

# Sitecore Experience Platform 8.1 Performance White Paper



**sitecore**<sup>®</sup>  
Own the experience<sup>™</sup>

## Table of contents

Table of contents .....	2
Executive summary .....	3
Overview .....	3
Key findings .....	3
Deployment .....	5
Virtual Machine instance size .....	6
Azure configuration .....	7
Software .....	8
Sitecore configuration .....	8
Test methodology .....	10
Site specifications: .....	10
Test scenarios .....	10
Single interactions .....	10
Anonymous multiple interactions .....	12
Authenticated multiple interactions .....	12
Load configuration .....	13
Run settings .....	14
Data configuration .....	14
Catalog data (Sitecore items) .....	14
User profiles .....	14
Test rig configuration .....	14
Performance metrics .....	15
Test results .....	17
General performance findings .....	17
Key metrics .....	18
Key metrics .....	19
CPU time .....	20
Detailed breakdown of the Sitecore namespace CPU cycle cost .....	21
Site database servers (Web, Master, Core) .....	22
Session database server .....	23
Processing server .....	24
Processing: Session and tracking .....	25
MongoDB processing server .....	26
Reporting database server .....	27
Analytics - MongoDB server .....	28

# Executive summary

## Overview

The performance tests used the Sitecore Reference Storefront Powered by Commerce Server website to measure the key performance metrics in a scaled deployment. The website traffic was tested using a collection of common scenarios. The following statements summarize the most important information to consider when you interpret the results described in this whitepaper and build your own sites with Sitecore Experience Platform (SXP) 8.1.

- Duration: 1 hour, excluding a 5-minute warm up and 2-minute cool down
- User load: Step from 110 to 800 users
- User think time: Normal distribution profile based on 2 seconds
- In an enterprise deployment, static content is typically located in the content delivery network (CDN) or cached by a network appliance.

The results presented in this document do not take into account the cost of providing static content.

- The Sitecore Reference Storefront Powered by Commerce Server website, using Microsoft ASP.NET MVC, was used as the test site. The deployments included Commerce Server databases and related packages that enabled the test site.
- The *analytics* database on MongoDB server was configured using two shards to scale writing performance on the primary instances.
- Processing used two processing pool lanes to split the write capabilities of the *tracking\_live* and *tracking\_contact* databases across multiple disks.
- The database servers were configured with an instance type of Extra Large A4. The SQL data and log files were stored on a hard drive that consisted of multiple virtual hard drives and software striped using Microsoft Storage Spaces.
- The MongoDB instances used Premium Disk to increase disk throughput.
- You can use the deployment techniques described in this document for on-premise and other Infrastructure as a Service (IaaS) cloud providers.
- The results are tightly linked to the software and virtual machine configurations used, including the instance size, the number of instances that make up each server farm, the amount of system memory on each instance, the database disk configuration, and the virtual CPU capabilities.

## Key findings

- To achieve the best results, you should always follow the Azure SQL on VM and Sitecore CMS/DMS guidelines:
  - Performance best practices for SQL Server in Azure Virtual Machines  
<https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-sql-server-performance-best-practices/>

- Performance Guidance for SQL Server in Azure Virtual Machines

<http://msdn.microsoft.com/en-us/library/dn248436.aspx>

- CMS Performance Tuning Guide

<https://sdn.sitecore.net/Reference/Sitecore%207/CMS%20Performance%20Tuning%20Guide.aspx>

- The size of each virtual machine has a significant effect on performance.

When you try to balance cost with the required performance levels, a cost analysis should include a comparison of the instance sizes with the size of the farm.

- The load on the content delivery web servers was distributed using Azure Load Balancer, which randomly distributes the load.

The load was evenly distributed, but VM instances would occasionally show processor time percentage unevenness. This was due to the fact that performance in a VM instance does not consider load balancing.

- In a multi-region deployment, use Azure Traffic Manager to ensure that the closest geographic location is recommended.

For more information about Azure Traffic Manager, see:

<https://azure.microsoft.com/en-us/services/traffic-manager/>

- Using Microsoft SQL Server page compression can improve disk throughput at the expense of CPU time.

For these tests, page compression was used for the databases that experienced significant write activity.

- Key test results

The following results were collected over the 1-hour test run:

- Total page views: 991,000
- Average response time: 0.29 seconds
- Max number of simultaneous user load: 800 users
- Average % CPU processor at max load: 82.2%

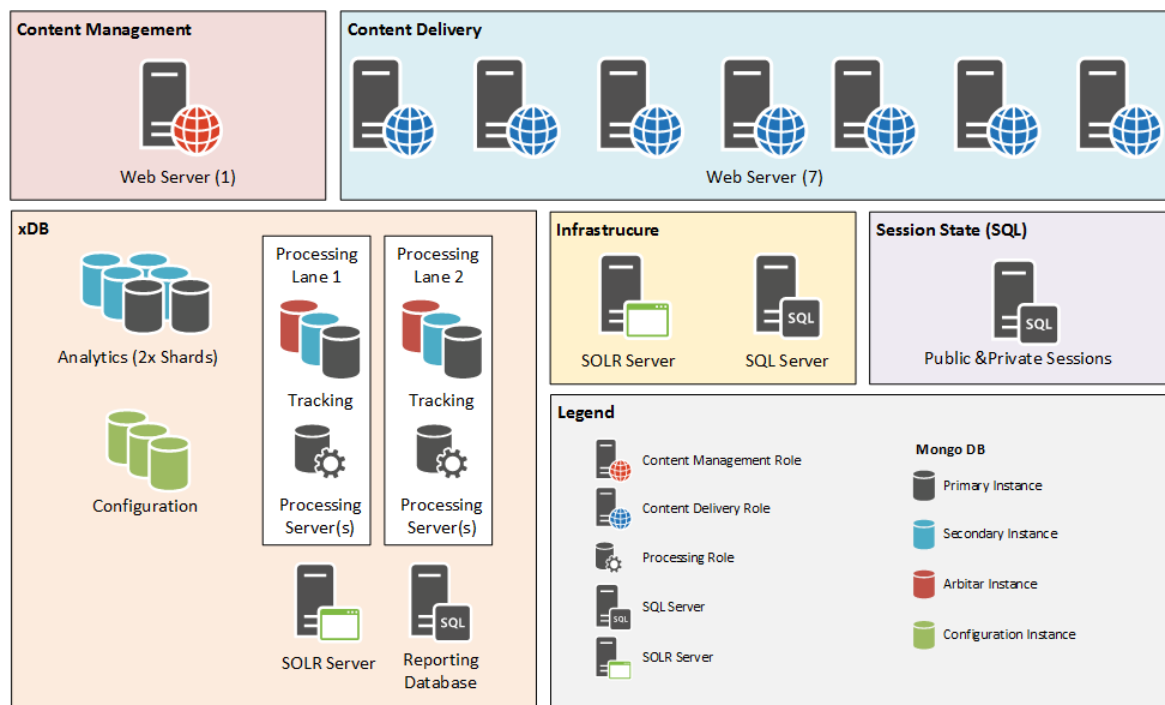
## Deployment

A Sitecore Experience Platform scaled configuration was used for testing. All of the Sitecore Commerce 8.1 components were installed on separate instances to maximize scalability.

In this document we refer to the internet-facing load balanced web servers as Content Delivery(CD) servers.

Testing focused on the production level runtime performance of the CD servers.

The Content Management (CM) portion of the deployment was not configured behind a load balancer. Sitecore does allow the CM portion of the deployment to be scaled horizontally.



For more information about the supported Sitecore CMS deployments, see the Sitecore CMS Scaling Guide:

<http://sdn.sitecore.net/Reference/Sitecore%207/Scaling%20Guide.aspx>.

## Virtual Machine instance size

The following settings were applied to the Microsoft Azure Virtual Machine instances:

Virtual machine	Instance size
Web server instances	Extra large (A4) instance <ul style="list-style-type: none"><li>• Includes: Content Management, Content Delivery, and Processing instances</li><li>• 14 GB memory</li><li>• 8 virtual cores</li><li>• 64-bit platform</li></ul>
Database instances	Extra large (A4) instance <ul style="list-style-type: none"><li>• 14 GB memory</li><li>• 8 virtual cores</li><li>• Database disk (containing data and log files); sixteen 25GB virtual hard disk, software striped with 64-KB cluster size using Storage Spaces</li><li>• 64-bit platform</li></ul>
MongoDB instances	Large (DS3) instance <ul style="list-style-type: none"><li>• 7 GB memory</li><li>• 4 virtual cores</li><li>• 64-bit platform</li><li>• Configured with Premium Disk Storage Accounts (SSD-based arrays) to maximize write performance</li></ul>
Index instances	Large (A3) instance <ul style="list-style-type: none"><li>• 7 GB memory</li><li>• 4 virtual cores</li><li>• Index disk is made up of four 100GB disks striped together with a cluster size of 64-KB using Storage Spaces</li><li>• 64-bit platform</li></ul>
Load test rig instances	Large (A3) instance <ul style="list-style-type: none"><li>• 1 controller</li><li>• 5 agents</li></ul> Followed Microsoft's guidance: <a href="http://msdn.microsoft.com/en-us/library/ff937706.aspx">http://msdn.microsoft.com/en-us/library/ff937706.aspx</a>

## Azure configuration

The following settings were applied to Microsoft Azure:

