PARASOFT

TECHNICAL WHITEPAPER

Understanding & Optimizing Parasoft Virtualize Deployments & Performance

Contents

Preface

This whitepaper explores how service virtualization requirements and complexity can impact performance testing environments and resource consumption at scale. Parasoft demonstrates benchmarks based on testing of virtual assets with varying degrees of complexity and throughput and describes best practices as it pertains to resource consumption and achieving production-like fidelity in virtualized environments. Further, Parasoft provides technical recommendations regarding virtual service design, deployment, and infrastructure configurations to meet testing demands. Readers should assess their own service virtualization initiative to sufficiently prepare their virtual performance testing deployment.

Introduction: What Should My Deployment Look Like?

Parasoft is often asked two questions when architecting a deployment of service virtualization.

1. How do I handle n transactions per second during load or performance testing?
2. How many servers do I need in my deployment?

There's no one answer to these two questions. The solution depends on several key factors.

- The behavior that virtual assets are simulating.
- The size of the payloads being passed back and forth.
- The distribution of load across the assets.
- The intended use for the deployed assets.

How Do I Handle N Transactions per Second During Load or Performance Testing?

With simple assets and small payloads, any service virtualization solution should be able to achieve an impressive throughput. However, if your environment simulates complex behavior and/or involves large payloads, you're unlikely to experience this idealized performance during your day-to-day usage on a single server.

To shed light on the issue of service virtualization server performance, Parasoft designed and executed a benchmark study to determine what factors exert the greatest influence over server performance. We then used this benchmark to measure what levels of throughput Parasoft Virtualize is able to achieve under a range of conditions (asset complexity, payload size, server cores). This paper presents the results of that performance benchmarking, then outlines best practices we've developed for optimizing performance within the expected ranges. At the end of this document, we also include a survey that can be used to help understand the performance requirements and influencing factors of the assets being created. Together this information will help you:

- Understand how server throughput is influenced by asset complexity, payload size, and, to a lesser extent, server cores.
- Set realistic expectations for Parasoft Virtualize throughput considering your asset complexity and payload size.
- Perform your own benchmarking to compare the throughput of Parasoft Virtualize with that of other service virtualization solutions.
- Deploy Parasoft Virtualize in a way that supports your throughput requirements.
- Identify and resolve any Parasoft Virtualize configurations that might be preventing you from achieving the desired throughput.

How Many Servers Do I Need in My Deployment?

When determining the size of your Service Virtualization architecture you need to consider more than just the performance of the asset, you also need to consider the workflow for asset creation, validation and deployment. As we talk about the deployment of your Service Virtualization architecture, we will discuss the use of stage and production instances of Parasoft Virtualize, as well as additional instances to support clustering and fail-over. When running in Kubernetes, consider using autoscaling to horizontally scale a cluster on demand.

I. Service Virtualization Performance Benchmark

Benchmark Overview

Parasoft developed the following benchmark to assess service virtualization server throughput as virtual assets are leveraged during load/performance test scenarios.

The Virtual Assets

A set of virtual assets were created to represent a wide range of complexity and payload size.

- **Complexity.** Assets were categorized into high, medium, and low complexity. Factors that influence complexity include the type of correlation—simple request/response matching, parsing XML or JSON data, reading external data sources, and custom scripted operations.
- **Payload size.** Assets were configured to use file-based and in-project response payloads of 250KB.

For a description of each virtual asset used in the benchmarks, see IV. Benchmark Virtual Asset Details.

Load Generation

Parasoft Load Test was used to perform all load tests, running a series of generators on AWS t2.2xlarge instances to drive the load against each of the assets. Each asset was load tested for a duration of 10 minutes with a load of virtual users scaled to the number of cores running on the server: 8 cores/500 VUs, 16 cores/1000 VUs, 32 cores/2000 VUs. Tests were run against virtual assets hosted in a Parasoft Virtualize war deployment, hosted in AWS.

Server Configuration

The results shown in this paper were from tests run on the following AWS instance types:

» Virtualize servers

Model	vCPU	Memory (GiB)	Storage	Network Bandwidth (Gbps)	EBS Bandwidth (Mbps)
c5a.2xlarge	8	16	EBS-Only	Up to 10	Up to 3,170
c5a.4xlarge	16	32	EBS-Only	Up to 10	Up to 3,170
c5a.8xlarge	32	64	EBS-Only	10	3,170

All instances have the following specs:

Up to 3.3 GHz 2nd generation AMD EPYC Processor

EBS Optimized

Enhanced Networking

» Load generation

Instance	vCPU	CPU Credits / hour	Mem (GiB)	Storage	Network Prformance
t2.2xlarge	8	81	32	EBS-Only	Moderate

All instances have the following specs:

Intel AVX, Intel Turbo

Up to 3.0 GHz Intel Scalable Processor

» Parasoft Virtualize memory management set to "-XX:MaxRAMPercentage=70 -XX:InitialRAMPercentage=70"

Memory and Processors

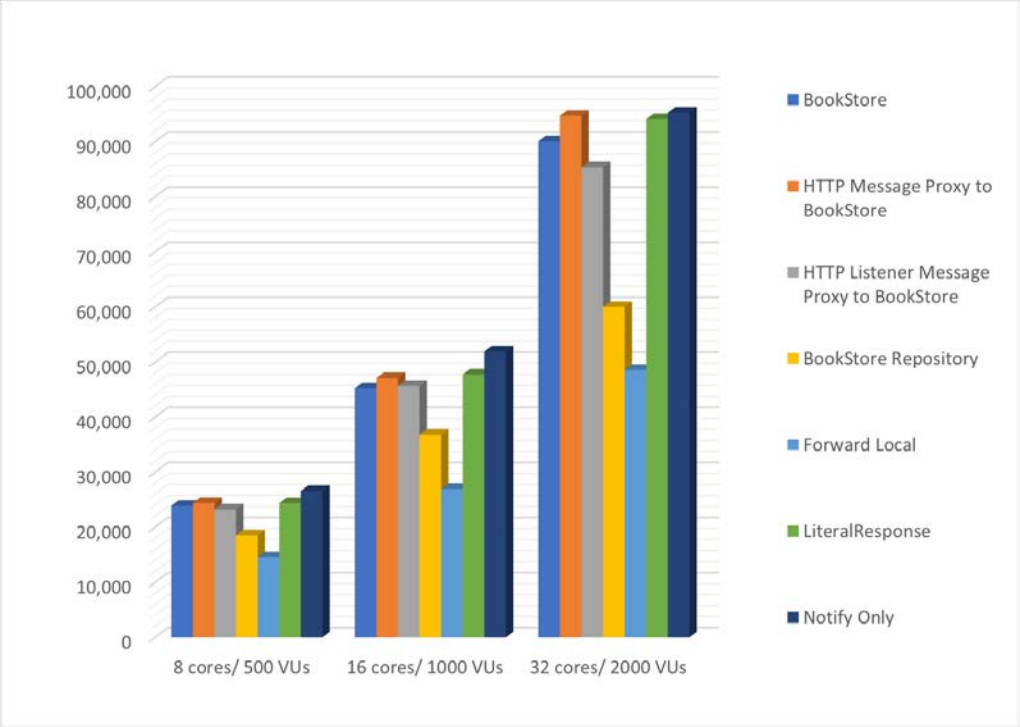
To assess the impact of additional cores, the 150vu/5-minute tests are run on 8 and 32 cores. Increasing the available memory past 4GB does not yield significant performance benefit—the gating factor is processing power.

