second with increasing workers.

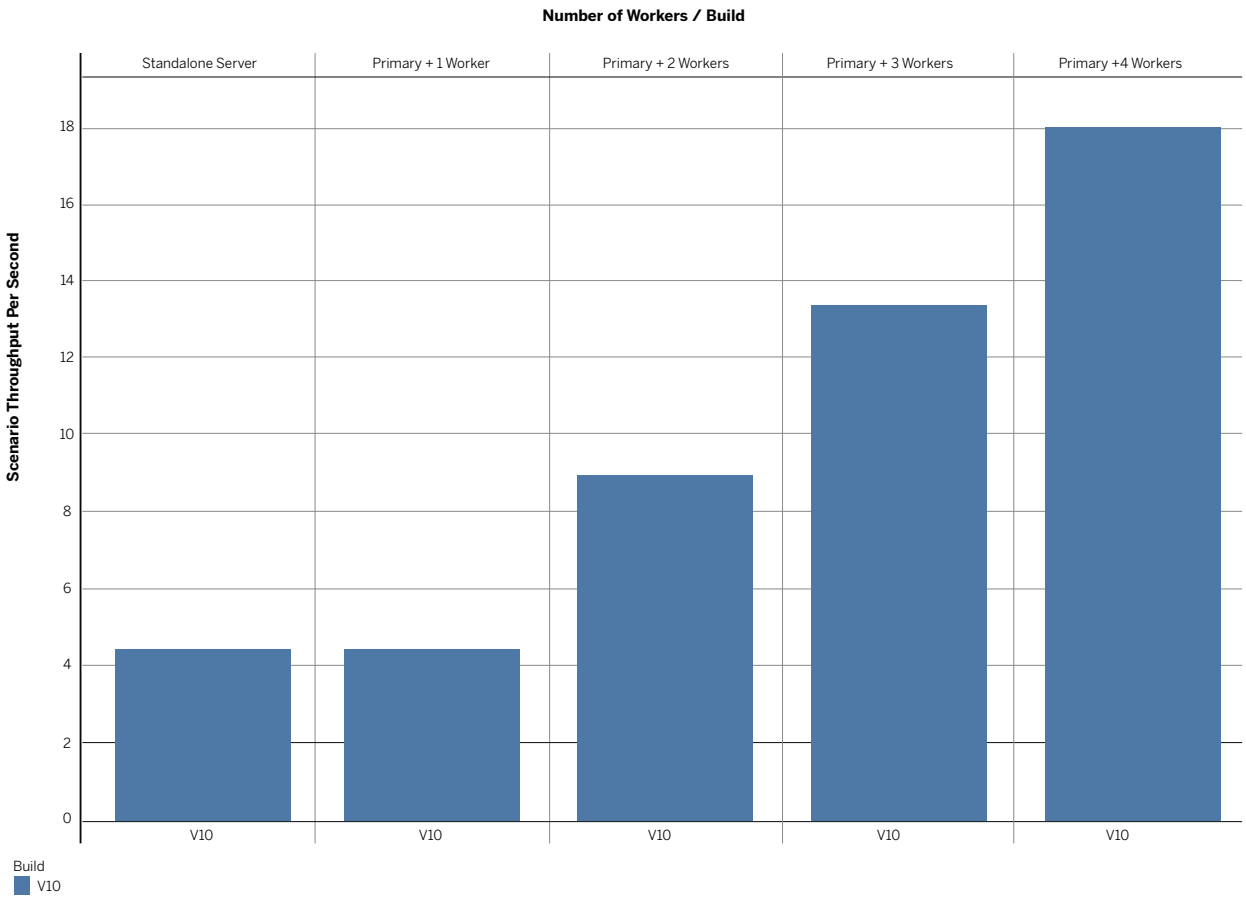

Figure 7: Scenario throughput per second

Each column shows the cluster topology. The further-most left column shows the single-server setup. The second column shows the primary/worker cluster configuration with a single worker. For each remaining column we add a single worker node, as described earlier. The height of the bar shows the average scenario throughput per second (TPS). The TPS represents the amount of work the server is taking on. As shown, TPS increased linearly as we added more worker nodes. We observed that as we increased the loads on the cluster, the CPU utilization across the cluster was averaging about 80% The figure below shows the CPU utilization across the cluster with increasing loads with a 95% confidence band.