Microsoft Azure	Settings
SQL server	<p><b>SQL Server in Virtual Machine Performance Guidance</b> Both of the SQL servers in this deployment were configured for the Online Transaction Processing (OLTP) workloads.</p> <ul style="list-style-type: none"><li>• <a href="https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-sql-server-performance-best-practices/">https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-sql-server-performance-best-practices/</a></li><li>• <a href="http://msdn.microsoft.com/en-us/library/dn248436.aspx">http://msdn.microsoft.com/en-us/library/dn248436.aspx</a></li></ul> <p><b>Tables with SQL page compression enabled</b> The following tables that are write-heavy had SQL Page Compression enabled to optimize disk usage:</p> <ul style="list-style-type: none"><li>• Sitecore Analytics.</li><li>• Sitecore Commerce Transaction Table.</li></ul> <p><b>Note</b> Compression has an impact on CPU usage.</p> <p><b>Disk configuration</b></p> <ul style="list-style-type: none"><li>• Combine the disk with storage spaces using a 64-KB cluster size.</li></ul> <p>For more information about Microsoft Storage Spaces, see: <a href="http://social.technet.microsoft.com/wiki/contents/articles/15198.storage-spaces-overview.aspx">http://social.technet.microsoft.com/wiki/contents/articles/15198.storage-spaces-overview.aspx</a></p>
MongoDB server	<ul style="list-style-type: none"><li>• Premium Storage was used to optimize the disk's write capabilities.</li></ul> <p>For more information about premium disks, see: <a href="https://azure.microsoft.com/en-us/documentation/articles/storage-premium-storage-preview-portal/">https://azure.microsoft.com/en-us/documentation/articles/storage-premium-storage-preview-portal/</a></p>
Storage accounts	<p>The storage accounts that hosted the virtual hard disks and data disks were split as follows:</p> <ul style="list-style-type: none"><li>• Infrastructure: Content Management, Directory Server</li><li>• Main SQL database: Sitecore and Sitecore Commerce Databases</li><li>• Reporting SQL database: Reporting databases</li><li>• Content Delivery: Content Delivery</li><li>• Index: SOLR instances</li><li>• Mongo (Premium Storage Account): MongoDB instances</li></ul>

## Software

The following software was used:

Server	Software
Web server <ul style="list-style-type: none"><li>Content Management</li><li>Content Delivery</li></ul>	<ul style="list-style-type: none"><li>Microsoft Windows Server 2012 R2 Datacenter</li><li>Microsoft Internet Information Services (IIS) 8.5 (Integrated Mode/.NET Framework v4.0)</li><li>Sitecore CMS 8.1<ul style="list-style-type: none"><li>Sitecore.Solr.Support</li><li>Sitecore Commerce Connect 8.1</li></ul></li><li>Sitecore Commerce 8.1<ul style="list-style-type: none"><li>Sitecore Commerce Server Connect 8.1</li><li>Sitecore Reference Storefront Powered by Commerce Server 8.1</li></ul></li></ul>
Database server	<ul style="list-style-type: none"><li>Microsoft Windows Server 2012 R2 Datacenter</li><li>Microsoft SQL Server 2014 Enterprise</li></ul>
Index server	<ul style="list-style-type: none"><li>Microsoft Windows Server 2012 R2 Datacenter or Linux Server (CentOS-based was tested)</li><li>Apache Tomcat (8.0.22 was tested)</li><li>Apache SOLR (4.10.4 was tested)</li><li>Java Runtime Environment (JRE) was 1.7 (64-bit edition)</li></ul>
Load test rig servers	<ul style="list-style-type: none"><li>Microsoft Visual Studio 2013 Ultimate (Update 4)</li><li>Microsoft Visual Studio 2013 Test Controller (Update 4)</li><li>Microsoft Visual Studio 2013 Test Agent (Update 4)</li></ul>

## Sitecore configuration

The following settings were applied to Sitecore and its components during the test.



Sitecore	Settings
Content Delivery	<p><b>CMS Performance Tuning Guide</b>  <a href="https://sdn.sitecore.net/Reference/Sitecore%207/CMS%20Performance%20Tuning%20Guide.aspx">https://sdn.sitecore.net/Reference/Sitecore%207/CMS%20Performance%20Tuning%20Guide.aspx</a></p> <p><b>Recommended Settings</b></p> <ul style="list-style-type: none"> <li>Rendering Caching was configured to cache where applicable for the Sitecore Reference Storefront Powered by Commerce Server website</li> <li>In the <code>sitecore.config</code> file, the value of the <code>Caching.DisableCacheSizeLimits</code> setting was set to <code>false</code>.</li> </ul> <p>You typically use this setting to tune cache sizes to acceptable levels during development. It is best practice to set this setting to <code>false</code> in production environments unless a Sitecore Support representative has instructed you to set it to <code>true</code> to address a specific problem.</p> <ul style="list-style-type: none"> <li>Sitecore log file writes were kept to a reasonable level. In the <code>sitecore.config</code> file, in the <code>logger</code> section, the <code>level</code> value was set to <code>WARN</code>.</li> </ul>
Sitecore Analytics	<p><b>xDB Configuration Guide</b>  <a href="https://sdn.sitecore.net/SDN5/Reference/Sitecore%207/xDB%20Configuration%20Guide.aspx">https://sdn.sitecore.net/SDN5/Reference/Sitecore%207/xDB%20Configuration%20Guide.aspx</a></p> <ul style="list-style-type: none"> <li>The <i>analytics</i> MongoDB database can be sharded to increase its scaling capabilities. For this test, the MongoDB instance consisted of two shards.</li> <li>A combination of MongoDB instances and processing servers, collectively called Processing Lanes, can be deployed to increase the scalability of the processing tier. For these tests, two processing lanes that made up of a single processing role were hosted in an A4 instance.</li> <li>The processing lanes were split, with 4 CD servers configured to use one lane and the remaining 3 CD servers connected to the other lane. <ul style="list-style-type: none"> <li>The 1st processing lane's databases were hosted on a P20 premium disk.</li> <li>The 2nd processing lane's databases were hosted on a P10 premium disk.</li> </ul> </li> </ul> <p>For more information about Azure Premium Disks, see:  <a href="https://azure.microsoft.com/en-us/documentation/articles/storage-premium-storage-preview-portal/">https://azure.microsoft.com/en-us/documentation/articles/storage-premium-storage-preview-portal/</a></p>

# Test methodology

## Site specifications:

Configuration	Values
Web technology	ASP.NET MVC
Number of pages	Over 1 million (product detail pages)
Multi-language	3 languages
Search page	True, search is access on product detail pages
Caches are limited (CD instances)	No, <code>DisableCacheSizeLimits = true.</code>
HTML caches are used	True
Number of page templates	27
Web index number of documents	3,499,533 documents
Web index size	3.94 gigabytes
Number of user profiles	10 million

## Test scenarios

Performance testing employed the following scenario groups:

- Single interactions – Anonymous and authenticated users that create a single analytics interaction.
- Anonymous multiple interactions – Anonymous users that create multiple analytics interactions
- Authenticated multiple interactions – Authenticated users that create multiple analytic interactions

### Single interactions

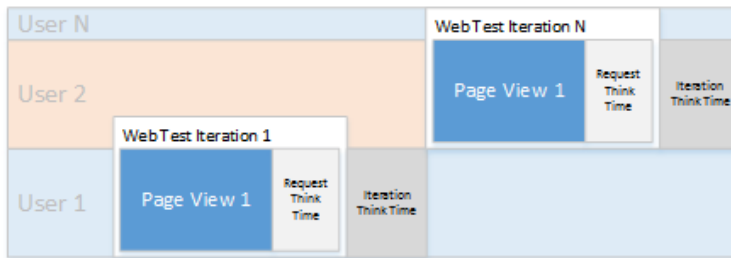
#### Anonymous with a single interaction and a single page view per interaction

This scenario simulates anonymous visits to the website that create a single interaction with a single page view.

1. Browse to a single page.

# Performance White Paper

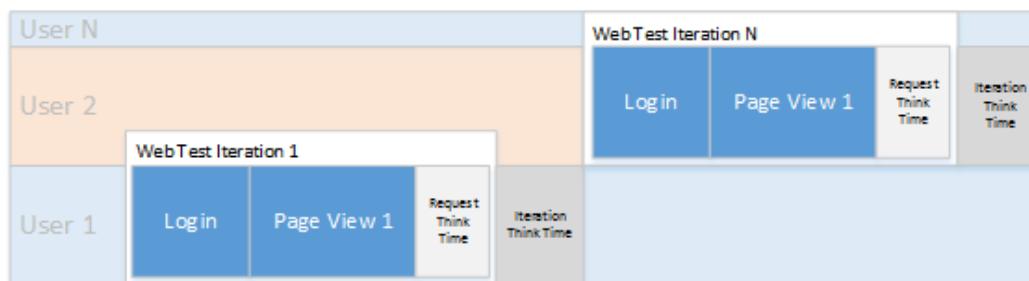
2. Clear your cookies and browse to a new page.



## Anonymous with a single interaction and many page views per interaction

This scenario simulates anonymous visits to the website that create a single interaction with multiple page views.

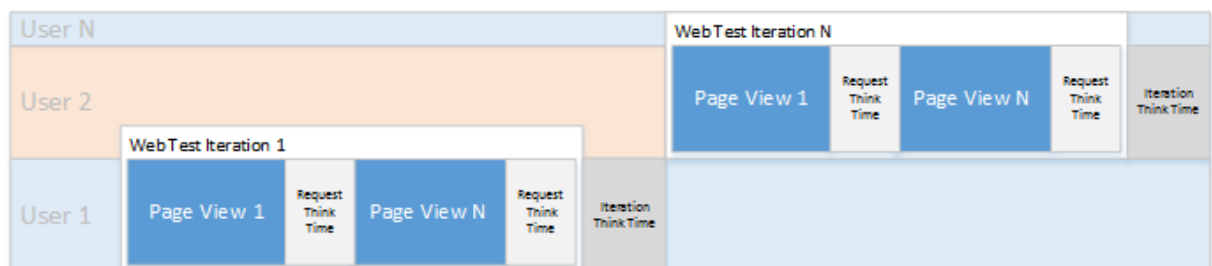
1. Browse to 10 random pages.
2. Clear your cookies and browse to 10 new pages.



## Authenticated with a single interaction and a single page view per interaction

This scenario simulates authenticated, or identified, visits to the website that create a single interaction with multiple page views.