Benchmarking Results With Parasoft Virtualize

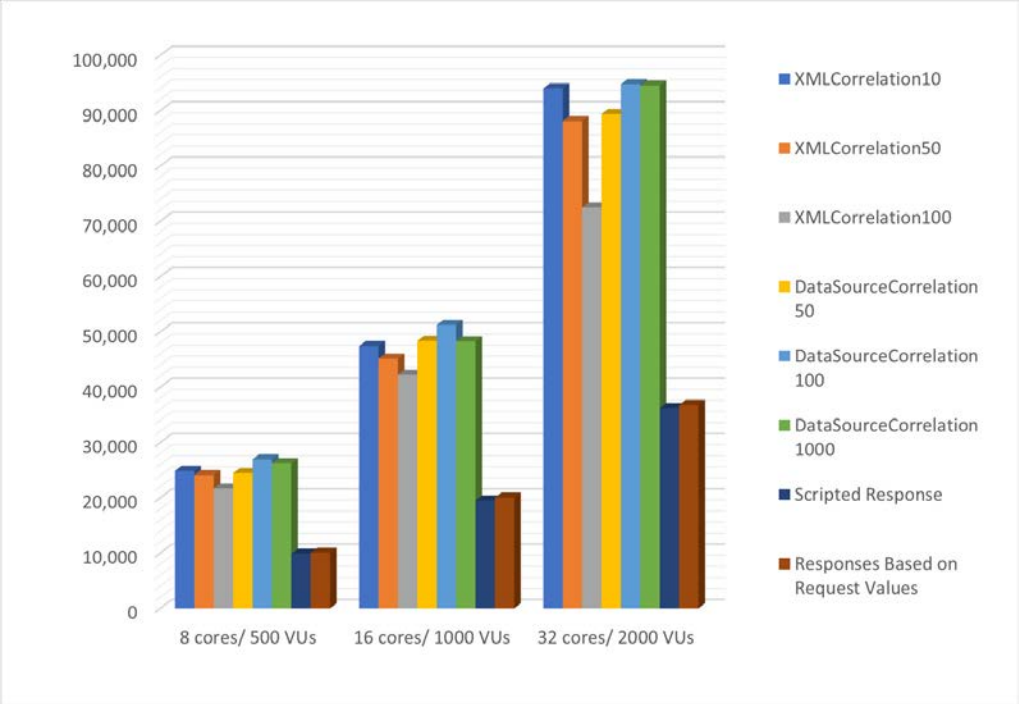
The results achieved with Parasoft Virtualize demonstrate that asset complexity and payload size are the key factors that influence performance. The number of cores also impacts performance if the available memory is above 4GB. All results below are reported in transactions or hits per second (HPS).

Multiple Cores Over 10 Minutes

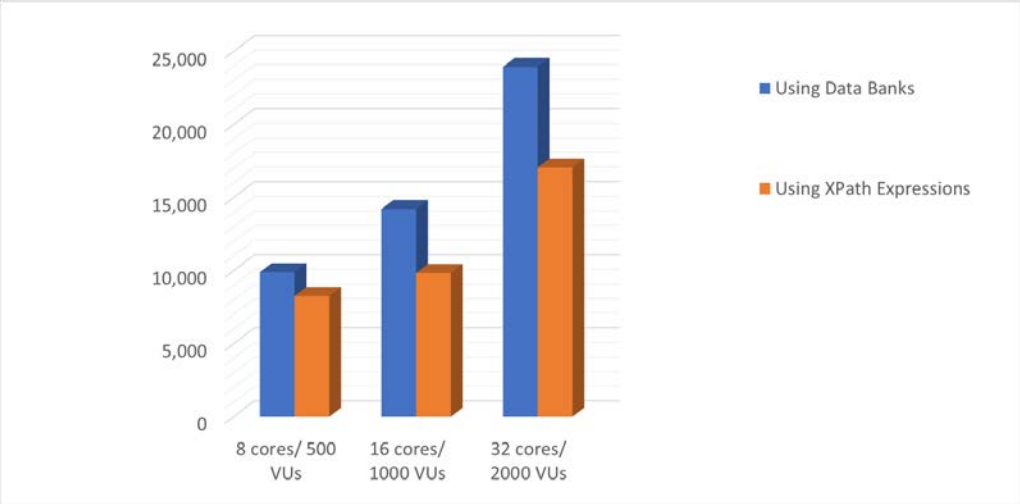
Low Complexity	HPS at 8 Cores/500 VUs	HPS at 16 Cores/1000 VUs	HPS at 32 cores/2000 VUs
BookStore	23,898	45,226	90,057
HTTP Message Proxy to BookStore	24,403	47,116	94,727
HTTP Listener Message Proxy to BookStore	23,243	45,659	85,360
BookStore Repository	18,541	36,820	60,018
Forward Local	14,567	26,969	48,554
LiteralResponse	24,363	47,745	94,114
Notify Only	26,590	51,901	95,292



