

Are Noisy Neighbors in Your Data Center Keeping You Up at Night?

Control virtual-machine resources with Intel® Resource Director Technology

All businesses, big or small, are looking for cost-effective IT solutions to help them create flexibility as their businesses grow. Toward that end, many businesses are moving their compute and storage resources to the cloud. A typical cloud hosts multiple customer applications on the same physical infrastructure, which can lead to a “noisy neighbor” syndrome. This occurs when a few applications use disproportionately large amounts of shared resources and negatively impact the performance of other applications within the same infrastructure. This type of interference can occur on almost any type of shared resource, including CPU, storage, network, and even software components. Cloud service providers (CSPs) often try to mitigate this problem by restricting the amount of resources that a customer application can use to avoid oversubscription of resources.

Ensuring consistent performance and quality of service (QoS) is one of the most important tasks for most CSP operators, and noisy neighbors can impede QoS. To better manage noisy neighbors, it is crucial for CSPs to have sufficient monitoring and control mechanisms in place to detect and address noisy-neighbor interference at a more granular level. Intel® Resource Director Technology (Intel® RDT) offers such mechanisms to monitor and control interferences that occur on the processor.

Interference on the Processor

Like any shared resource, there can be interference on the microprocessor that hosts multiple applications. To minimize the interference, administrators typically allocate dedicated cores for applications and avoid oversubscribing the cores. But even then, there are common subsystems on the processor that are shared between the cores. Interference on these subsystems, especially the last-level cache (LLC) and integrated memory controller, can have a severe impact on the end-user experience. It takes significantly more time to access data from memory if the data is not found in the cache. This gets worse if there is also contention in accessing memory. Together, these issues bring down the overall application performance and worsen the end-user experience.

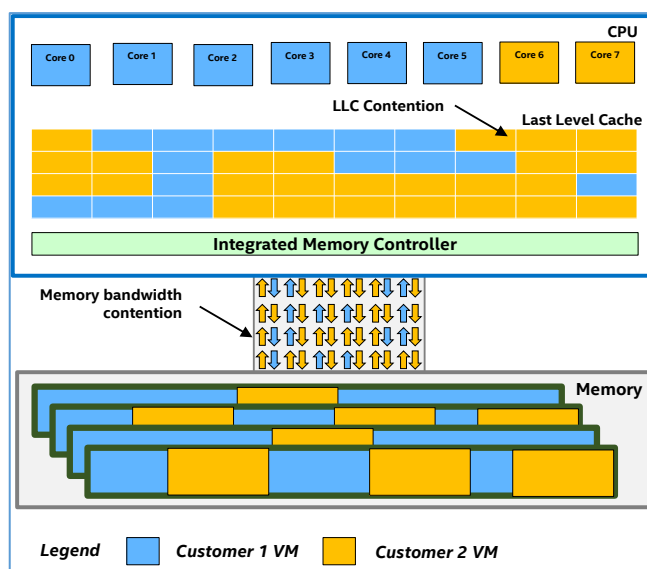


Figure 1. Cache and memory interference between applications

Figure 1 illustrates such an interference scenario. It shows a multi-core processor with eight cores hosting applications from two different customers. Each virtual machine (VM) has its own CPU cores, but the VMs share the LLC and memory bandwidth. The LLC usage and memory access of each customer’s application is illustrated using different colors.

Intel Resource Director Technology

Intel® RDT is an advanced resource monitoring and control feature set that is designed to improve visibility into, and control over, usage of shared platform resources such as memory bandwidth and LLC. It is featured on the Intel® Xeon® processor family as part of Intel's multi-year initiative to provide more insight into, and granular control over, shared resources. Future Intel® technologies will continue

to advance noisy-neighbor detection and mitigation beyond what is currently available. Intel® RDT currently supports the following features:

- Cache Monitoring Technology (CMT) helps you find a noisy neighbor over-utilizing the last-level cache (LLC). Read more about CMT at the Intel® Developer Zone: <https://software.intel.com/en-us/blogs/2014/06/18/benefit-of-cache-monitoring>
- Memory Bandwidth Monitoring (MBM) helps you find which noisy neighbor is using too much memory bandwidth. Read more about MBM at the Intel Developer Zone: <https://software.intel.com/en-us/articles/introduction-to-memory-bandwidth-monitoring>
- Cache Allocation Technology (CAT) allows you to quiet the noisy neighbor and allocate proper resources to apps or VMs that have higher priority. Read more about CAT at the Intel Developer Zone: <https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology>
- Code and Data Prioritization (CDP) enables you to give priority and protection to apps or VMs with large code or data footprints so that they don't have to contend for resources. Read more about CDP at the Intel Developer Zone: <https://software.intel.com/en-us/articles/introduction-to-code-and-data-prioritization-with-usage-models>

Applying Intel RDT for the Best User Experience in the Cloud

Let's take a closer look at Intel RDT to see how it helps detect and mitigate noisy neighbors. To do this, we simulated a typical computer server on the cloud that is hosting multiple VMs. A 2-socket server powered by Intel Xeon Processor E5-2699 v4 was used for our experiment. We ran Xen 4.6* as the hypervisor to host these VMs. The platform hosted five VMs running the CentOS 7.1* operating system along with the Xen control domain, Domain 0 (DOM0). All the VMs were configured to share the same processor socket, but each had its own cores, as illustrated in Figure 2. Each VM hosted an independent application. We used a mixture of applications for this experiment, primarily Enterprise Java* and Java* server workloads. See the VM Description sidebar for more information on the different VMs.

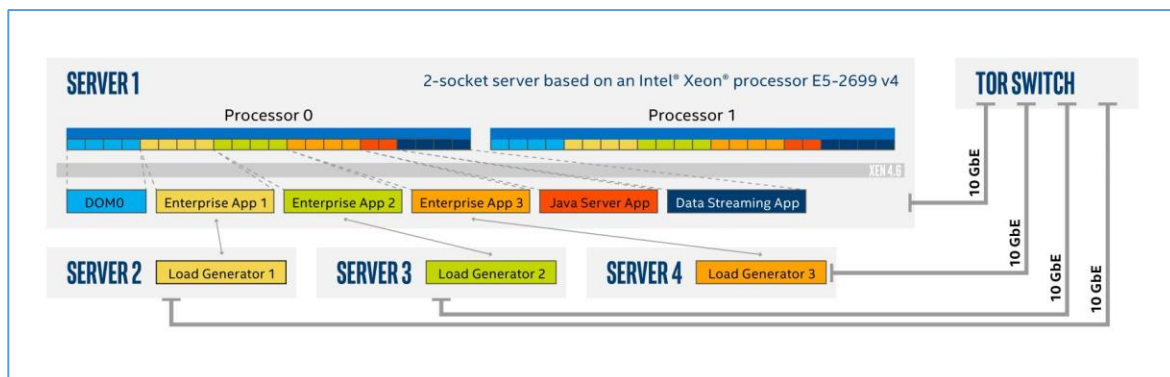


Figure 2. Intel® RDT test topology

Experiment

The objective of our experiment was to study the impact of different neighbors on an application's performance and to show how Intel RDT can help to detect and mitigate a noisy-neighbor situation. Application performance is usually measured by its throughput—the rate at which the application handles requests—and by response time—the time taken for the application to service the request and respond back to the users. In our experiment, we ran all the applications at a fixed throughput rate and

studied the changes in average response time under different scenarios. Most of our discussion here focuses on the performance of VM 1 (Enterprise App 1).

We ran the applications in different combinations to better understand how they interfere with each other. During each test, the applications were loaded with a fixed number of users and were run for a fixed duration while recording the average response time. In the last test case (Case 6), we utilized Intel RDT to mitigate the effect from a noisy-neighbor (VM5).