1. Trigger the login page, which calls Analytics Identification using `Tracker.Current.Session.Identify()`
2. Browse to a single page.
3. Clear your cookies and browse to a new page.



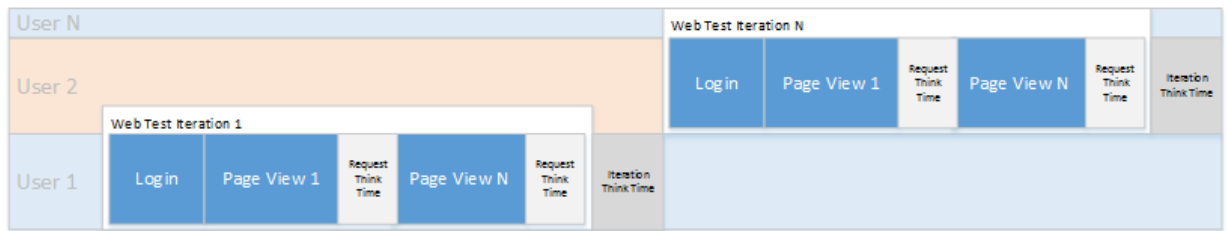
## Authenticated with a single interaction and many page views per interaction

This testing scenario simulates authenticated, or identified, visits to the website that create a single interaction with multiple page views.

1. Trigger the login page, which calls Analytics Identification using `Tracker.Current.Session.Identify()`
2. Browse to 30 random pages.

# Sitecore Experience Platform 8.1

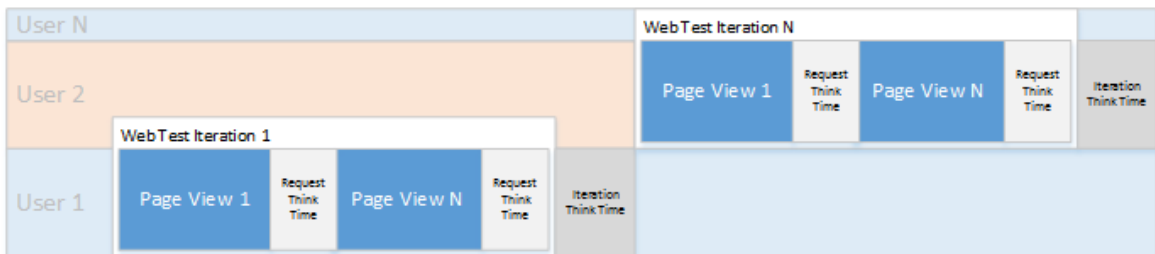
3. Clear your cookies and browse to multiple pages again.



## Anonymous with a single interaction and a large number of page views per interaction

This scenario simulates anonymous visits to the website that create a single interaction with a large number of page views. This testing scenario simulates a robot interacting with the site.

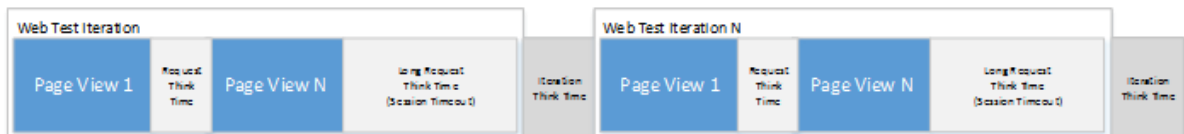
1. Browse to 200 random pages.
2. Clear your cookies and start browsing again.



## Anonymous multiple interactions

### Anonymous with multiple interactions and multiple page views per interaction

1. Browse to 10 random pages.
2. Wait 240 seconds to allow the session to timeout and then repeat.



## Authenticated multiple interactions

### Authenticated with multiple interactions and multiple page views per interaction

1. At the start of each user thread, trigger the login page, which calls Analytics Identification using `Tracker.Current.Session.Identify()`
2. Browse to 10 random pages.
3. Wait 240 seconds to allow the session to timeout and then repeat the process.



## Load configuration

The following table lists the load that was used in each scenario:

Scenario group	Interactions	Page views per interaction	Percentage of traffic	Number of users	Authentication type
Single interactions	1	1	20%	160	Anonymous
	1	10	50%	400	Anonymous
	1	200	10%	80	Anonymous
	1	1	5%	40	Authenticated
	1	10	5%	40	Authenticated
Anonymous multiple interactions	Multiple	10	5%	40	Anonymous
Authenticated multiple interactions	Multiple	10	5%	40	Authenticated

## User specifications

Configuration	Values
Max number of users	800
Think time	2 seconds
Single interaction group	Step <ul style="list-style-type: none"> <li>Initial user count: 100</li> <li>Maximum user count: 720</li> <li>Step duration in seconds: 150</li> <li>Stamp ramp time in seconds: 30</li> <li>Step user count: 100</li> <li>Percentage new user: 0</li> <li>Think profile: normal distribution</li> </ul>
Anonymous multiple interactions group	Step <ul style="list-style-type: none"> <li>Initial user count: 5</li> <li>Maximum user count: 40</li> <li>Step duration in seconds: 150</li> <li>Stamp ramp time in seconds: 30</li> <li>Step user count: 5</li> <li>Percentage new user: 100</li> <li>Think profile: normal distribution</li> </ul>
Authenticated multiple interactions group	Step <ul style="list-style-type: none"> <li>Initial user count: 5</li> <li>Maximum user count: 40</li> <li>Step duration in seconds: 150</li> <li>Stamp ramp time in seconds: 30</li> <li>Step user count: 5</li> <li>Percentage new user: 100</li> <li>Think profile: normal distribution</li> </ul>

## Run settings

- Duration: 1 hour
- Warm up: 5 minutes
- Web test connection model: Connection per user
- Dependent request: Off

## Data configuration

### Catalog data (Sitecore items)

- Products: 1 Million – 50% products, 50% product families
  - Product families contained a random number of variants from 1-20.
  - 50 % of all products and product families included between 1-4 relationships.
- Categories: 11,000
- Category depth: 5
- Products/Category: 100

### User profiles

- 10 million users.
  - Each commerce user includes one shipping address and four credit cards.
  - Each credit card includes one billing address
- 40 million credit cards
- 50 million addresses
- A total of 100 million profiles objects were used.
- All the commerce profiles were imported into Sitecore.

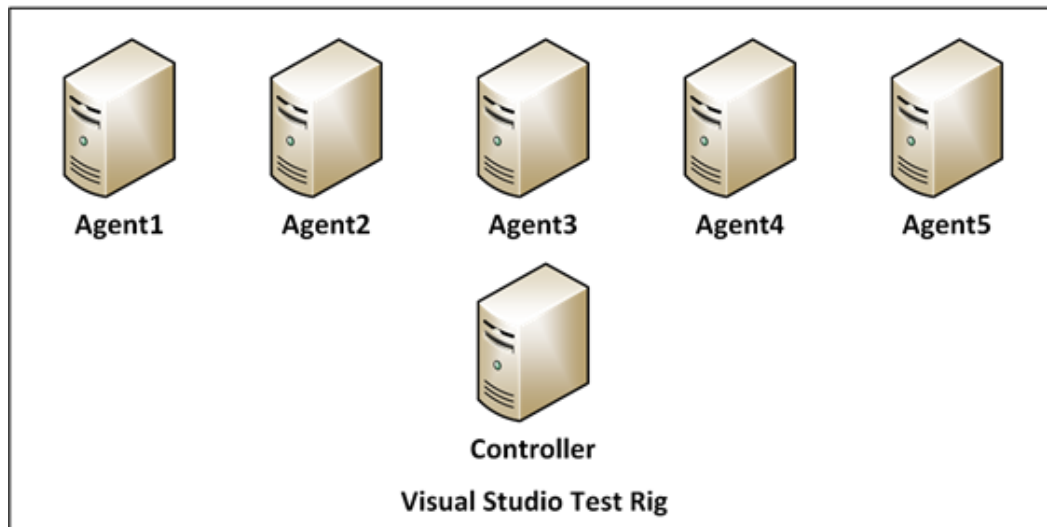
## Test rig configuration

To generate the performance load, this test used a Visual Studio 2013 Load Test Rig with the following configuration:

- The Visual Studio 2013 test agents and the controller were deployed to Azure Large A3 instances.
- Each agent had a single data disk attached, and the agent service was set to utilize the attached disk as the working directory. This ensured that the agent had enough disk capacity to handle load generation, because the default OS disk did not offer the required performance level.

# Performance White Paper

For more information on setting up the Visual Studio Test agents and controllers, visit:  
<http://msdn.microsoft.com/en-us/library/dd648127.aspx>



## Performance metrics

Counter	Performance counter	Description
Disk % idle time	\LogicalDisk(*)\% Idle Time	The percentage of time that the disk system was not processing requests and no work was queued.
% processor time	\Processor Information(_Total)\% Processor Time	The primary indicator of the processor activity.
ASP.NET request queued	\ASP.NET\Requests Queued	The number of requests that are currently in the queue.
Avg. page time	-	The average time a page responds to the test agent.
Batch requests/sec	\SQLServer:SQL Statistics\Batch Requests/sec	The number of Transact-SQL batch requests received by the server per second. This statistic is affected by all constraints such as I/O, the number of users, cache size, complexity of requests, and so on. High values mean good throughput.

Counter	Performance counter	Description
Network bytes/second	\Network Interface\Bytes Total/sec	Shows the rate at which bytes are sent and received on the network interface, including framing characters. Bytes total/sec is the sum of the values of Network Interface\Bytes Received/sec and Network Interface\ Bytes Sent/sec.
Pages/second	-	The number of pages per second, measured by the test agent
Private bytes	\Process\Private Bytes	The current byte size of the virtual address space that the process is using.
Requests/Second	-	The number of requests per second measured at the test agent. This number includes all the requests to the test deployment; page requests and any dependent requests that were transmitted are included.
User load	-	The current number of users.
Virtual bytes	\Process\Virtual Bytes	The current size, in bytes, of the virtual address space that the process is using.
Working set	\Process\Working Set	The set of memory pages, or areas of memory allocated to a process that were recently used by the threads in the process.