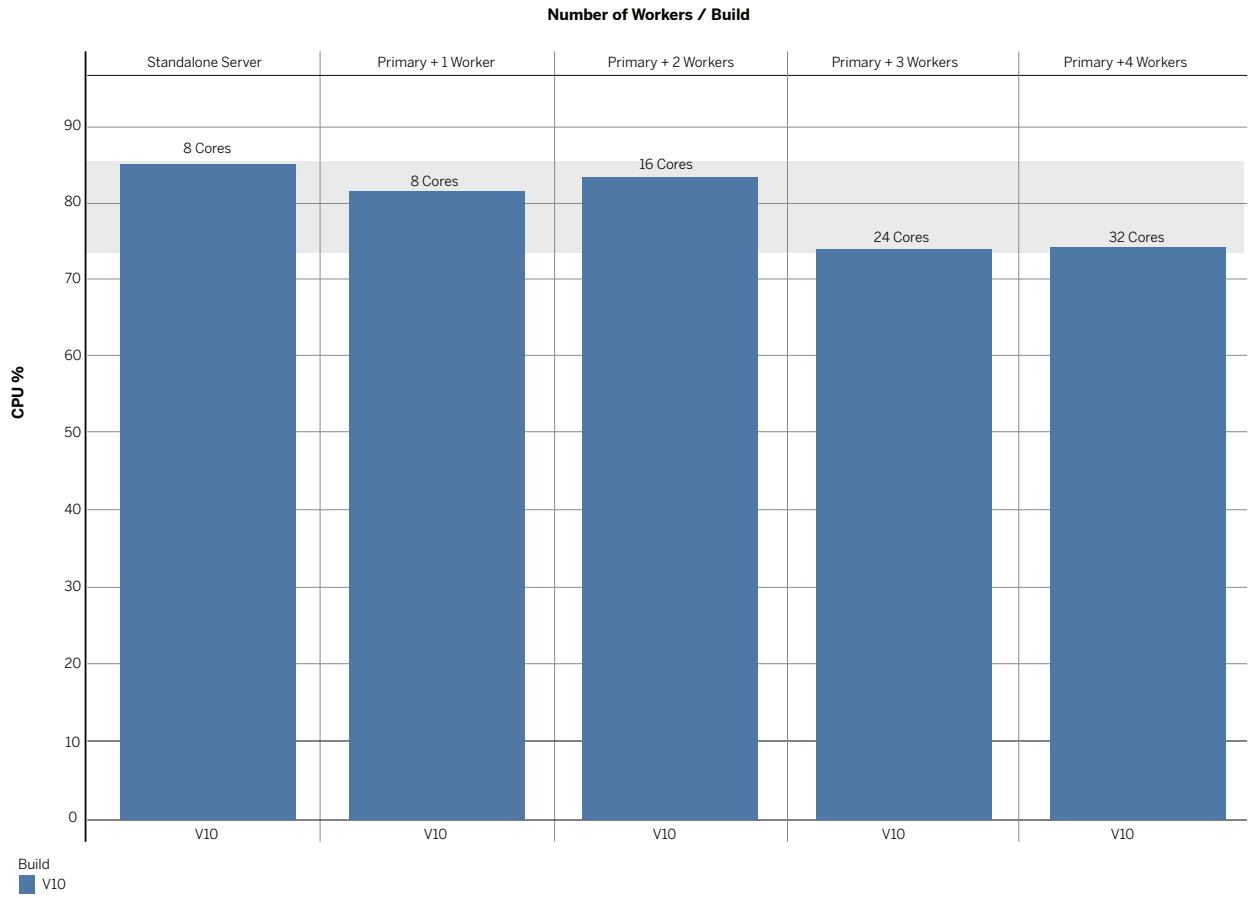

Figure 8: CPU utilization with increasing loads across the cluster

As Figure 8 shows, adding more workers to distribute CPU load across the cluster optimizes the system by providing some headroom for load bursts. In the case where fewer worker nodes were configured in a given cluster, the CPU utilization was comparatively higher than clusters with more workers. In these lower-worker node cases, compute bound processes are competing for the limited resources. During these tests, we observed the error rates from the server and they were well within the 2% goal we had set out as part of the methodology. Depending on your workloads, you may see higher error rates (lower quality of service) when clusters are constrained to fewer machines and/or are limited in capacity.

Our next question was how does Tableau Server 10 compare to Tableau Server 9.3 with the same testing methodology?

The results: under the same methodology run on each the version, Tableau Server 10 throughput improved when compared with Tableau Server 9.3.