- **Case 1:** VM 1 (Enterprise App 1) running alone
- **Case 2:** VMs 1 and 2 (Enterprise Apps 1 and 2) running at the same time
- **Case 3:** VMs 1–3 (Enterprise Apps 1, 2, and 3) running at the same time
- **Case 4:** VMs 1–4 (Enterprise Apps 1, 2, and 3 and Java Server App) running at the same time
- **Case 5:** VMs 1–5 (Enterprise Apps 1, 2, and 3, Java Server App, and Data Streaming App) running at the same time
- **Case 6:** VMs 1–5 (Enterprise Apps 1, 2, and 3, Java Server App, and Data Streaming App) running at the same time; **use Intel® RDT to control cache usage**

Figure 3 shows the changes in performance of Enterprise App 1 when its neighbors start to run in parallel. Clearly, running the Data Streaming App (VM5 in Case 5) has the most significant impact on VM1's performance and increases the average response time by 190 percent from the baseline, which is much

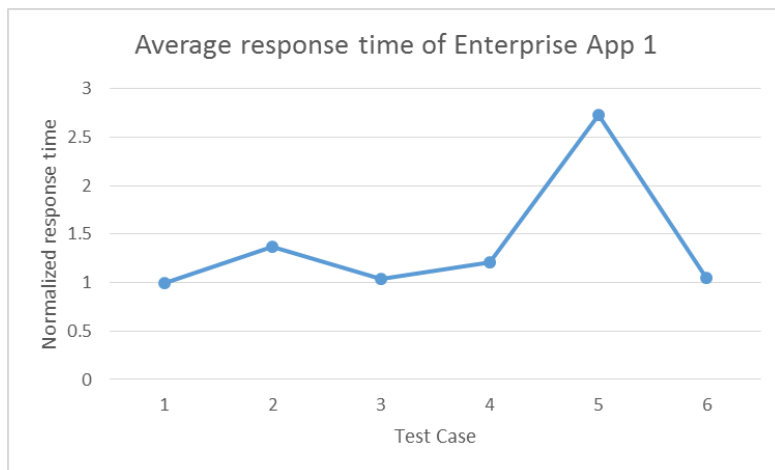


Figure 3. Average response time (seconds) of Enterprise App1 for the different case. Lower is better.

larger than the average run-to-run variation. After applying Intel RDT in Case 6 (as shown later), the response time of Enterprise App 1 came back down to acceptable levels.

VM Description

VM 0—DOM0 (8 vCPUs, 8 GB memory)

On Xen* platforms, DOM0 is the first domain started by the hypervisor at boot, and it typically runs the Linux* OS (in our environment, we used CentOS 7.1*). It has privileged access to the hardware and can control other guest domains. In the Xen para-virtualization model, DOM0 hosts back-end drivers, which act as a conduit between the virtual and physical hardware.

VMs 1, 2, and 3—Enterprise App (8 vCPUs, 60 GB memory)



Each VM hosted an independent instance of Ticket Monster, an open-source sample Java EE* application emulating an online booking web site. Ticket Monster uses H2*, a relational database-management system (RDBMS) based on Java, as an in-memory database. For our experiment, we designed a workflow using Apache* JMeter* scripts to drive the load. Each instance of the Ticket Monster application serves requests generated by an independent load generator. More details about this workload can be found in [Appendix A](#).

VM 4—Java Server App (4 vCPUs, 15 GB memory)

This VM ran SPECjbb2015*, a benchmark developed by SPEC to measure server-side Java performance. Refer to spec.org/jbb2015/ for more info. We ran SPECjbb2015 in composite (standalone) mode; thus, it did not require an external load generator. We used SPECjbb2015 as a Java server workload, and it was not intended as a benchmark.

VM 5—Data Streaming App (8 vCPUs, 30 GB memory)

We ran the STREAM* benchmark on this VM to simulate a data streaming application. STREAM is a benchmark designed to measure sustainable memory bandwidth and the corresponding computation rate for a simple vector kernel. Our objective was to use this benchmark to simulate an aggressive cache user—an application that occupies a large amount of cache but that doesn't reuse the cache effectively. Such a characteristic could be exhibited by some scientific, multimedia, and financial applications.

Examining Potential Causes for Interference

The focus of our experiment was to study interference on the processor, and hence we carefully constructed the experiment as follows:

- The five VMs were self-contained; so there was no direct software dependency between the applications.
- The load generators ran on different machines to reduce the chances of interference between them.
- The test workloads didn't use much persistent storage. This reduced chances of storage interference.