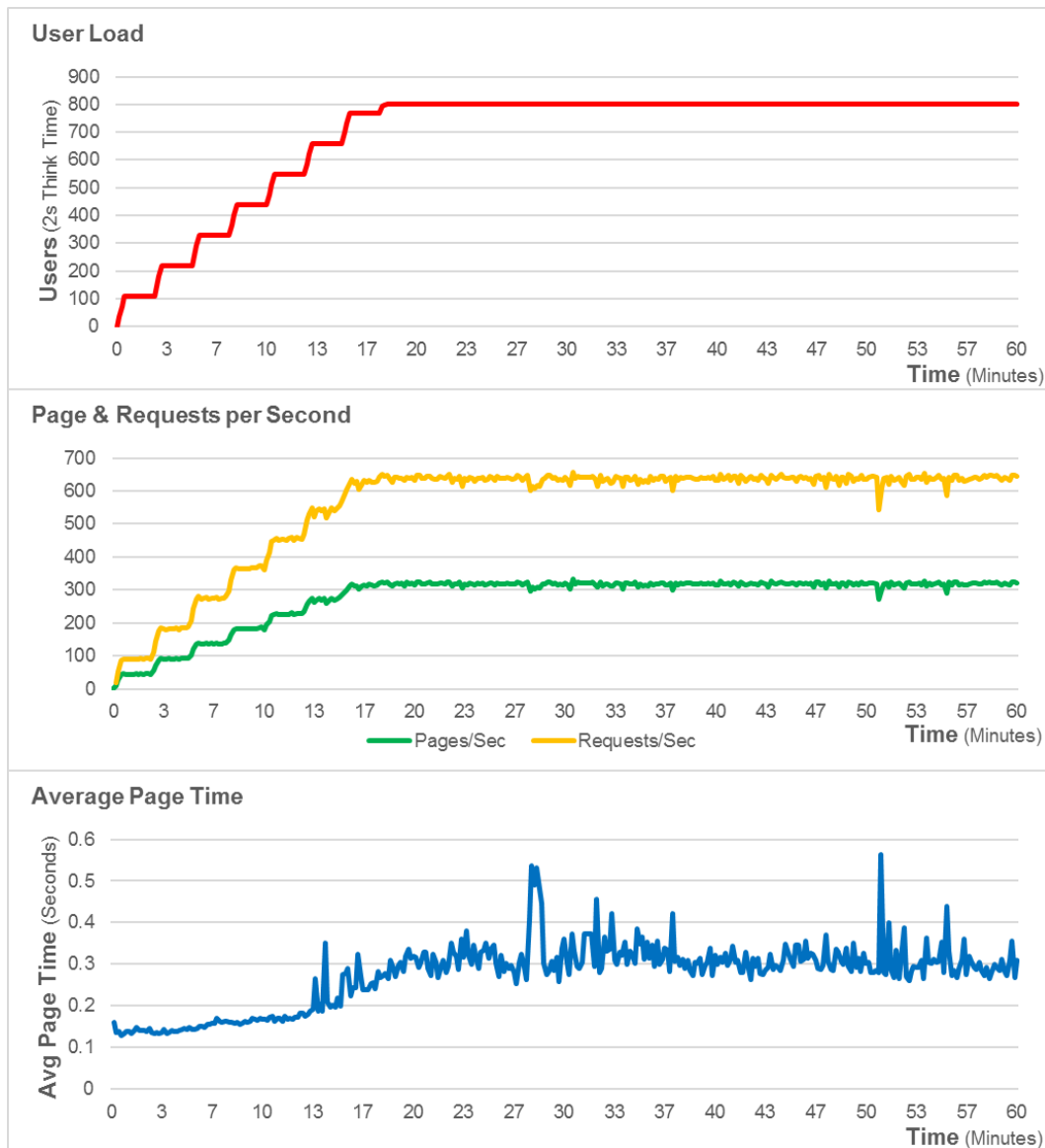
## Test results

During the test run, a brief network outage with the storage account occurred at the load test rig. This caused a spike in the results for 4 out of 360 counter collections. For the charts contained in this whitepaper, the affected results of those collections are averaged out. This type of issue is seen often in a shared hosting environment such as Azure.

### General performance findings

- For the tested scenarios, the main performance bottleneck on the CD servers was processor time.  
Increasing the number of cores, core speed, and the number of server instances will allow the site to scale further, provided there is enough capacity on the MongoDB and SQL instances.
- The out of process session state plays a major role in the performance capacity of the CD servers. Using the session performance script, which places sessions in the temp database, greatly improves performance and scale.
- Azure Premium disk was used to scale the Analytics and Processing Servers. On-premise deployments should consider using SAN or SSD-based disks for similar throughput results.
- The session databases were hosted on Premium Disk. The shared and private sessions were located on a different disk because Azure disks generally do not perform well with sequential disk writes such as database log files.
- MongoDB primaries need good disks to perform and scale well. Testing determined that having at least one secondary with a similar disk configuration ensures that data is safely synced in a timely fashion.
- The MongoDB processing instances can act as a bottleneck for a deployment's CD farm. When this is the case, and depending on the Processing Role configuration, the primary processing disk can be overwhelmed to the point that front-end servers slow down. This is caused by the time it takes for updates to be made to the MongoDB database, which is dependent on the underlying disk capabilities.  
CD server writes are related to the number of private and shared sessions expiring. One way to scale the MongoDB processing capabilities is to add multiple processing lanes, which effectively spreads the writes across multiple MongoDB instances. This is an effective way to avoid scale limitations in a CD farm.
- MongoDB instance throughput can be improved by ensuring that databases and journaling are on different disks.
- The MongoDB Wired Tiger storage engine provided a slight throughput improvement over the MMapv1 storage engine. Load testing with the expected user load matrix and production site would be required to determine which storage engine is best suited.

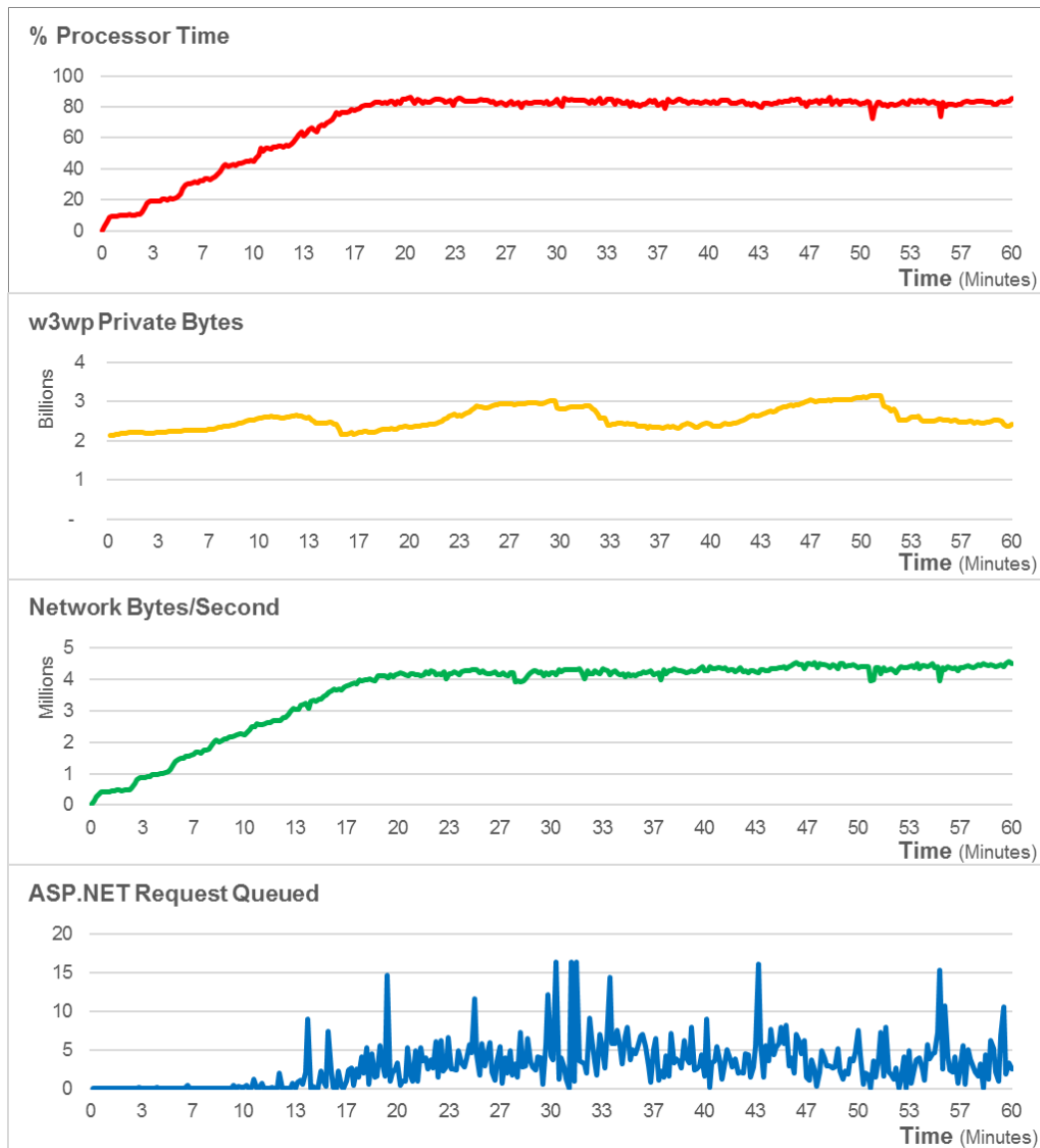
## Key metrics



These charts display the peak number of pages per second and the average page time. Most of the pages requested during testing consisted of 2 requests to the server, the page request, and the visitor identification request.