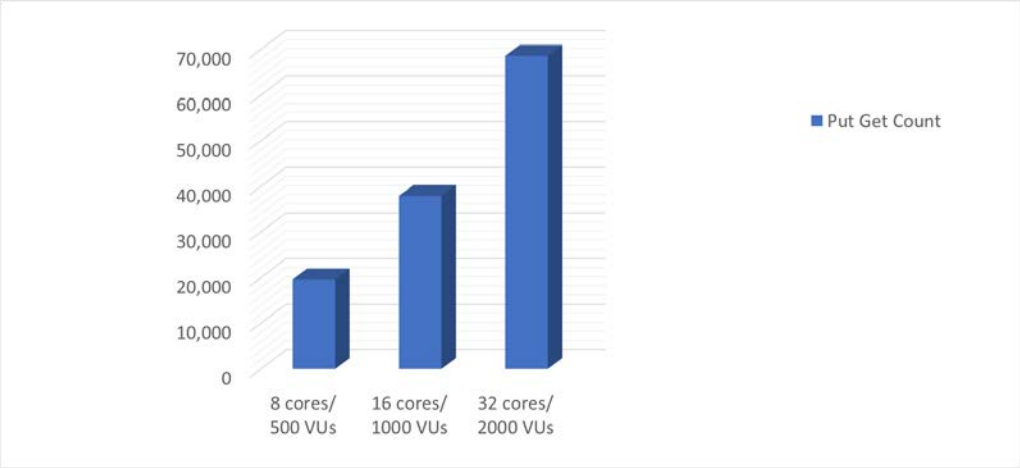
Medium Complexity	HPS at 8 Cores/500 VUs	HPS at 16 Cores/1000 VUs	HPS at 32 Cores/2000 VUs
XMLCorrelation10	24,827	47,422	93,999
XMLCorrelation50	24,092	45,105	88,074
XMLCorrelation100	21,649	42,217	72,461
DataSourceCorrelation50	24,415	48,336	89,361
DataSourceCorrelation100	26,940	51,199	94,748
DataSourceCorrelation1000	26,243	48,247	94,512
Scripted Response	9,872	19,490	36,209
Responses Based on Request Values	10,029	20,019	36,777



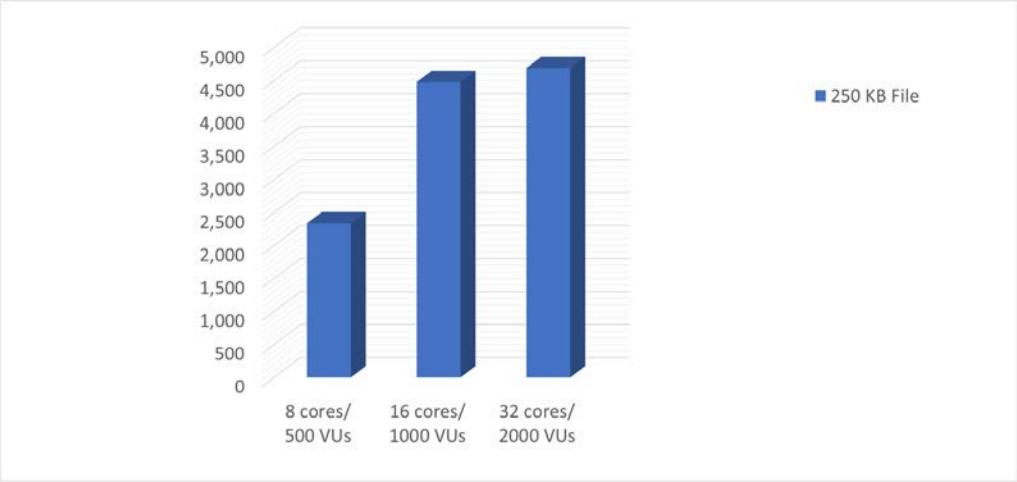
Data Banks and Expressions	HPS at 8 Cores/500 VUs	HPS at 16 Cores/1000 VUs	HPS at 32 Cores/2000 VUs
Using Data Banks	9,886	14,206	23,937
Using XPath Expressions	8,248	9,830	17,046



High Complexity	HPS at 8 Cores/500 VUs	HPS at 16 Cores/1000 VUs	HPS at 32 Cores/2000 VUs
Put Get Count	19,604	37,920	68,589

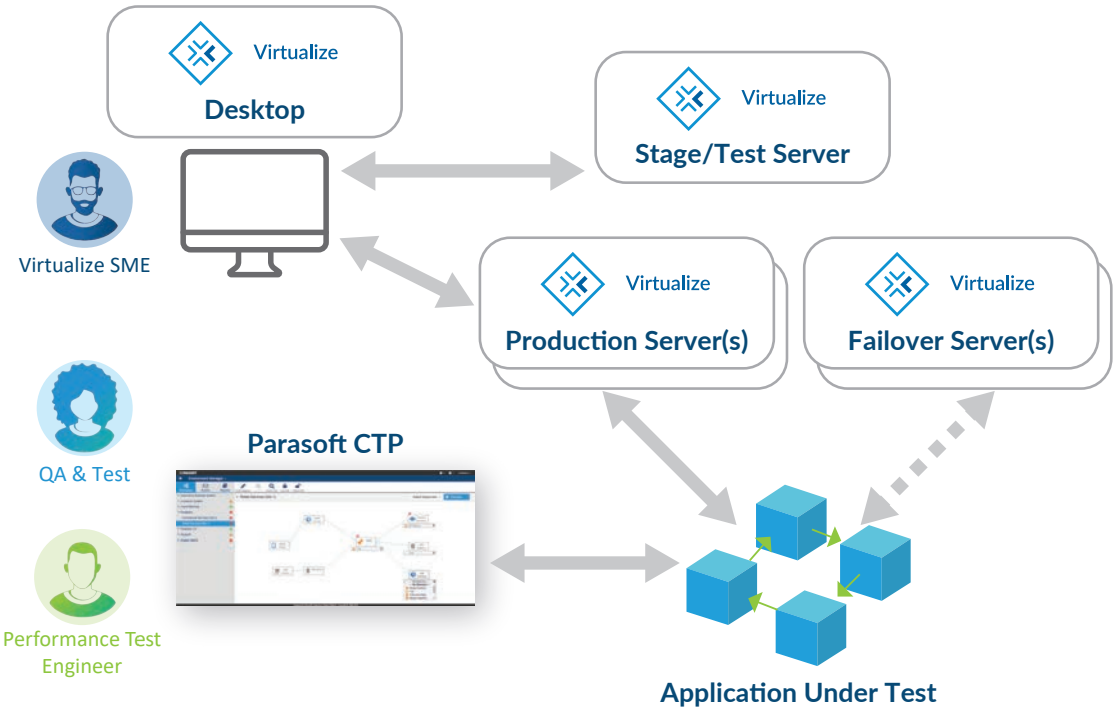


Large Payloads	HPS at 8 Cores/500 VUs	HPS at 16 Cores/1000 VUs	HPS at 32 Cores/2000 VUs
250 KB File	2,337	4,477	4,691