But there could be other sources of interference, such as:

- The enterprise applications were web applications that received their requests over the network. The three enterprise applications and their corresponding load generators shared the same network infrastructure. This can be a potential avenue for interference. Also, DOMO handled the network back end for all the three enterprise applications; hence another possibility of interference. To minimize the impact of network interference, we ran the workloads at moderate loads. As we can see from Case 2 and Case 3, the interference between the three enterprise applications was minimal compared to the data streaming app (see Figure 3).
- The VMs hosting the applications were assigned different cores on the processor in order to avoid interference from shared cores. However, the VMs still shared LLC and the memory controller on the processor. In this paper, we focus on these interferences.

Detecting a Noisy Neighbor Situation before the Customer Calls

In a production data center, it is much harder to look at interference in a controlled fashion. Especially in infrastructure-as-a-service (IaaS) or platform-as-a-service (PaaS) clouds, the CSPs might not have access to application performance data. In addition to Intel RDT, other counters provided by the platform Performance Monitoring Unit (PMU) can provide valuable information around monitoring application performance in these scenarios. For example, using these counters, system operators can compute the cycles per instruction (CPI) of an application. This captures the average number of CPU cycles that it takes to execute a microprocessor instruction, and it can be used as a proxy for application performance. Figure 4 shows the average CPU

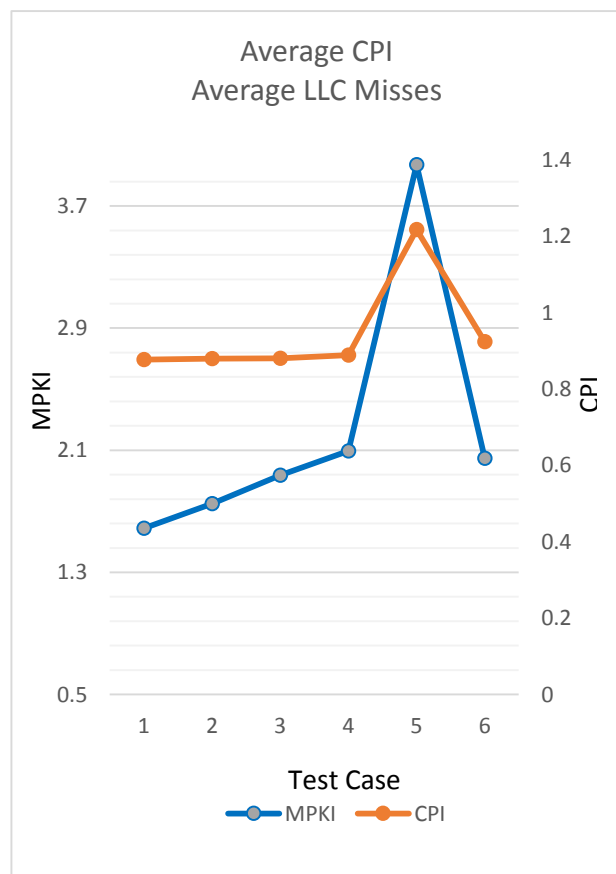


Figure 4. Average cycles per instruction (CPI) and LLC misses, misses per kilo instructions (MPKI), can help to detect the presence of a noisy neighbor. Lower is better.

clock cycles required to execute each CPU instruction for enterprise app 1 during the different test cases. Figure 4 also shows the LLC misses (misses per kilo instructions or MPKI), which can help to detect if applications might be suffering from LLC contention. We plotted the LLC misses of Enterprise App 1 for every 1,000 instructions. We can see that there is a large increase in both cache misses and CPI during Case 5, and that these issues are mostly recovered in Case 6. This correlates with the performance data shown in Figure 3. When the processors attempt to execute instructions, and the information is not found in the cache, then they must spend more time waiting for the information to come back from memory, which has much higher latency than processors' caches.

For more information on performance monitoring, check out the chapter on performance monitoring in the [Intel Software Development Manual](#).

Detecting a Noisy Neighbor by Monitoring Cache and Memory Usage

Intel RDT is integrated into multiple operating system and hypervisor environments. More information on accessing and using Intel RDT features on can be found in [Appendix B](#).