Testing resulted in a maximum page per second value of 327 pages, based on 654 requests. At maximum load, the average pages being handled per second was 318 pages, based on 636 requests, with a maximum page response time of 0.56 seconds and an average of 0.36 seconds. The worst performing page was the Login Submit page, with an average page time of 0.76s. The maximum user load was determined by targeting 80% of total content delivery % processor time.

## Key metrics

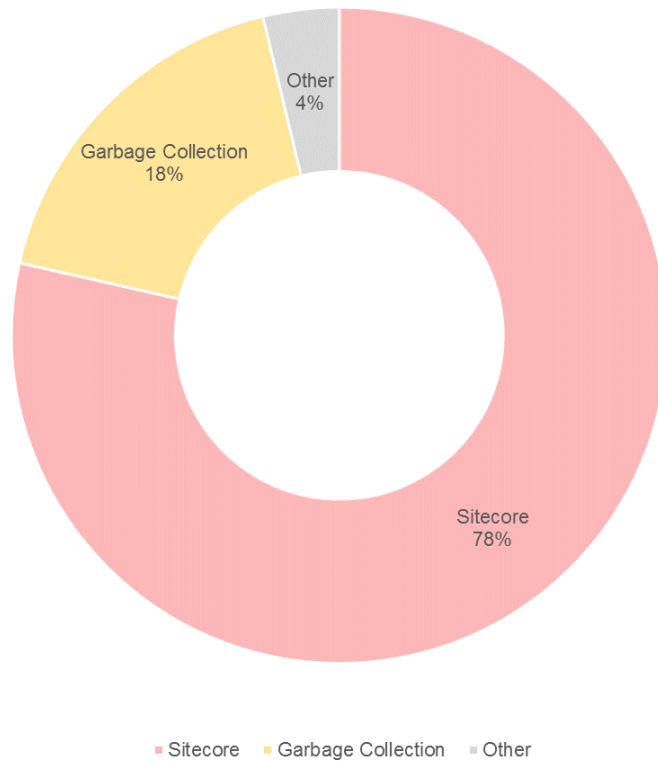


The CD load target was 80% of the available % processing time at maximum load.

At maximum load, the average processor time was 82.8%, the average Network Bytes/Second was 4.3 MB/Second, and the number of ASP.NET Request Queued was 4. Because the Processing Queue's MongoDB instance played a role in the capacity of the CD servers, a second processing lane was created in the test deployment.

## CPU time

Breakdown of CPU Time Usage



### Notes:














































- The CPU trace was only performed on a single content delivery server in sampling mode.
- Sitecore appears to be the major contributor to CPU cycle consumption during the test run.
- The Other section includes the cost found within the Reference Storefront Site's namespace.

The following signatures appeared to be hotspots in the Sitecore namespace:

- `Sitecore.SessionProvider.SessionStateSerializer.Compress()`
- `Sitecore.SessionProvider.Sql.SqlSessionStateStore.GetItemExclusive()`
- `Sitecore.Data.Fields.Field.GetTemplateField()`
- `Sitecore.Analytics.Data.HttpSessionContextManager.GetSession()`

# Performance White Paper

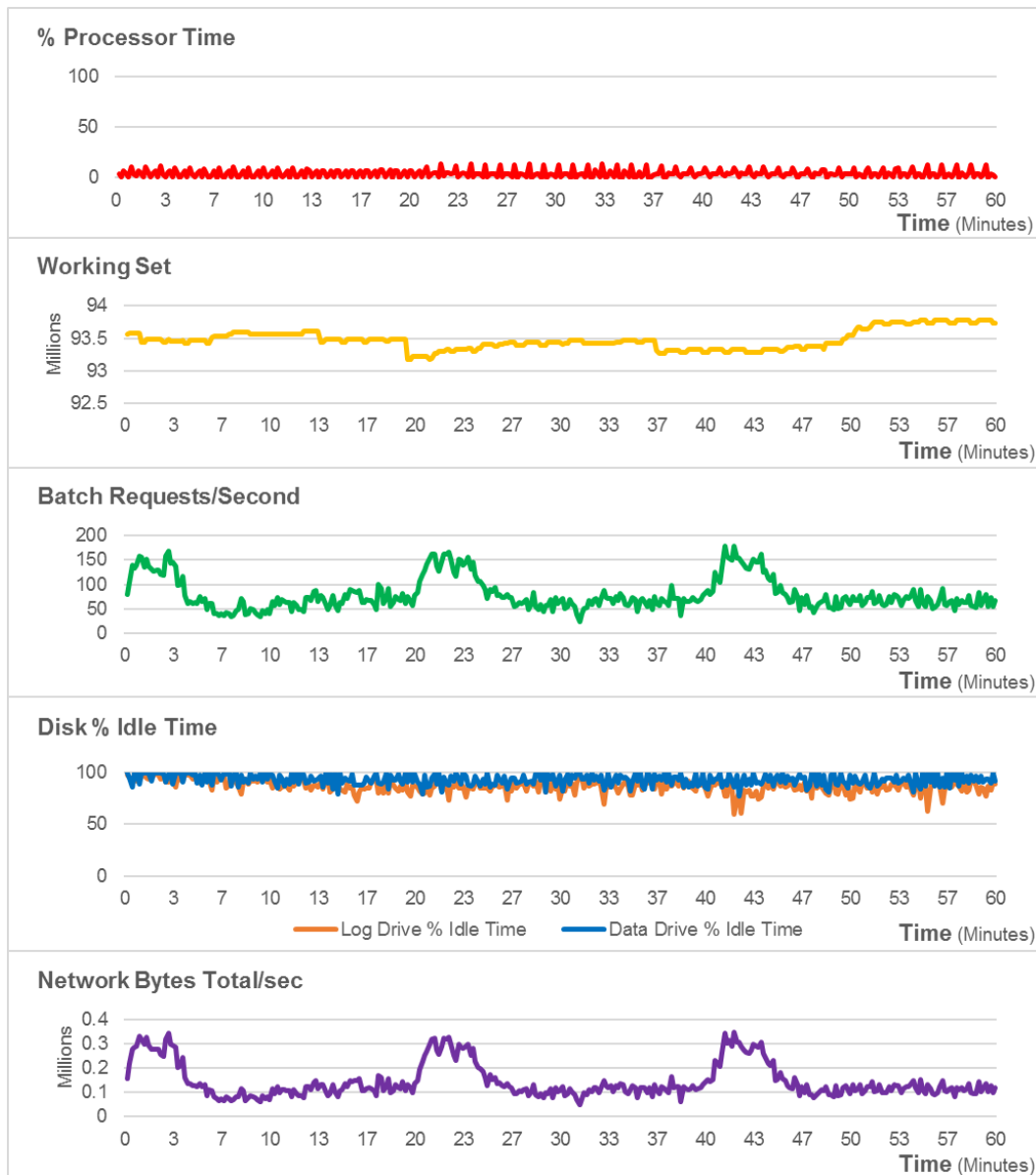
## Detailed breakdown of the Sitecore namespace CPU cycle cost