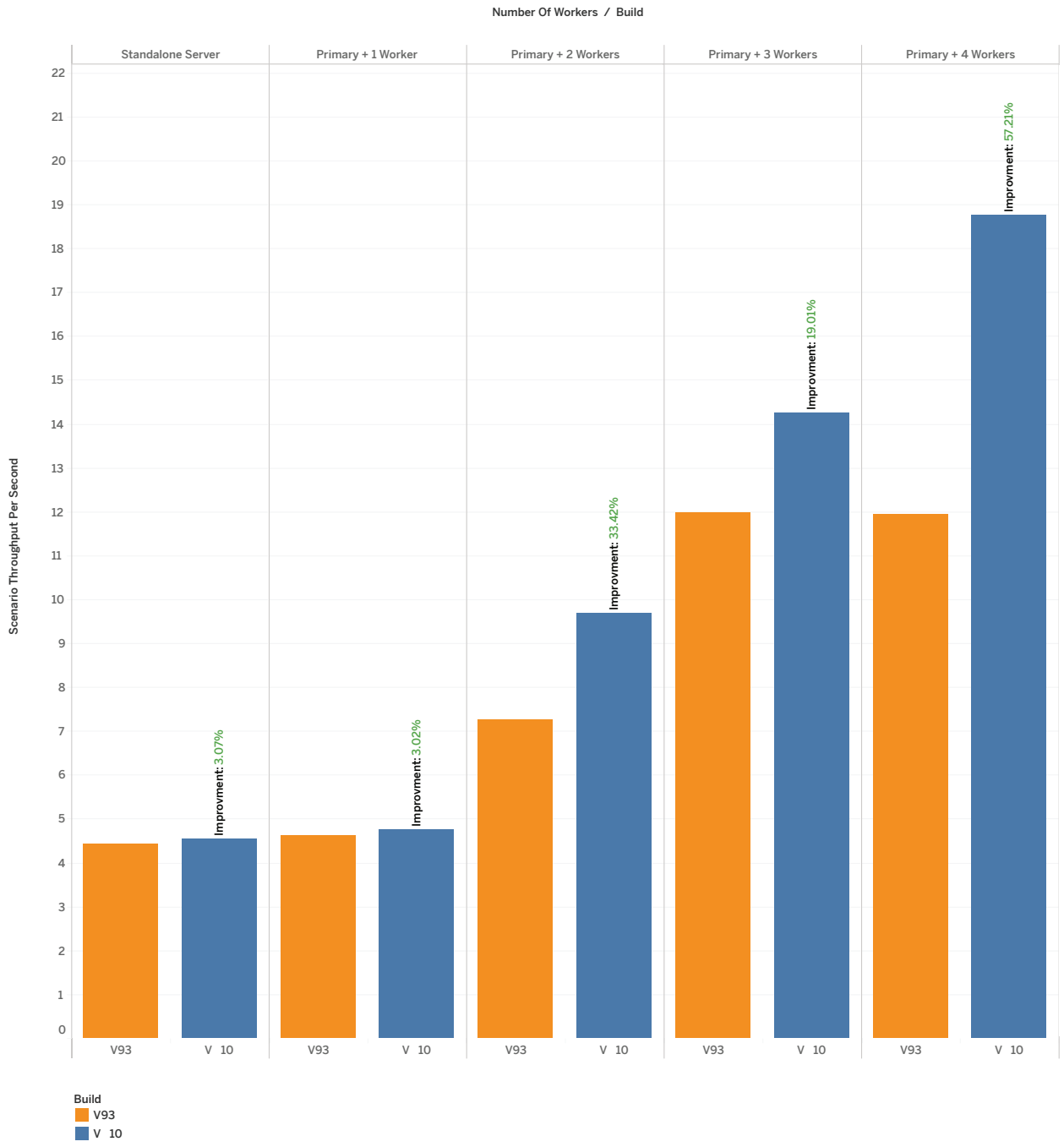

Figure 9: Comparing Scenario Throughput per second 9.3 vs 10.0.1

In Figure 9, scenario throughput for Tableau Server 9.3 (orange bars) is compared to the scenario throughput for Tableau Server 10 (blue bars). Each pane shows the configuration of the cluster topology as described by the column header. As noted earlier, when we added more nodes to the cluster, Tableau Server 10 not only scaled linearly it also scaled scenario throughput better when compared with Tableau Server 9.3.

As you compare versions, understand that the only appropriate comparison we can make is by comparing the two versions with the improved testing methodology that captures real-world user scenarios. For this reason, comparing Tableau Server 10 results to our prior whitepaper results is not an accurate comparison as the testing methodology has changed significantly.

While it's important to observe throughput, we wanted to make sure the end user response time for the entire test scenario was adequately performant and not failing under increased loads.

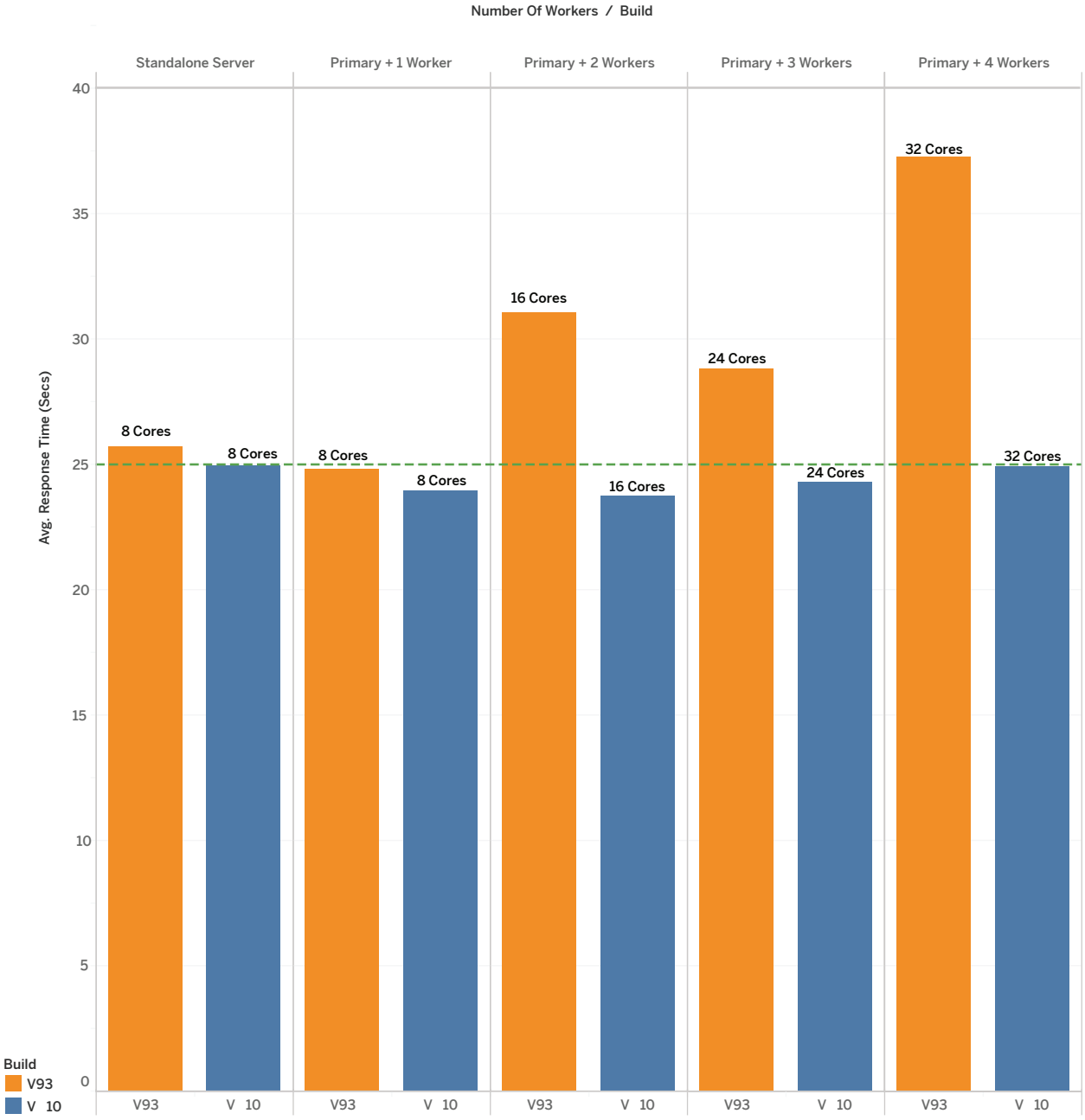

Figure 10: Comparing average scenario response times (seconds): 9.3 vs 10.0.1

In the Figure 10, the height of the orange bar shows the average scenario response time for version 9.3. The height of the blue bar shows the same metric for version 10.0.1. As we increased the end user loads on the cluster hosting version 9.3, we observed incremental increases in response times. Tableau Server 10, on the other hand, showed a more consistent response time as we continued to add virtual users to the system. This shows that Tableau Server 10 delivers better performance and responsiveness to end users relative to Tableau Server 9.3 for the same workloads.