II. Parasoft Virtualize Deployment Recommendations

The Parasoft Virtualize solution consists of desktop and server components. This section describes how the various solution components are leveraged within the typical Parasoft Virtualize workflow, then details hardware and infrastructure recommendations associated with a typical Parasoft Virtualize deployment.



Desktop Components

The Parasoft Virtualize Desktop allows Parasoft Virtualize SMEs (subject matter experts) to manage the Parasoft Virtualize Servers, capture traffic, model and configure assets, and deploy the finalized assets to the Parasoft Virtualize Server.

Virtualize Desktop Requirements

Recommended Hardware	» CPU: Multi-core 2GHz+ » Ram: 2GB free (8GB total system memory recommended) » OS: Windows, Linux, or Mac
Required Software	» Parasoft Virtualize
Recommended Software	» Parasoft SOAtest: Used for testing deployed Virtual Assets » SCM Client: Used for centrally managing Virtual Asset definitions

Server Components

For a deployment of the Parasoft Virtualize solution, Parasoft recommends a minimum of three server machines or VM instances to host:

- Parasoft CTP
- An instance of Parasoft Virtualize to host staged deployment and testing of virtual assets
- An instance of Parasoft Virtualize hosting production assets for consumption by the application/system under test

Additional production instances of Parasoft Virtualize Servers may be required for the following:

- Separation of functional testing and load and performance testing to ensure assets leveraged during functional testing are not impacted by active performance tests and visa versa.
- Clustering/distribution of assets to achieve performance requirements to achieve performance SLAs it may be appropriate to distribute assets across different Parasoft Virtualize instances. Either via manually distributing specific assets or automated distribution of load via clustering.
- Operational failover in the case where a production server may become unavailable, it is advisable to mirror the Parasoft Virtualize instance and deployed assets.

Parasoft CTP Requirements

Recommended Hardware	» CPU: Multi-core 2GHz+ » Ram: 8GB free » OS: Windows or Linux
Required Software	» Parasoft CTP » Web Application Server, like Apache Tomcat 9 » Database: MySQL 8, Oracle 19c, 23c, MariaDB 10.5, HyperSQL 2.7.1

** Disk space depends on the amount of archived transaction history. The above figure represents the typical deployment.*

Parasoft Virtualize Stage Server Requirements

Recommended Hardware	» CPU: Multi-core 2GHz+ (minimum of 4 cores recommended) » Ram: 4GB free (8GB total system memory recommended) » OS: Windows or Linux
Required Software	» Parasoft Virtualize
Recommended Software	» SCM Client: Used for centrally managing Virtual Asset definitions

Parasoft Virtualize Production Server Requirements

As discussed previously, performance and therefore hardware recommendations for a Parasoft Virtualize Server depend on a number of factors:

- Size of the response payloads
- Required throughput
- Required level of concurrency
- Complexity of the virtual assets deployed
- Number of virtual assets deployed on the server

As part of a Parasoft Virtualize deployment, Parasoft representatives can assist an organization in determining the correct deployment strategy. As an initial guideline, below is the recommended baseline hardware configuration:

Baseline Hardware Specification	» CPU: Multi-core 2GHz+ (minimum of 8 cores recommended) » Ram: 8GB free (16GB total system memory recommended) » OS: Windows or Linux
Required Software	» Parasoft Virtualize
Recommended Software	» SCM Client: Used for synching Virtual Assets on Virtualize Server

III. Tips for Achieving Optimal Performance From Parasoft Virtualize

As the number of deployed Virtual Assets increases and as the amount of traffic being handled by the Virtualize server increases, the performance of Virtualize can degrade. This section defines best practices to use when building, deploying, and running Virtual Assets so that they have the best possible performance.

Virtual Asset Optimization

There are several things to look for in a virtual asset file that can reduce its performance.

Unused Message Responders

If a responder is added to the Responder Set, Virtualize will look at its Responder Correlation for each incoming request. If no requests are being sent to this Virtual Asset that would match, this uses CPU time unnecessarily.

How to identify: Monitor the Virtual Asset and run many use cases. Look for Responders that are defined but not listed in the Event Log.

How to fix: Disable or remove the Responder. Alternatively, move it to the bottom of the Virtual Asset, after a catch-all.

Virtual Assets Used as Proxies

Using a Message Forward Tool as a Message Proxy to route to other Virtual Assets is less performant than using a Message Proxy.

How to identify: Anywhere that a Virtual Asset (.pva) is being used as a proxy – usually it will contain a single Responder with a chained Message Forward Tool.

How to fix: Replace Forward Tools with either a direct Virtual Asset endpoint or a Message Proxy or replace a .pva proxy with a Message Proxy. Message Proxies are roughly 25% faster than .pva proxies and use less memory.

Unused Data Source Rows

Like with responders, Virtualize searches Data Sources from the first row until it finds a matching value. Rows that are never used are still checked, so removing unused rows of data can improve performance, especially if there is a large number of them (over 10,000).

How to identify: Identify all the data used in the use cases that the Virtual Asset supports.

How to fix: Go through the data source and find rows that are not used by the use cases and remove them.

Ordering of Responders and Data Source Rows

Virtualize always searches sequentially through responders and data sources to find a match. Placing the most commonly used responders/operations and data source rows at the top can have a significant impact on performance.

How to identify: Use monitoring on the Virtual Asset to see which responders are showing up the most often.