<b>4 Sitecore</b>		19.84%	1,032,188 ms
Sitecore.Mvc.Controllers.SitecoreActionInvoker.InvokeAction(ControllerContext, String)		0.99%	51,750 ms
Sitecore.Mvc.Extensions.XmlExtensions.ToXmlNode(XElement)		0.98%	51,156 ms
Sitecore.Caching.Generics.ContextCache`1.GetData(String)		0.92%	47,734 ms
Sitecore.Mvc.Presentation.ViewRenderer.Render(TextWriter)		0.69%	36,047 ms
Sitecore.Pipelines.CorePipeline.Run(PipelineArgs)		0.67%	34,891 ms
Sitecore.Common.Switcher`2.GetStack(Boolean)		0.65%	33,688 ms
Sitecore.Mvc.Presentation.XmlBasedRenderingParser.Parse(XElement, Boolean)		0.64%	33,313 ms
Sitecore.Pipelines.CorePipelineFactory.GetPipeline(String, String)		0.63%	32,578 ms
Sitecore.Collections.FieldCollection.ReadAll		0.57%	29,625 ms
Sitecore.Caching.ItemCache.GetItem(ID, Language, Version)		0.47%	24,641 ms
<b>4 Sitecore.Data</b>		16.66%	866,984 ms
Sitecore.Data.Fields.Field.GetTemplateField		3.16%	164,438 ms
Sitecore.Data.Items.RenderingItem.IsAvalableNotBlobNotSystemField(Field)		2.74%	142,578 ms
Sitecore.Data.Templates.Template.AddFields(Template, SafeDictionary[IDTemplateField])		1.78%	92,813 ms
Sitecore.Data.Items.RenderingItem.GetParametersInternal		1.44%	75,031 ms
Sitecore.Data.Items.RenderingItem.CacheDataRetrieving(String, Func[T])		1.02%	53,250 ms
Sitecore.Data.Templates.Template.AddBaseTemplates(List[Template], Boolean, Set[ID])		0.66%	34,266 ms
Sitecore.Data.ID.ToString		0.36%	18,797 ms
Sitecore.Data.Engines.TemplateEngine.GetTemplate(ID)		0.31%	16,344 ms
Sitecore.Data.Templates.Template.GetImmediateBaseTemplates		0.30%	15,859 ms
Sitecore.Data.Items.Item.TryGetCacheValue(Item, ItemUri&)		0.28%	14,625 ms
<b>4 Sitecore.SessionProvider.Sql.SqlSessionStateStore</b>		15.36%	799,109 ms
Sitecore.SessionProvider.Sql.SqlSessionStateStore.GetItemExclusive(Guid, String, SessionStateLockCookie, SessionStateLockCookie&, Int32&)		10.81%	562,266 ms
Sitecore.SessionProvider.Sql.SqlSessionStateStore.UpdateAndReleaseItem(Guid, String, String, SessionStateActions, SessionStateStoreData)		4.49%	233,844 ms
Sitecore.SessionProvider.Sql.SqlSessionStateStore.InsertItem(Guid, String, Int32, SessionStateStoreData)		0.05%	2,813 ms
Sitecore.SessionProvider.Sql.SqlSessionStateStore.RemoveItem(Guid, String, String)		<0.01%	141 ms
Sitecore.SessionProvider.Sql.SqlSessionStateStore.ReleaseItem(Guid, String, String)		<0.01%	47 ms
<b>4 Sitecore.SessionProvider.SessionStateSerializer</b>		14.83%	771,578 ms
Sitecore.SessionProvider.SessionStateSerializer.Compress(SessionStateStoreData)		12.88%	670,313 ms
Sitecore.SessionProvider.SessionStateSerializer.Decompress(Byte[])		1.80%	93,844 ms
Sitecore.SessionProvider.SessionStateSerializer.Serialize(SessionStateStoreData, Boolean)		0.06%	3,234 ms
Sitecore.SessionProvider.SessionStateSerializer.Serialize(BinaryWriter, SessionStateStoreData)		0.05%	2,547 ms
Sitecore.SessionProvider.SessionStateSerializer.Deserialize(Byte[])		0.03%	1,641 ms
<b>4 Sitecore.Analytics</b>		11.86%	617,047 ms
Sitecore.Analytics.Data.HttpSessionContextManager.GetSession		9.10%	473,625 ms
Sitecore.Analytics.Tracking.SharedSessionState.SharedSessionStateManager.LockAndLoadContact(Guid)		2.29%	119,281 ms
Sitecore.Analytics.Pipelines.EnsureSessionContext.ClusterCheck.ExtractRequestedHost(HttpContextBase)		0.05%	2,781 ms
Sitecore.Analytics.Pipelines.EnsureSessionContext.ClusterCheck.ThisRequestShouldNeverBeRedirected(Uri)		0.02%	1,250 ms
Sitecore.Analytics.Data.TrackingField.ProcessProfileKeys(ContentProfile, ProfileItem)		0.02%	1,250 ms
Sitecore.Analytics.Web.ContactKeyCookie.ctor(String)		0.02%	1,156 ms
Sitecore.Analytics.TrackerSampling.IsSampling(HttpSessionState)		0.02%	1,094 ms
Sitecore.Analytics.Pipelines.StartAnalytics.StartTracking.Process(PipelineArgs)		0.01%	750 ms
Sitecore.Analytics.Tracking.CurrentInteraction.InitializeEventKey		0.01%	625 ms
Sitecore.Analytics.Tracking.TrackingSiteContext.get_EnableTracking		0.01%	609 ms

### Note

Percentages are based on the total time traced.

## Site database servers (Web, Master, Core)

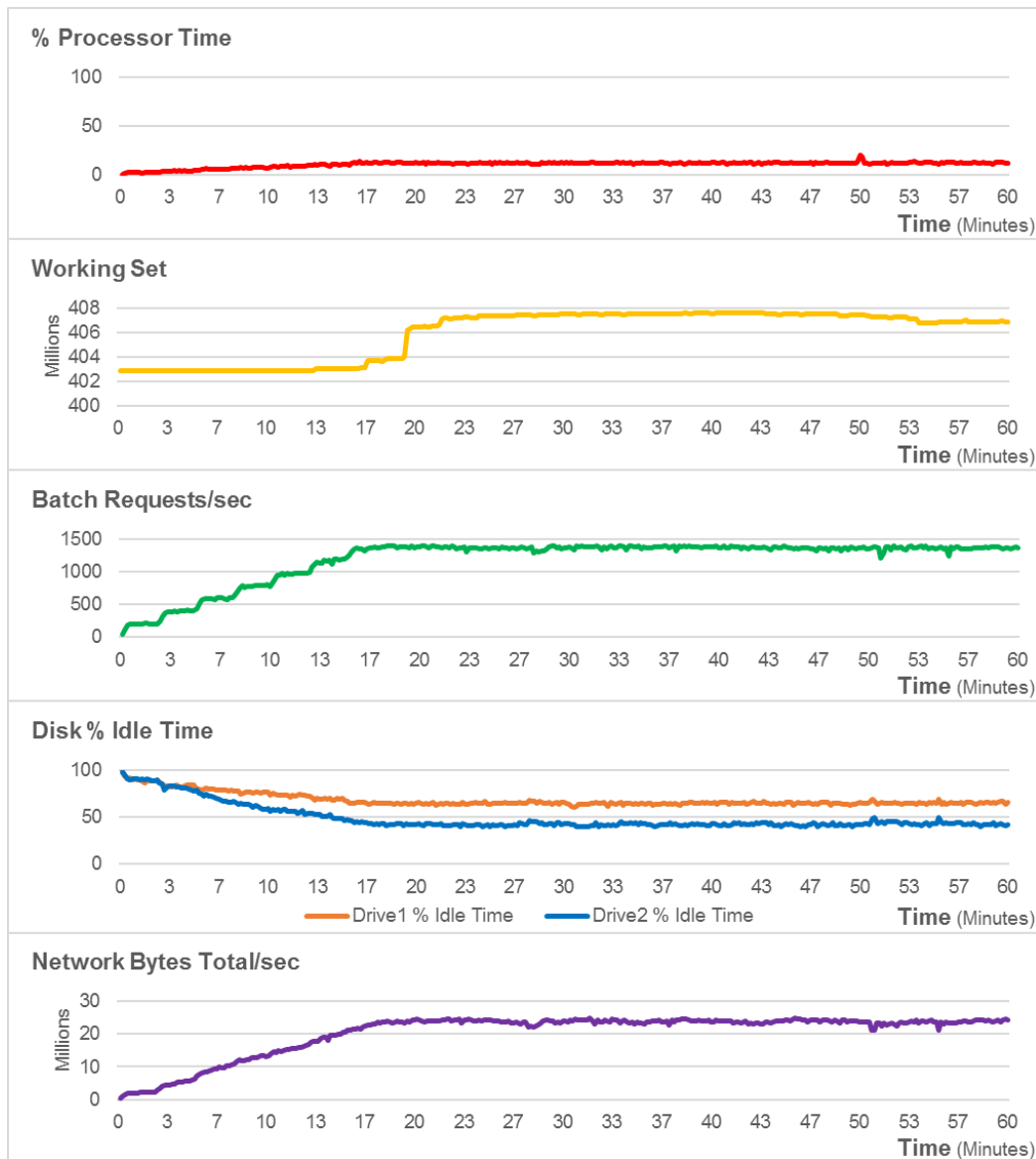


The site database server was more than capable of handling the load generated by the CD servers in the test run.

### Note

The Sitecore Reference Storefront Powered by Commerce Server was configured with the optimal HTML cache settings. This can have a tremendous effect on the database server load.

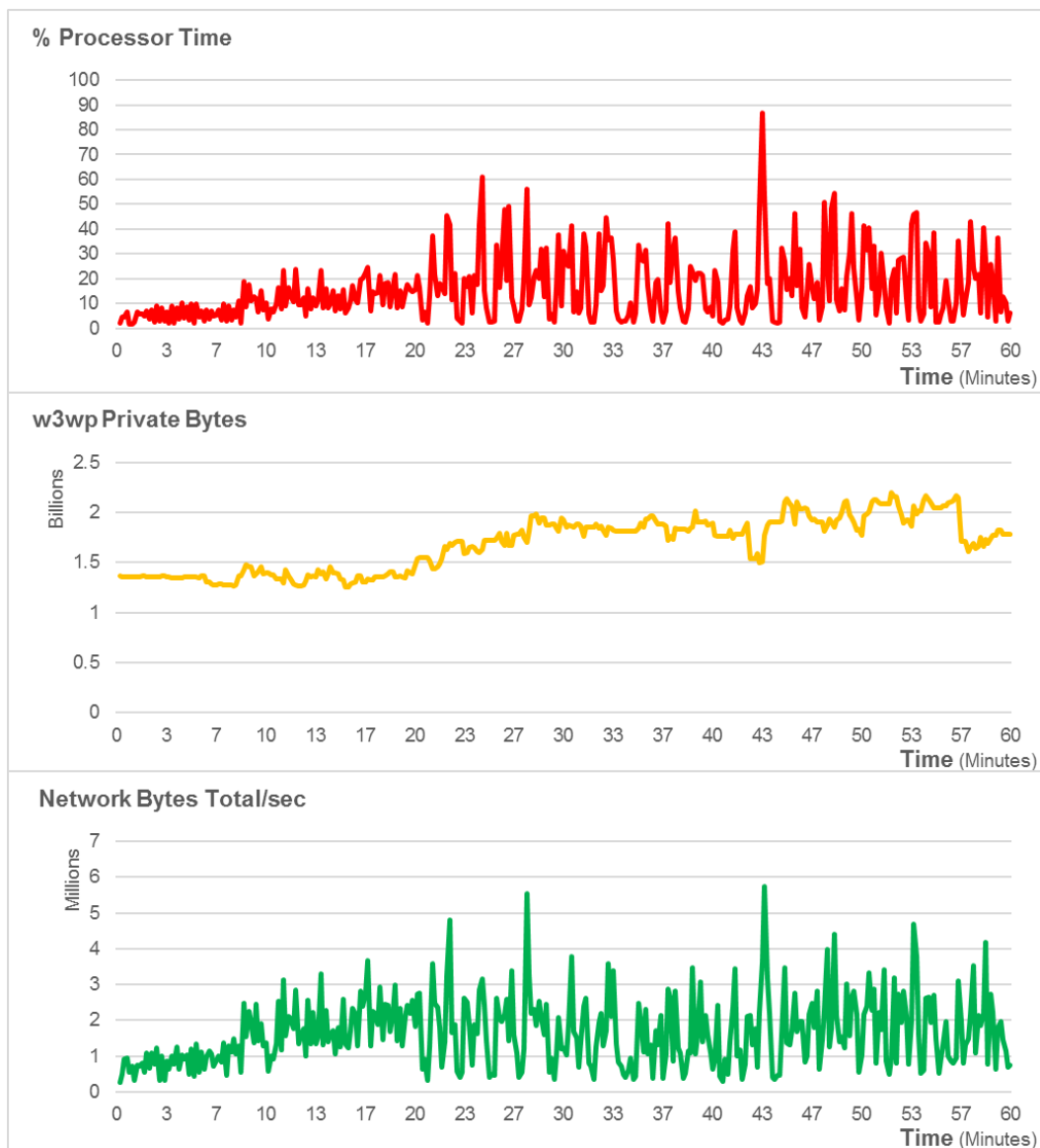
## Session database server



These charts display the processes that take place when the SQL server instance works with the Session database.