On Xen, Intel RDT features are integrated into the Xen xl tool stack and can be accessed from the command line in DOM0. These commands use Intel RDT to monitor and control individual VMs. For example, the following command displays the cache occupancy of all monitored VMs:

```
$ xl psr-cmt-show cache-occupancy
```

We used a shell script similar to the one shown in Figure 5 to collect cache occupancy and memory bandwidth of the VMs during the course of the experiment. Figure 6 shows average cache occupancy and memory bandwidth across all VMs for each test case.

The cache-occupancy plot shows that during Case 5, the data streaming app hogged a large portion of the cache. Thus it negatively impacted all the other VMs performance. When the required information is not in the cache, the

```
Sample Script:Collect total memory bandwidth of all VMs

# First step is to enable CMT monitoring on the VMs
for id in $(seq 0 5); do
    xl psr-cmt-attach $id
done

# Monitor total memory bandwidth
watch -n 1 xl psr-cmt-show total-mem-bandwidth

# Detach VMs after monitoring
for id in $(seq 0 5); do
    xl psr-cmt-detach $id
done
```

Figure 5. Sample script to collect Intel® RDT data from DOM0

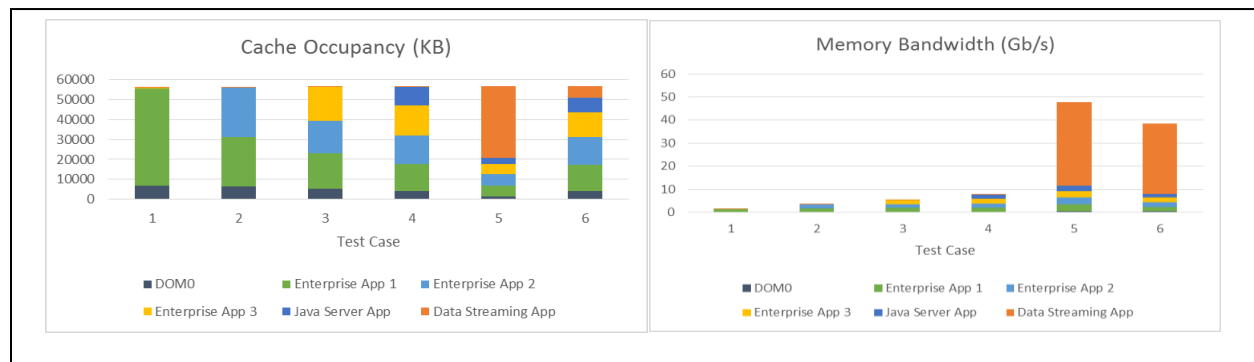


Figure 6. Cache occupancy and memory bandwidth

processor requires more time to get the information from memory, as memory speeds are much slower than cache. Also note that since DOM0 handles networking and storage back-end services for all the VMs running on the host, interference with DOM0 will potentially affect multiple VMs, even the ones that are not sharing the processor with the noisy neighbor.

From the memory-bandwidth plot, we can see that the Java applications did not use much memory bandwidth and seemed to have sufficient memory bandwidth even when the data streaming app was running. There was a slight increase in memory bandwidth used by these applications in Case 5. This was a result of the increased cache contention caused by Data Streaming App. Our analysis indicated that the increase in cache misses was the primary cause for the increase in user response time.

A Detailed Look at Cache Allocation Technology (CAT)

CAT is the Intel RDT mechanism for controlling access to LLC on the processor. To achieve this, Intel RDT can be used to partition the LLC according to Capacity Bitmasks (CBMs), which today correspond to the granularity typically corresponding to the number of cache ways. Each application can be associated with a portion of the cache using a bit mask, where each bit corresponds to one of the partitions. For example, the Intel Xeon processor E5-2699 v4 product family has 55 MB of 20-way associative LLC. CAT uses a 20-bit mask to control where a particular application (via Class of Service) can allocate. A bit mask of 0xffff allows full access to the cache, whereas a bit mask of 0x00003 allows access to only the last two cache partitions, effectively 5.5 MB. Such a mechanism provides great flexibility by allowing users to partition a cache in a shared, disjointed, or overlapping fashion. As stated earlier, Intel RDT features are accessible through a command-line interface (CLI) from DOM0 on Xen. The following command is an example of using CAT to associate the cache mask 0xfff (to allow access

```
$ xl psr-cat-cbm-set 1 0xfff
```

to last 12 cache ways) with a specific VM (with domain id 1, in this case):

Quieting a Noisy Neighbor

Our analysis indicates that the data streaming application occupied most of the LLC and negatively impacted the performance of the other applications. To mitigate this situation, we used Intel RDT to restrict the amount of cache accessible to the data streaming application. Table 1 shows the LLC partitions used in our experiment.