How to fix: Move the responders that are used most often to the top of the .pva file. For data source rows, identify which use cases are the most commonly run, then identify the data source row that correlates to them and move them to the top.

Inappropriate Response View

Each view in the Response tab takes a different amount of processing time to build a response. Using an appropriate one can improve response times.

How to identify: The Response tab uses a Form view, but no values are scripted.

How to fix: If using a static response, use a Literal view. This is the fastest. If using Multiple Responses, try placing the most common responses at the top of the list, and using simple Conditions whenever possible. If using Multiple Responses and only 1 response is in the list, switch to Literal View.

Inefficient Responder Correlation

When Virtualize is looking for a responder to use, it checks the correlation for each responder against the incoming request message. Optimizing these correlations can have a performance impact.

How to identify:

Check if:

1. Responder correlation sets a request body correlation when a path, transport, or method correlation would work be.
2. Multiple correlations are enabled but not needed to uniquely identify an operation.

How to fix:

Use a responder correlation on path, transport, or method. Example, for a RESTful request setting a correlation on the resources HTTP method and URL path is faster than using a correlation on the request body.

Unused Chained Tools

Chaining tools is a powerful feature in Virtualize, especially while debugging a virtual asset. However, leaving these tools in the virtual asset can have a performance impact, so removing unused ones can reduce memory overhead and CPU usage.

How to identify: Look for data banks and transformers with no extractions, extension tools with debugging statements (Application.showMessage), and other tools that don't serve a purpose in the functionality of the virtual asset. Edit tools, validation tools, file writers, and the like can all have a potential performance impact.

How to fix: Disable or remove the unused tools.

Inefficient Custom Transport Extensions

When using a custom listener in Virtaulize, be sure that the implementation does not invoke `IMessageHandler.handleMessage(…)` from threads that are created and destroyed repeatedly. Virtualize responder internals relies on caching some resources on a per-thread basis, so if a new thread is created for each request message or if the threads are recycled too often then such per-thread caching optimizations will fail to boost the throughput. If custom transport implementations are multi-threaded, then it is highly recommended that threads are pooled and reused in order to avoid cache thrashing within Virtualize.

Additional Optimization Tips

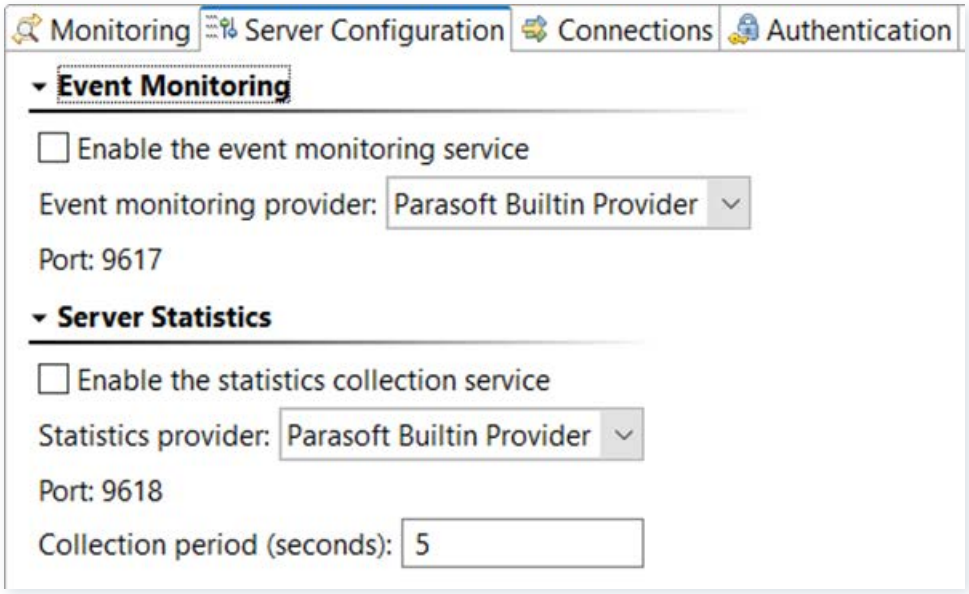
Aside from optimizing the .pva, there are other ways to configure Virtualize which will also improve performance.

Disable Event Monitoring

Always disable event monitoring on the server before starting a load test. Event monitoring is a debugging tool to help when creating virtual assets and as such has a high overhead that will slow down performance tests.

To completely disable monitoring do the following:

1. In the Virtualize Server view, double-click your remote server and click the **Server Configuration** tab.
2. Disable **Enable the event monitoring service** and **Enable the statistics collection service**, then save.



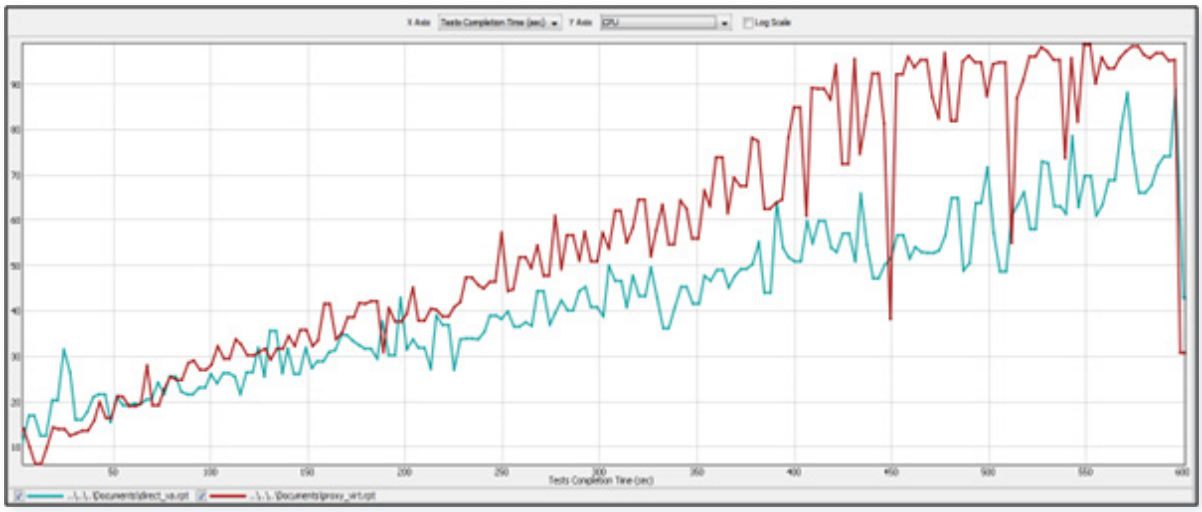
3. Stop and re-start the server.