The database peak % processor time was 19.9%. The session databases were split onto different disks for private and shared sessions. The session databases used synonyms to point to the data in the temp tables, which greatly optimized the session data usage. This produced a peak of 1,404 batch requests a second during the test run. To configure the process of using synonyms in the session databases, run the following script: `Sessions db performance boost.sql`.

## Processing server

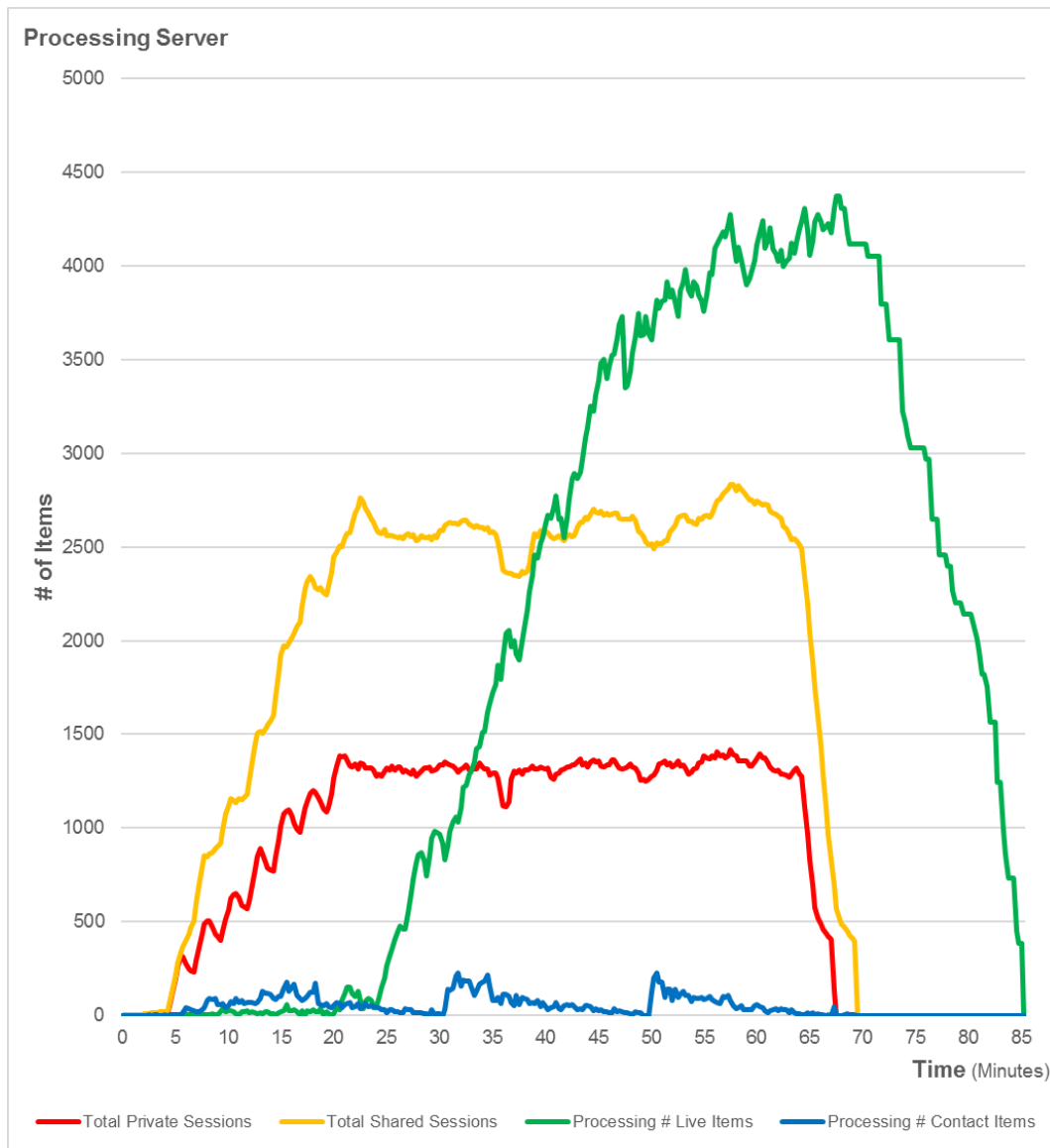


These charts only display the larger processing lane for this test deployment. Processing server usage depends on the rate at which contacts and interactions are created. In this test run, multiple users performed multiple page views and this did not represent the worst case scenario in terms of creating interactions.

In additional testing, which is not described in this whitepaper, it was determined that a load scenario with all new users performing a few page views can have significantly more impact of the processing server(s). When determining the size of the processing tier, if the expected load profile includes many users with a low number of page views per user, you should consider having more processing lanes to ensure that items are processed efficiently during peak usage times.



## Processing: Session and tracking

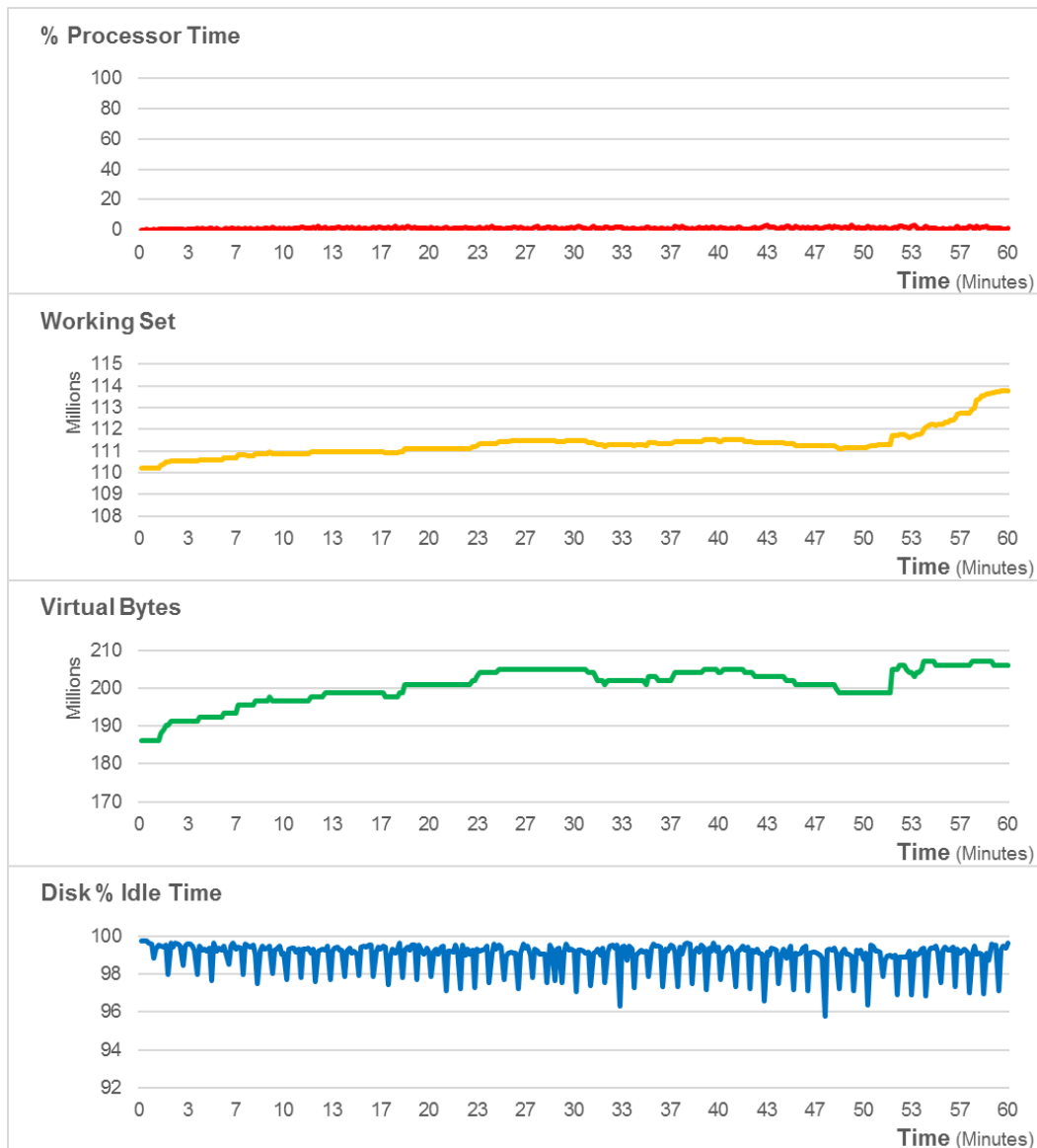


This chart displays the relationship between the session (SQL) and the tracking queues (MongoDB). This chart only includes the two larger processing lanes. The session count is distributed across the seven CD servers. The tracking count includes session expirations from four of these CD servers. Contact processing is much more efficient than live processing, where live processing depends heavily on the capacity of the SQL server that is hosting the reporting database.