Some of the improvements we observed in Tableau Server 10, with relatively consistent and stable response times under increasing loads, were the result of improvements we made to the cache components. Specifically, we did work in v10 to optimize caching for the bootstrap response scenario. The bootstrap scenario is the first call made to initialize and cache data for a user session. Without caching, earlier version of Tableau Server would compute and save the bootstrap data for every subsequent request, even if they were very similar or identical. In version 10, we perform smart caching operations for this scenario. These improvements made Tableau Server 10 more efficient by allowing it to process more user load while maintaining good response times. However, the server is now doing more work, as we observed earlier in the increased throughput.

All of the results above were observations on how Tableau Server scaled with increasing use of analytics by end users. As more end user workload increases on the server, you should ensure you have sufficient capacity by adding more worker nodes to your cluster to deliver a good quality of service experience to the end users.

The next set of questions centralized around user impact as a result of where and how Backgrounder is configured. Specifically, what is user impact of hosting Backgrounder processes on the same computer as the Analytics Services (VizQL Server) vs isolating Backgrounder processes on a different computer. We discuss those results in the section that follows.

Backgrounder Results

First, we wanted to quantify the impact of running the Backgrounder in our default single-server installation. What is the impact of this scenario on the end users and scalability in general? We ran our workload experiments and recorded the TPS on a single Tableau Server deployment. We then tested workloads on clusters, building out the cluster topology by adding workers.

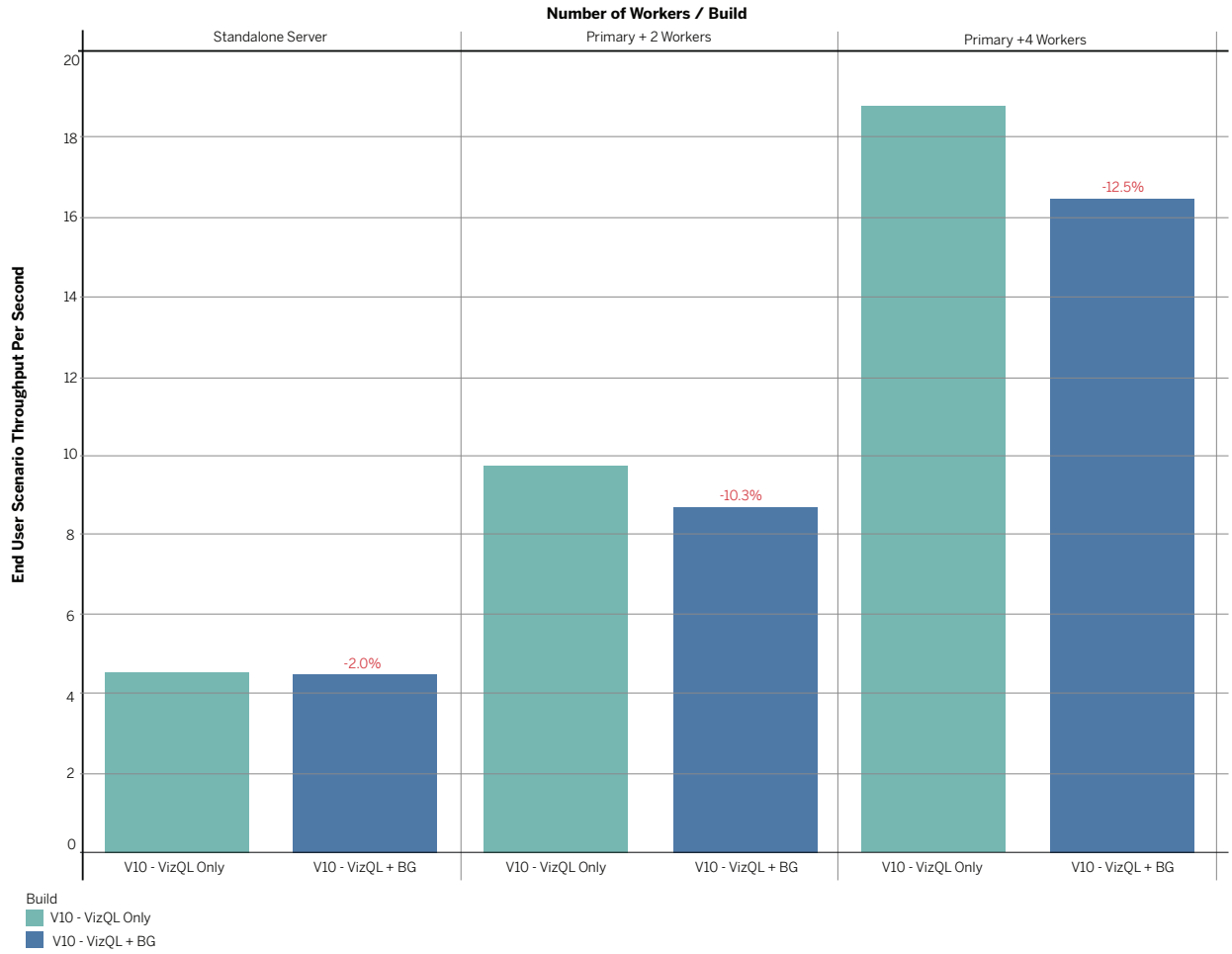

Figure 11: Impact to end user scalability of co-located Backgrounder and VizQL services