Note: Remember to re-enable monitoring after the load test is complete if it is needed. However, in general, Event Monitoring in Virtualize should be enabled on an as-needed basis in any environment where performance is a concern.

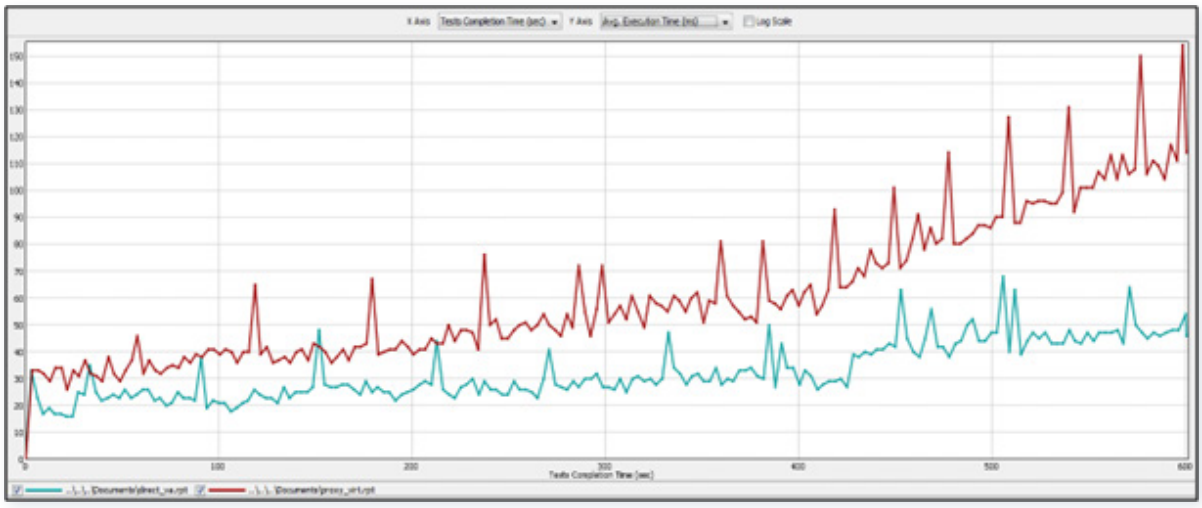
Identify Which Virtual Assets Are Causing Bottlenecks

When running into performance issues, it is always helpful to empirically identify which virtual assets are having the biggest impact on performance. To do this, try running small load tests for several use cases and tracking performance statistics. If a particular asset is performing poorly, focus on optimizing that one first.

For instance, Parasoft Load Test can gather performance statistics for each step in the use case.



In this graph, CPU usage was monitored while the TPS was steadily increased during two separate load tests—one in blue, and one in red. For one operation, CPU usage maxed out earlier. This indicates that this operation is one that should be further optimized.



Optimize Load Generation Tools

Load generation tools can be a significant factor when measuring and assessing throughput. Be sure that your load generation tool is not a bottleneck. Very often options such as real-time message validation on all messages, metric collection, and real-time graphing options can significantly impact the throughput that the tool can generate. In most high-throughput testing scenarios, configuration factors in load generation tools need to be accounted for or else inaccurate results would be obtained.

For example, Parasoft Load Test provides a range of options that configure the range of detail verbosity in different ways, including whether to record graphs for the metrics that are being tracked, individual hits, hit details and thresholds on the number of error details to store during a load test run. It has the ability to orchestrate load generation from multiple instances and multiple machines with or without response validation (high-throughput mode). The difference in throughput between the high-detail verbosity and the lowest with high-throughput mode can vary with a 10x order of magnitude. For example, a large test suite might reach ~50 HPS with validation and details, but over ~1000 HPS in high-throughput mode.

The best strategy when using Parasoft Load Test is to run at least one load generator instance (load test server) and one controller (more generators as needed to increase load), then configure the generators to run in high-throughput mode (unverified) at high load intensity and the controller instance to run with message verification, graph recording and error details, but with much smaller load intensity. In that configuration, the generators serve the bulk of the load while the controller plays the role of a probe or a metric and verification sampling engine. Load Test will tally the metric statistics in a fashion that includes confidence levels based on the ratios between verified and unverified traffic ratios. This strategy allows maximizing the amount of load that is generated without compromising validation and details of recording.

Operating System Tuning and Network Impact

Close Other Running Processes

It is generally a good idea to check the Virtualize Server for other applications running, immediately before running a load test, especially if they are known to use CPU and/or Memory resources. This recommendation applies to any cases where fast performance is a requirement.

System Tuning

Windows

If using Windows, there are some parameters you can tune in the Registry:

1. Click Start > run > regedit.
2. Browse to HKEY_LOCAL_MACHINE > SYSTEM > CurrentControlSet > services > Tcpip > Parameters and set the MaxUserPort to 7FA6 (32678 decimal).
3. Reduce the TcpTimedWaitDelay to 1E (30 decimal).
4. Restart the machine for changes to take effect.

This will increase the number of allowed concurrent TCP connections in Windows (MaxUserPort) and reduce the amount of time that an unused TCP port is reserved before it can be reused (TcpTimedWaitDelay)[1].

[1] *"Configuring Windows for high network connection rates", IBM CICS Transaction Gateway. http://publib.boulder.ibm.com/infocenter/cicstg/v6r0m0/index.jsp?topic=%2Fcom.ibm.cicstg600.doc%2Fcclla l0264.htm*

Linux

It may be necessary to tune your Linux server for high throughput. The following are some system settings that could be helpful. These should be applied to Virtualize and the load generation tool.

```
ulimit -u 4096
```

This sets the maximum number of processes available for a single user, which is important since Java threads are processes. Having this number too low could result in the inability to spawn new threads.

```
sysctl -w net.ipv4.tcp_tw_reuse=1
sysctl -w net.ipv4.tcp_tw_recycle=1
```

These allow the operating system to quickly reuse and recycle network sockets. This is important while under heavy load as sockets can be used and discarded frequently.

```
sysctl.conf
```

You may need to tune tcp buffers. Based on your network configuration there are several sysctl parameters that can be modified. Refer to http://fasterdata.es.net/host-tuning/linux/ for more information.