The following settings were used for the Aggregation Module:

- Settings file: `Sitecore.Analytics.Processing.Aggregation.Services.config`
- aggregator: MaxThreads set to 16
- contactProcessing: MaxThreads set to 4

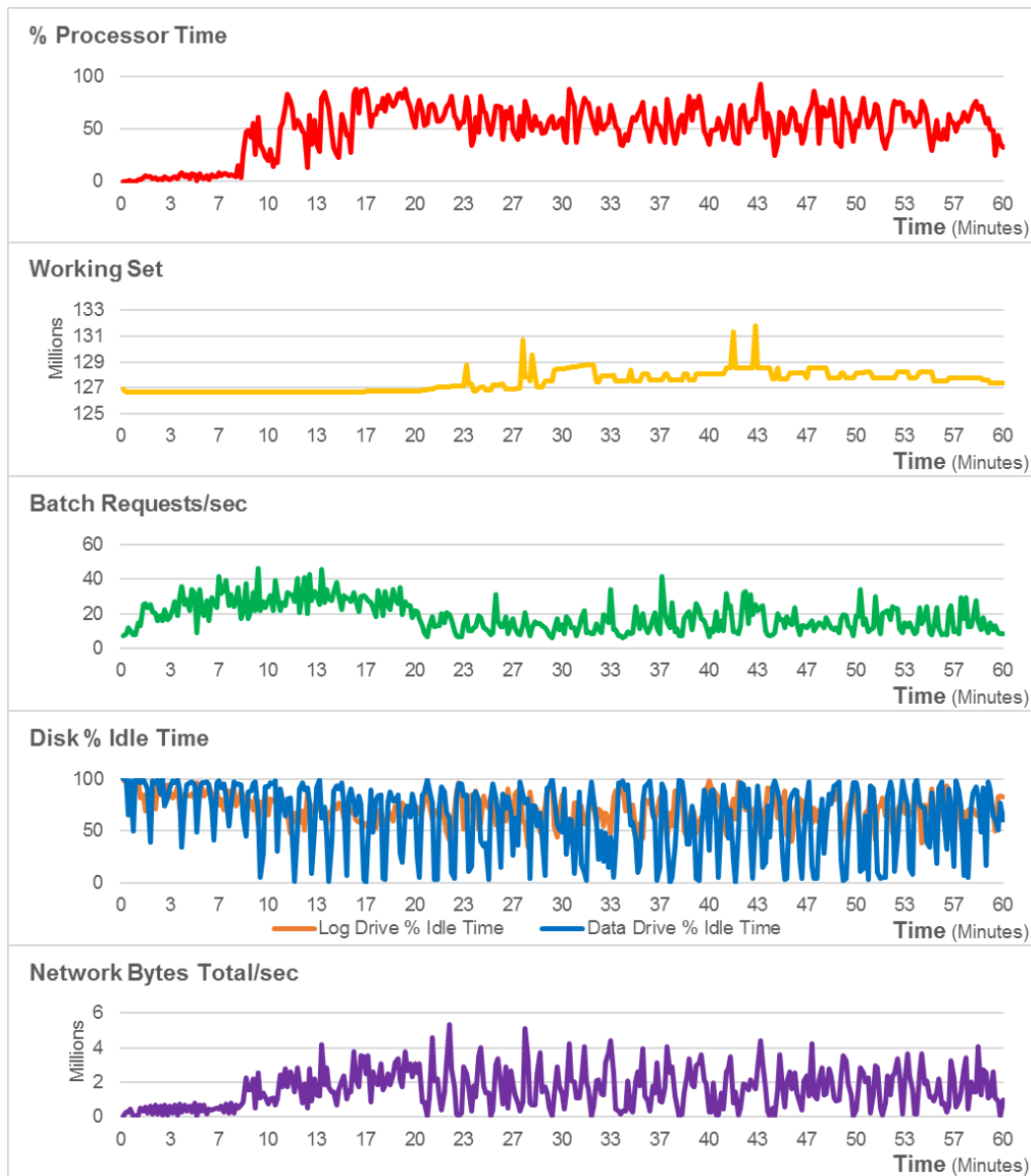
## MongoDB processing server



In our test deployment, we used two processing lanes. One lane handled four CD servers while the other lane handled the remaining three. For the larger lane, the MongoDB tracking databases were hosted on an Azure premium disk with a size of P20. The smaller lane used a disk with a size of P10. These results are averaged across the two server instances.

The average % processing time was not an issue during this test run. Under an extreme load, we found that the CPU usage can spike, which typically indicated that disk sub-system was not able to keep up with the incoming load. Possible solutions for this issue include separating the journal and the data files across separate disks, using the Wired Tiger storage engine, and increasing the number of processing lanes to split the disk writes.

## Reporting database server

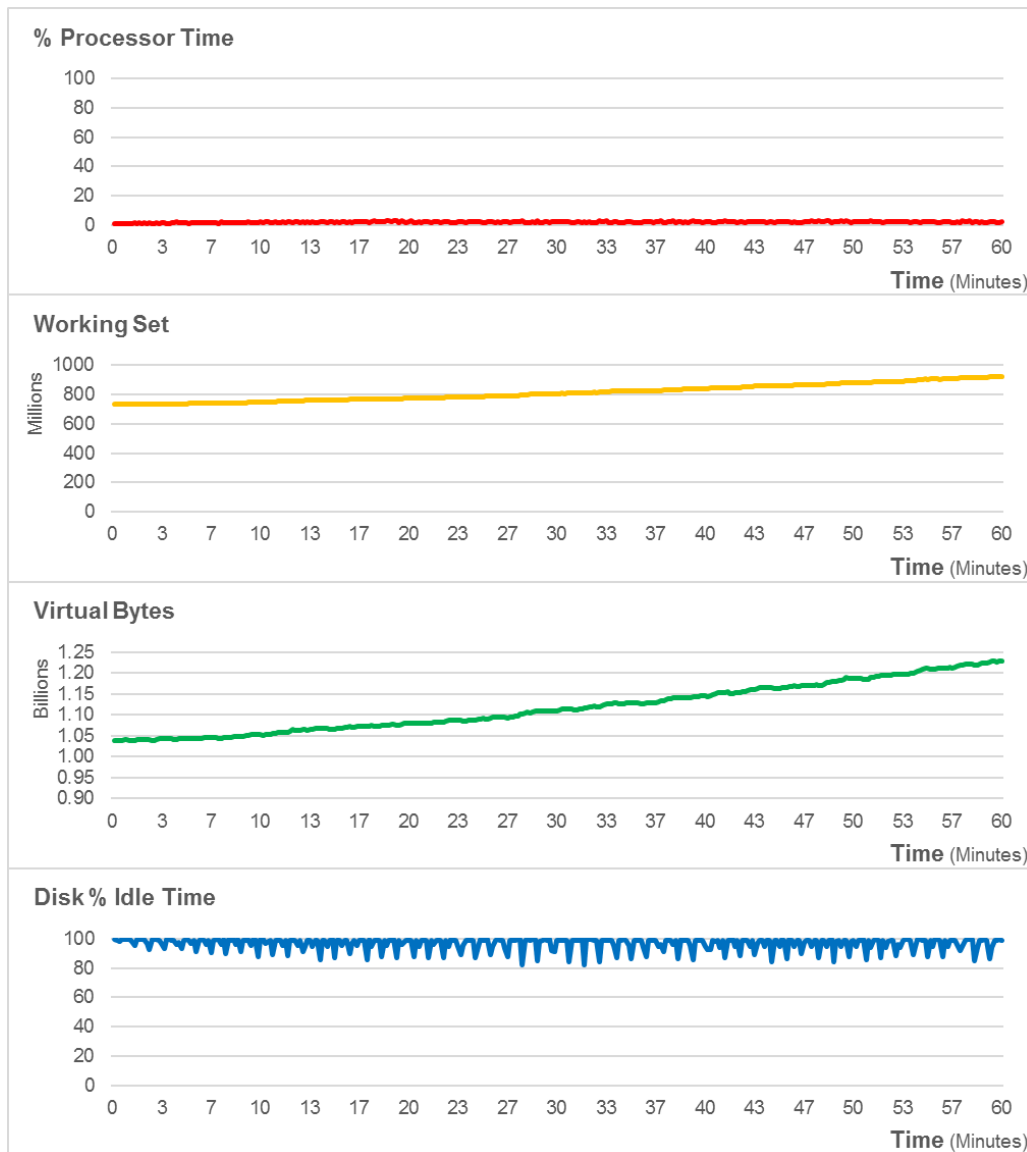


The reporting server was heavily impacted by the processing tier. In most cases, our testing found that the reporting server can limit the total processing throughput capability.

The SQL server configuration followed this guide:

<https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-sql-server-performance-best-practices/>

## Analytics - MongoDB server



To ensure maximum throughput, the primary instances used P30 Azure premium disks.

In this test, CPU usage was not an issue. If there is a high load, the CPU can start to spike, usually indicating that the disk subsystem was not keeping up with the incoming load. If this becomes an issue, the following actions can be taken to increase throughput:

- Increasing the number of shards
- Place the journaling and data on separate disks.

The information contained in this document represents the current view of Sitecore Corporation on the issues discussed as of the date of publication and is subject to change at any time without notice. This document and its contents are provided AS IS without warranty of any kind, and should not be interpreted as an offer or commitment on the part of Sitecore, and Sitecore cannot guarantee the accuracy of any information presented. SITECORE MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

The descriptions of other companies' products in this document, if any, are provided only as a convenience to you. Any such references should not be considered an endorsement or support by Sitecore. Sitecore cannot guarantee their accuracy, and the products may change over time. Also, the descriptions are intended as brief highlights to aid understanding, rather than as thorough coverage. For authoritative descriptions of these products, please consult their respective manufacturers.

All trademarks are the property of their respective companies.

©2016 Sitecore Corporation. All rights reserved.