Figure 11 shows the results of two testing scenarios. The green bars represent the test scenario where we ran a VizQL-only workload. The VizQL-only workload simulates end user workloads. The blue bars represent the test scenario where we then added a fixed amount of backgrounder workload to the VizQL workload on the same cluster. We then recorded the changes in the TPS for the end users' workload across the two scenarios. We knew this would have an impact because both VizQL Server and Backgrounder are compute intensive workloads, but our intent was to measure this impact for the specific workloads.

We observed between 2-12% reduction in the overall end user scenario throughput. This reduction shows that the ability of the cluster to service end users is impacted by the cost of processing the backgrounder workload. Extrapolating this data shows that if the cluster was servicing 100 end user scenarios per unit of time, then the addition of the constant backgrounder load reduces capacity to 88 user scenarios in the same unit of time. This reduction in throughput could translate to significant impact to end user scalability and/or the quality of service depending on several factors such as the workloads, peak load bursts, hardware limitations, and infrastructure variables.

This test illustrates the importance of properly resourcing hardware according to Tableau Server workload planning. Running Tableau Server on under-powered or constrained hardware may result in reduced throughputs, failed backgrounder jobs, delays in subscription notifications and/or end user errors manifesting as performance issues. When such conditions occur, consider expanding the cluster to isolate backgrounder workloads on dedicated hardware. By isolating the Backgrounder service, you will free up competing processes that are compute bound that would otherwise be vying for the same resources when co-located on the same computer.

Let's review that impact on a specific workload for subscriptions when backgrounder is co-located with VizQL server and when it's isolated on its own machine.

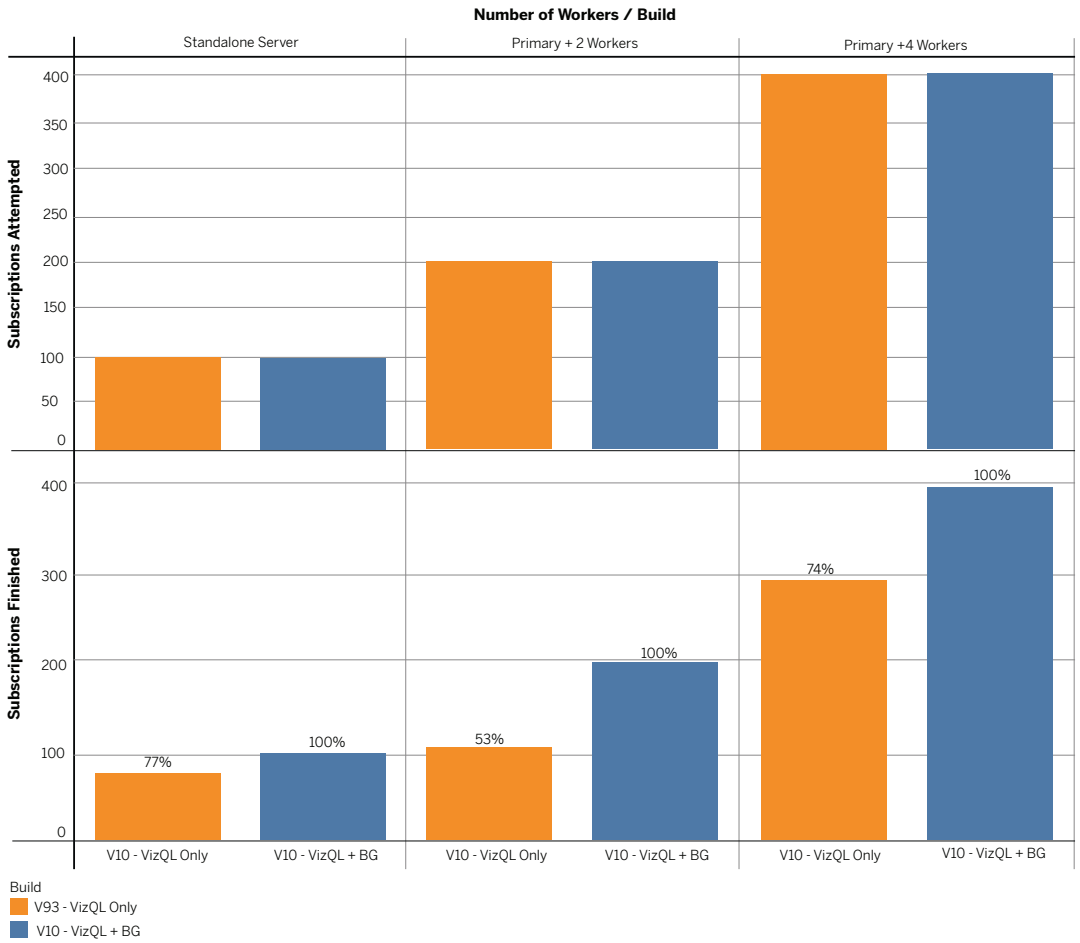

Figure 12: Improvements in subscription notification

In Figure 12 each pane shows the Tableau Server cluster topology in 9.3 (orange) and 10 (blue) while each bar represents the number of subscriptions notifications attempted and the number of notifications completed successfully. In each iteration, we increased the number of worker nodes in steps of 0, 2, 4. and each worker node had only one Backgrounder process configured. (Omitting 1 and 3 worker testing did not compromise the fidelity of results we sought.) We limited the Backgrounder process to one per worker node to establish a consistent measurement for comparisons.