Network Bandwidth

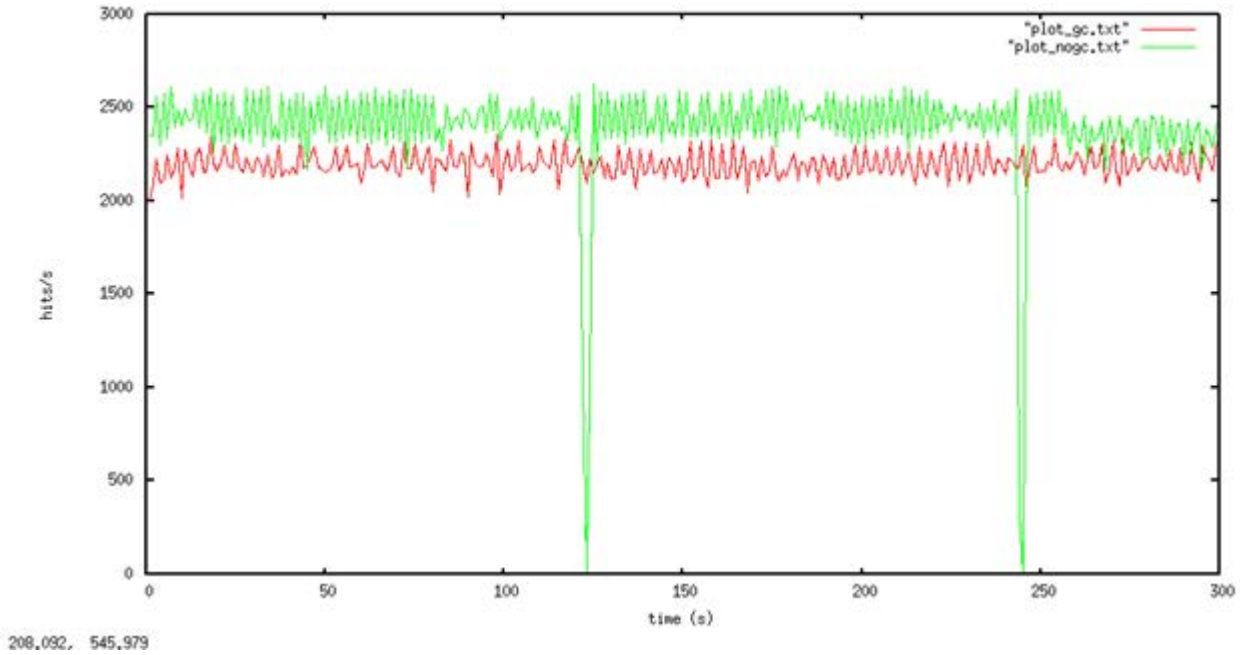
In most cases, network latency within the same subnet is not a concern since application processing time is usually a much bigger factor. However, throughput can be significantly impacted by the message sizes. For example, a machine hosting Virtualize that has a network bandwidth of 100Mbps can process up to 12.5 MB/s or 12,800 KB/s. So, if Virtualize processes request/response message pairs that total 1 KB in size (1 KB per hit) then the maximum theoretical throughput that can be achieved on that machine would be 12,800 HPS. However, if the request/response message sizes are 250 KB per hit, then the maximum possible throughput becomes 12800/250 = 51.2 HPS.

Tuning the Parasoft Virtualize Application

You can also improve the performance of the Virtualize application.

Java Garbage Collection

Java garbage collection (GC) can have a profound effect on performance. The default garbage collector is tuned for higher average throughput, at the expense of occasional “full-stop” garbage collection sweeps. The JVM has many GC parameters that can be tuned for your system.



In this graph, we see the throughput over time—higher is better. The green line shows the default garbage collection behavior, while the red line shows a garbage collection scheme with lower average throughput, but more stable performance. The Virtualize GC parameters used for the red line were:

```
-J-XX:+UseConcMarkSweepGC -J-XX:+DisableExplicitGC
-J-XX:+UseCompressedOops -J-XX:NewRatio=1
```

Java Heap Size

Our performance tests on most PVAs indicate that a maximum 4GB of heap size is sufficient as long as response message sizes configured in the PVAs are under 200KB in size. In the cases where consistency in response times is desired, it is recommended that the initial heap size is set to the same amount as the maximum heap size in order to minimize the impact of the JVM increasing the heap by Virtualize when under load, because such increases in heap size by the JVM result in dips in throughput and response times. In cases where that is not a concern, it is suitable for most cases to use the following settings:

```
-J-Xms2g -J-Xmx4g
```

Tomcat Thread and Connection Pools

Tomcat keeps a pool of processing threads, used for processing requests, and connections. The number of connector threads is 150 by default. When a connection is made, a processing thread is assigned to it. If Tomcat runs out of processing threads, the connection will wait until one is available. But if too many incoming connections are attempted (Tomcat runs out of threads and connections), then further connection attempts will be refused.

If you plan to apply a level of concurrency that is below 150 concurrent connections (concurrent users), then no changes should be made in the Tomcat configuration settings. If the concurrency level is above 150, then the pool size can be increased in order to see if there is gain in throughput or response times. In general, make sure that other potential bottlenecks have been addressed before changing Tomcat server parameters. Consider increasing the Tomcat connection pool size if you experience "connection refused" errors.

Please note that increasing the number of threads will increase the "warm up" time that Virtualize requires in order to reach its full performance capability. See the Virtualize Warmup Time section for more details.

Increase the size of the thread and connection pools with these steps:

1. Browse to <TOMCAT_HOME>/conf
2. Edit server.xml and look for a section that looks like this:

```
<Connector
   URIEncoding="UTF-8" connectionTimeout="20000"
enableLookups="false"
   name="default" port="9080" protocol="HTTP/1.1"
redirectPort="9443"
   server="Parasoft Server"/>
```

3. Add the acceptCount and maxThreads parameters to the Connector element. For example:

```
<Connector
   URIEncoding="UTF-8" acceptCount="300" connectionTimeout="20000"
   enableLookups="false" maxThreads="750" name="default"
port="9080"
   protocol="HTTP/1.1" redirectPort="9443"
   server="Parasoft Server"/>
```

This tells Tomcat to keep a processing thread pool of 750 threads, and queue up to 300 connection attempts. Theoretically, you could handle up to 1050 connections at once with these settings.