As we increased the backgrounder subscription load, Tableau Server 10 was able to complete all the work that was submitted. In the same tests, Tableau Server 9.3 would begin to saturate and some subscriptions failed. This behavior could be exacerbated in an under-resourced and over-loaded topology. In addition, we observed that Tableau Server 10 could take on increasing number of subscriptions as more workers were added to the system. An improvement in Tableau Server 10 that explains the above observation has to do with the introduction of image caching for notifications. This feature allows the Backgrounder to do much less work for the same tasks tied a schedule. In scenarios where the same work is being executed on the same schedule, Tableau Server 10 now caches the results from the first execution and serves the results to subsequent requests for the same workload. This means the same work is completed more efficiently.

For the subscriptions that use the same workbooks on same schedule, we saw the time needed to complete the subscription was reduced between 60-90% compared to 9.3. Figure 13 illustrates this improvement.

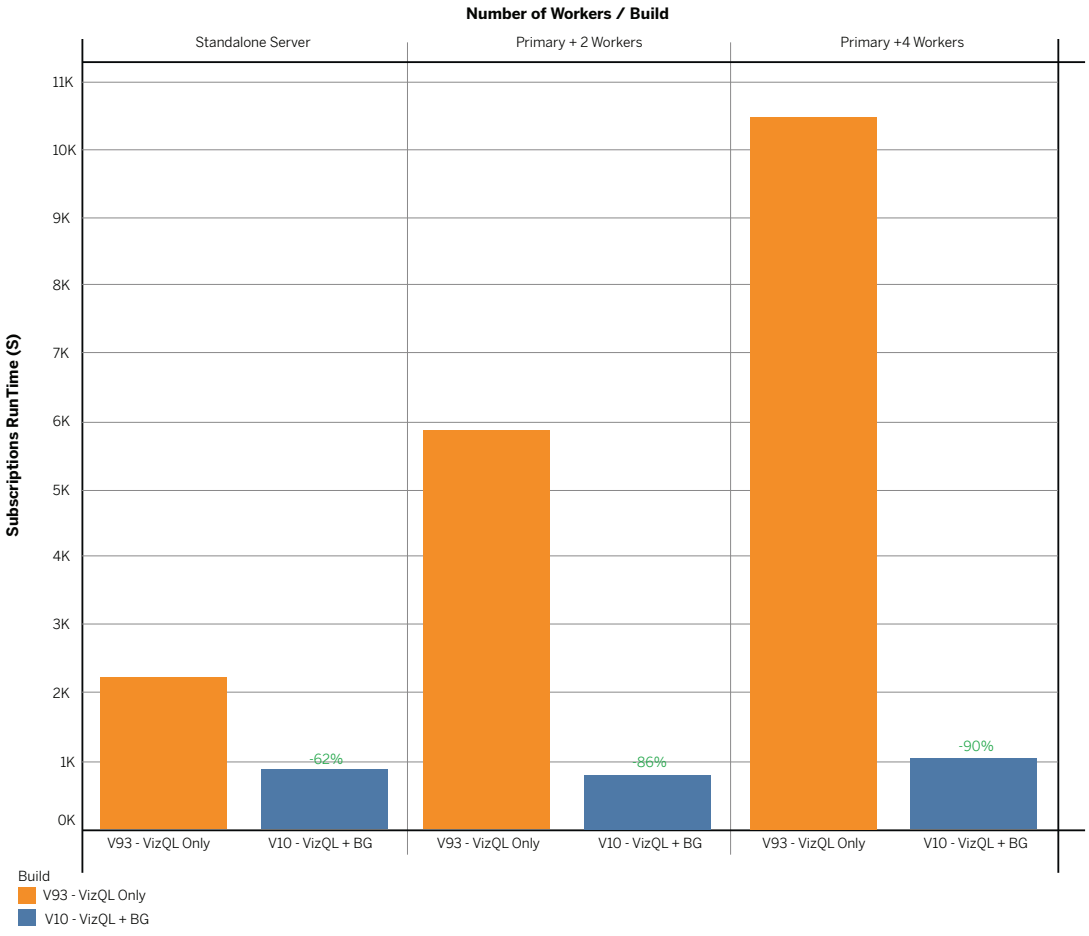

Figure 13: Subscription run time (time to completion): 9.3 vs 10

What this means, is that any notifications to the same workbook on the same schedule will take much less time to complete. However, if you have notifications to different workbooks, or user filters are set, the subscriptions will take processing and compute time before the notification can be delivered to the user. Different workbooks and user filters require that Tableau Server runs the entire end user visualization pipeline, which includes data query and data visual processes for each workbook or filtered view.

The benefit of subscriptions is that they provide business users with the data they care about in a timely manner. Effective extract refreshes help your organization make good decisions based on appropriately fresh data. Since the Backgrounder process manages both of these critical functions, planning your subscriptions schedules such that duplicate work is tied to the same schedule can ensure benefits of the cache.

Isolating the Backgrounder process

With the Backgrounder process isolated on its own worker node, we ran the same subscription experiments. We observed that the Backgrounder worker node was able to complete 400 subscriptions on a single 4 core machine. This is same number of subscriptions that we recorded when a single Backgrounder process was co-located with each VizQL worker across 4 worker machines.

The important lesson here is that while the Backgrounder process itself scales, isolating the process delivers similar scale but does not impact the end user quality of service. The Backgrounder process is single-threaded and is designed to complete jobs as quickly as possible. Given this design, a Backgrounder process will consume a full core when it has work to do. On an isolated machine, the aggressive compute usage of the Backgrounder process will not interfere with other user-facing Tableau services.

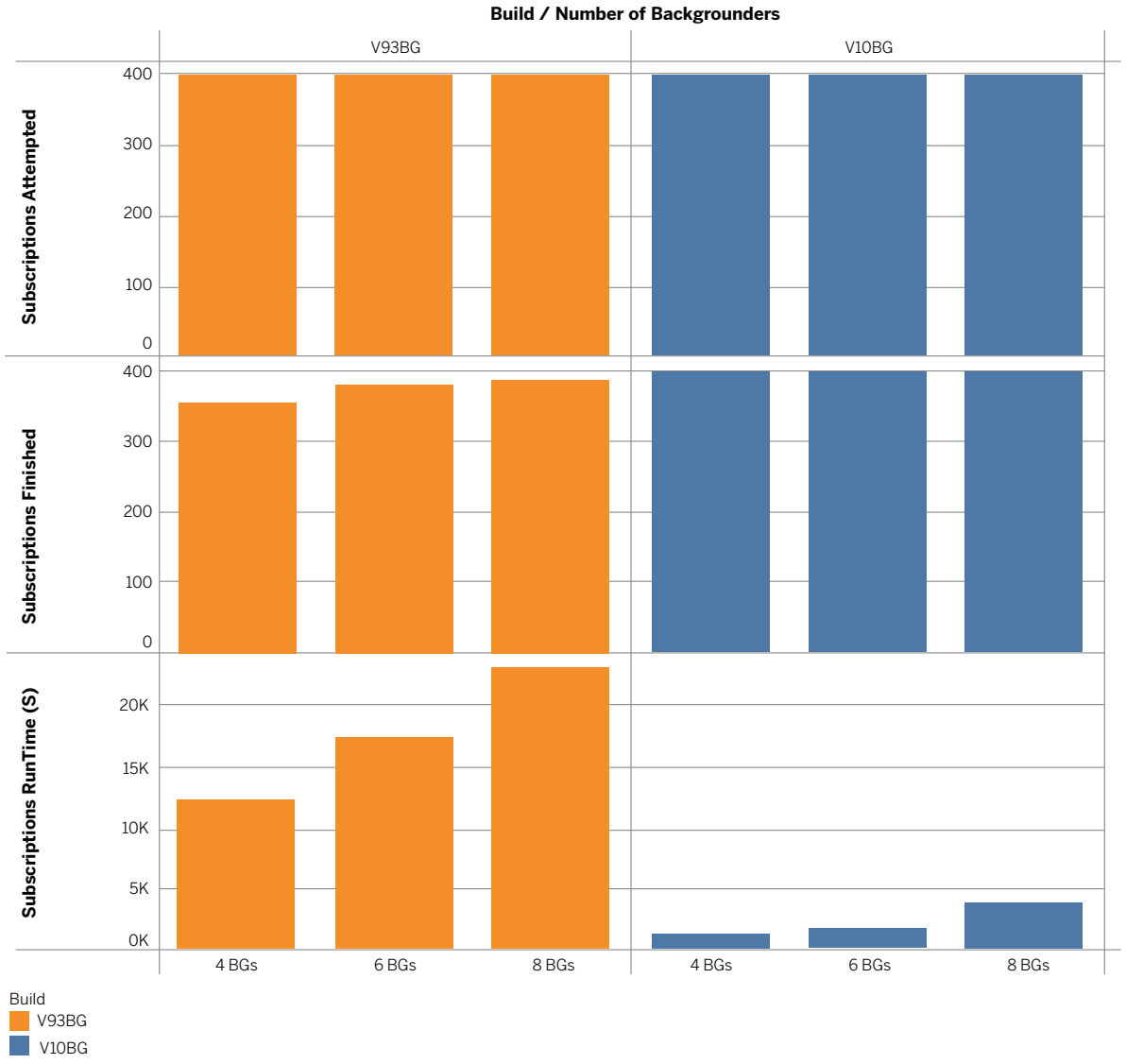

Figure 14: Adding Backgrounder processes to a 4 core computer: 9.3 vs 10

As shown in Figure 14, adding Backgrounder processes to a single 4-core computer enabled Tableau Server 10 (blue) to complete subscriptions in significantly less time when compared to 9.3 (orange). However, continuing to add Backgrounder processes on a computer that is physically limited by cores has a negative impact. As shown, adding 8 Backgrounder processes to a computer bound by 4 cores slows completion time. This is due to the single-threaded design of the Backgrounder.

Lastly, in our experiments where we tested extract refreshes on worker nodes running isolated Backgrounder processes, we observed that Server 9.3 and Server 10 were comparable. Both versions completed the same number of extract refreshes in about the same amount of time. An important detail to consider as you scale for extract refreshes is that extract refreshes are highly dependent on external databases for adequate performance. (In our tests, we refreshed data using workbooks that used published extracts from MS SQL Server.) Extract refresh performance and scale is highly reliant on the database hardware specifications. In addition, data characteristics, such as the types of joins and

the complexity of the queries that are executed will impact scale. For this reason, you should ensure that extract refreshes and notifications have sufficient capacity available to them to complete their work before end user peak loads are expected on the system.

Backgrounder Considerations

The Backgrounder process does much of the work related to extract refreshes, subscriptions, and other scheduled background jobs. These jobs don't compete with capacity if you schedule them to run at off-peak hours. When it is not possible, you should plan for and add capacity needed for your backgrounders and other non-user-facing workloads to run along with user-facing processes.

Backgrounders are designed to consume an entire core's capacity per process because they are designed to finish the work as quickly as possible. When you run multiple Backgrounder processes, you should consider the fact that a Backgrounder process may compete for computational and network resources with other services that are running on the same machine.

Best Practices – DIY Scale Testing

If you are looking to conduct your own load testing to find out how Tableau Server scales in your environment with your workloads, here are some best practices.