Virtualize Warmup Time

When a PVA is first deployed in Virtualize, that PVA will take time before it performs well. This is due to various caching and pre-compiling strategies that are used in Virtualize, which favor optimization for high throughput over long durations of time over instant performance immediately after deployment.

The warmup times can range from a few seconds to a few minutes. The warmup times are highest when heavy use of scripting (Groovy or Jython in extension tools or scripted Form XML/Form Input parameters) is used in the PVAs, because Virtualize will need to cache the script execution engines for each thread.

For example, if a PVA has a total of 50 scripts, and Tomcat is setup with 150 threads, Virtualize will need to make 50x150=7,500 script engine initializations in order to accommodate each thread with its own script execution engine. This is needed in order to maintain script execution integrity under concurrency. In this case, Virtualize will need to process a minimum of 7,500 requests (one per thread per script) before they are all cached.

Applying load scenarios that ramp up the concurrency or the hit rate levels reduces warmup times.

IV. Benchmark Virtual Asset Details

The following virtual assets were used for the performance benchmarking covered in I. Service Virtualization Performance Benchmark.

Low Complexity	
BookStore	Simple response is specified via Form Input GUI.
HTTP Message Proxy to BookStore	A simple Message Proxy that forwards to the BookStore Virtual Asset.
HTTP Listener Message Proxy to BookStore	A simple HTTP Listener running on a separate port that forwards to the BookStore Virtual Asset.
BookStore Repository	Simple response parameterized using Data Repository.
ForwardLocal	Acts purely as a pass-through that forwards traffic.
LiteralResponse	Correlation based on SOAPAction or other headers; response is specified as literal input.
Notify Only	HTTP response but no response payload.

Medium Complexity	
XMLCorrelation10	Correlation based on XML (10 request/response pairs).
XMLCorrelation50	Correlation based on XML (50 request/response pairs).
XMLCorrelation100	Correlation based on XML (100 request/response pairs).
DataSourceCorrelation50	Parameterized responses with data source correlation (50 matching data rows).
DataSourceCorrelation100	Parameterized responses with data source correlation (100 matching data rows).
DataSourceCorrelation 1000	Parameterized responses with data source correlation (1000 matching data rows).
Scripted Response	Response is determined by a scripted data value (e.g., a derived value based on input).
Data Banks and Expressions	
Using Data Banks	A large request payload that extracts numerous values from the request and parameterizes the response.
Using XPath Expressions	A large request payload that uses inline expressions to parameterize the response.
High Complexity	
Put Get Count	Exhibits composite/stateful behavior (e.g., incrementing counter, order information) using stored data via put/get scripting calls.
Large Payloads	
File-Based Payloads	Uses response payloads of 250KB. Payload was specified as a file on disk.
Inline Payloads	Uses response payloads of 250KB. Payload was specified by copying file content directly into the virtual asset.

V. Parasoft Virtualize Deployment Survey

The purpose of this Deployment Survey is to capture system information to help with determining the correct deployment model for Parasoft Virtualize to meet performance needs.

Background (Optional)

- Company Name:
- Project:
- Primary Technical Contact:

Connectivity/Networking

- How many clients/applications will be accessing the virtual assets?
- What protocols are used when communicating with the Virtualize Server? (HTTP, REST, SOAP, and so on)
- What security mechanisms are used?
- What is the network bandwidth, latency and number of hops between the clients/applications and the Virtualize Server? For example, local network with 100 Mbps connection, 100ms ping time, through two routers.

Payloads

- What are the payload content types? For example, JSON, ascii, binary, XML.
- Are payloads encrypted?
- What are the average and maximum/peak request payload sizes?
- What are the average and maximum/peak response payload sizes?

Asset Structure and Behavior

- How many active Virtual Assets/PVAs will be deployed at any one time?
- What is the breakdown of active assets based on protocols used? For example, 75% HTTP, 25% IBM MQ.
- How many responders will be within each PVA?
- What is the breakdown of asset complexity? For example, 30% low, 50% medium, 20% high. See Benchmark Virtual Asset Details for guidance.
- What is the breakdown of data source types used? For example, 80% Excel, 20% CSV.
- What is the average and maximum size of data source used?

- What scripting is used? For example, Java, JavaScript, Jython, Groovy.
- What percentage of your assets use the following?
  - Values passed from Request to Response
  - Data Source Correlation
  - Scripted Values
  - Hierarchical Data Sources
  - CRUD modeling techniques

Performance SLA

- What are the observed and required average and maximum/peak hits per second?
- Are delays used in the virtual assets? If so, what durations and what delay calculation types are used?
- How many concurrent connections are expected? For example, 2000 TPS / 100ms response time = approximately 200 connections.

Existing Virtualize Server Details (if Appropriate)

- Operating System:
- Number of CPUs:
- Number of Cores per CPU:
- Amount of physical RAM installed on the machine:
- Amount of available RAM—RAM not allocated prior to launching Virtualize:
- Hard-drive total size and free space:
- Are applications other than the Virtualize Server (including Virtualize Desktop or Parasoft CTP) running on this machine?
- If so, what are they?

TAKE THE NEXT STEP

Contact our experts with further questions about preparing your virtual performance testing deployment.

About Parasoft

Parasoft helps organizations continuously deliver high-quality software with its AI-powered software testing platform and automated test solutions. Supporting the embedded, enterprise, and IoT markets, Parasoft's proven technologies reduce the time, effort, and cost of delivering secure, reliable, and compliant software by integrating everything from deep code analysis and unit testing to web UI and API testing, plus service virtualization and complete code coverage, into the delivery pipeline. Bringing all this together, Parasoft's award-winning reporting and analytics dashboard provides a centralized view of quality, enabling organizations to deliver with confidence and succeed in today's most strategic ecosystems and development initiatives—security, safety-critical, Agile, DevOps, and continuous testing.