1. **Don't treat Tableau Server as a black box.** Often, traditional load testing treats an application under test as a black box. This assumes no tuning/configuring or adjusting the deployment to meet load conditions. Tableau is designed to scale up and scale out and it helps to inform scalability testing by understanding Tableau architecture to drive the outcomes that work for your situation

2. **Pick the right tool for testing.** Tableau Server is a workhorse and does complex and resource-intensive work. There are many tools available to drive loads on Tableau Server. While Tableau doesn't directly support any of these tools, you should pick the one that allows for the greatest ease of use and represents your production environment the closest. Another consideration is ensuring you have the appropriate expertise in tooling and in Tableau Server available when doing load testing. We used Tabjolt for our testing. TabJolt is a point and run load testing tool based on JMeter and built to eliminate script maintenance while working with ad-hoc analytics solutions like Tableau.

3. **Select representative workbooks.** Often when we hear about performance or scale issues, it is because the workbooks being used are not authored with best practices in mind. If a single-user test on your workbooks shows a very slow response time, then you should optimize those workbooks before you embark on a load-testing project. Just like you wouldn't keep a poorly performing dashboard in a production environment, you wouldn't want to use it for testing.

4. **Start with the defaults**. When testing workbooks using live connections remember that with the introduction of parallelization in Tableau Server 9.0, you may not need as many VizQL servers as you may have deployed in previous version of Tableau Server. Start with the new 2-process default configuration and scale up incrementally as needed.

Best Practices for Optimization in The Real World

In addition to a system that is optimally designed, there are best practices that can be used to greatly improve performance and reduce average response time.

- Design your workbooks for beauty and performance. Most often when we hear customer suggest that their workbook is slow it's because it was designed without performance in mind. If a single user load time is slow, the workbook response times will be slow under heavy loads as well. While adopting a culture of analytics, providing avenues and teams where stewards can help users design great looking, insightful workbooks that perform well, will enable you to ensure you build and deliver scalable visualizations as well. Designing Efficient Workbooks is whitepaper that dives deeper into building efficient dashboards that perform well.

- The total response time an end user experiences is a combination of many things, but it's primarily time taken by Tableau combined with data retrieval. If your backend databases are slow, or your query times are slow, the visualizations will be slow. It's important to factor in your data strategy. Often data sources in an organization are curated and shared. You must ensure that you are delivering data that matters in a way that the data can support business user productivity. This means optimizing data. For example, you should be ensuring optimal joins and relevant levels of aggregation for fast queries against indexed tables. Having a good data hygiene process is important to keep your visualizations and performing well.

- Use Tableau data extracts. If your database queries are slow, consider using extracts to increase query performance. Extracts are stored locally on the server and run in memory so that users can access the data quickly without making requests to the database. Extracts can be filtered and aggregated easily and are ideal when users don't need row-level detail. Extracts significantly improve response time and enable your users to get into the analytic flow.

- Schedule updates during off-peak times. Often data sources are being updated in real time but users only need data daily or weekly. Scheduling extracts for off-peak hours can reduce peak-time load on both the database and Tableau Server. In addition, you could add additional Backgrounders on existing machines or use dedicated hardware if you have sufficient core capacity. Consider this option for faster completion of extracts.

- Avoid 'expensive' operations during peak times. Publishing, especially large files, is a very resource-consuming task. It's often easy to influence publishing behavior: ask users to publish during off-peak hours, avoiding busy times like Monday mornings. To learn when your servers are being used the most, use the Administrator Views, then create policy based on actual usage. Depending on how you have configured Tableau Server 10.0, publishing also means that a copy of the extracts is made on each of the cluster nodes for high availability. Doing this during off-peak times will also allow you to maximize network bandwidth.

- Cache views. As multiple users begin to access Tableau Server, the response time will initially increase due to contention for shared resources. With caching turned on, views from each request coming into the system will be cached and then rendered more quickly for the next viewer of the same dashboard.

- The Cache Server process, introduced in Tableau Server 9.0, can be warmed up by scheduling an email of common views following completed extract refreshes. That way future viewers are using the cached data from your earlier request. You may use other approaches to warm up the cache, such as an automated tool that loads up key visualizations that regularly see high traffic. The user can manually invalidate the external query cache at any time to refresh their data from the data source. This action also forces a regeneration of the cache. This way, the users can always get a fresh copy of the data regardless if a version is already in cache.

Summary

Tableau Server 10 is an enterprise class, scalable platform that can support any size organization. It can run on-premise in private clouds or public clouds and can scale linearly with added worker capacity. While each environment will have its own unique characteristics and configuration, Tableau Server's architecture will allow you to scale your deployments to meet your user demand needs.

While your scalability and performance mileage may vary and these aren't specific recommendations for all situations, Tableau Server 10 can support teams or departments between 25 to 100 users on 8-16 core capacity. As you look to support teams of 100 to 1000 users, depending on your usage and data freshness needs you may find 16 to 24 cores are good to get you started. If you need to support more users or additional background load, you could grow your deployment beyond 32-64 cores supporting larger and larger workloads by adding additional worker nodes to expand to even cloud scales.

About Tableau

Tableau helps people transform data into actionable insights that make an impact. Easily connect to data stored anywhere, in any format. Quickly perform ad hoc analyses that reveal hidden opportunities. Drag and drop to create interactive dashboards with advanced visual analytics. Then share across your organization and empower teammates to explore their perspective on data. From global enterprises to early-stage startups and small businesses, people everywhere use Tableau's analytics platform to see and understand their data.

Resources

Tableau for the Enterprise: An IT overview

Server Admin Guide

Tableau Server 10.0 High Availability: Delivering mission-critical analytics at scale

Tableau on Amazon Web Services