Table 1. LLC partitions used

VM(s)	Bitmask	Effective Cache
DOM0	0xffff	55 MB
Enterprise Java App 1, 2, and 3 and Java Server App	0xffffc	49.5 MB
Data Streaming App	0x3	5.5 MB

By applying such cache partitioning using Intel RDT, we found that both DOM0 and the Java VMs performed better with more cache resulting in significant performance improvements. The performance of the data steaming app largely remained unaffected, even when the amount of cache was restricted.

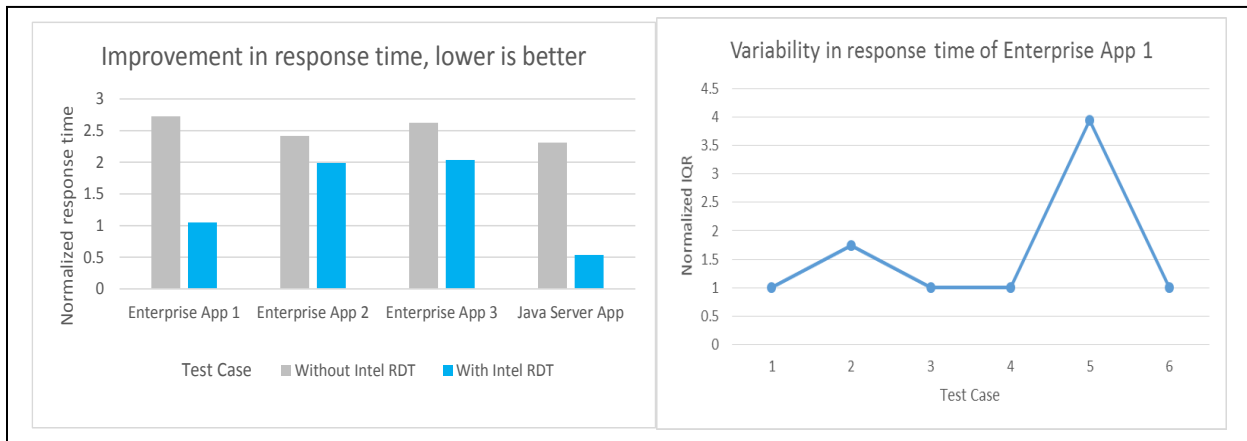


Figure 7. Intel® RDT CAT mitigated the noisy-neighbor effect and improved applications' performance and predictability

Figure 7 shows how different VMs gained back their performance after mitigating the noisy-neighbor situation using CAT. Please note that the impact of cache on performance is highly dependent on the application characteristics. In our experiment, we found that DOM0 performed better with more cache, even when it shared cache with the data streaming app. On the other hand, the Java VMs performed better when they had an exclusive section of the cache that was not polluted by the data streaming app.

Predictability of response time is important for applications, especially those that guarantee service-level agreements (SLAs). Figure 7 also shows variability in response time, as measured by interquartile range (IQR). We can see that there is a significant increase in variability of response time when the data streaming app starts to run. It gets back to acceptable levels when we mitigate the noisy-neighbor situation using Intel RDT CAT in Case 6.

Intel RDT Benefits a Wide Range of Workloads

Intel has conducted multiple studies on the benefits of Intel RDT as part of our effort to optimize cloud performance. We demonstrated that Intel RDT can mitigate the impact of noisy neighbors on a wide range of workloads, as shown in Figure 8.

In addition to the use cases discussed in this paper, Intel RDT can be used to prioritize an application over its neighbors by giving the priority app a greater share of resources. Various customers across the cloud datacenter, communications, and even in Internet-of-Things (IOT) domains have found similar value with RDT.

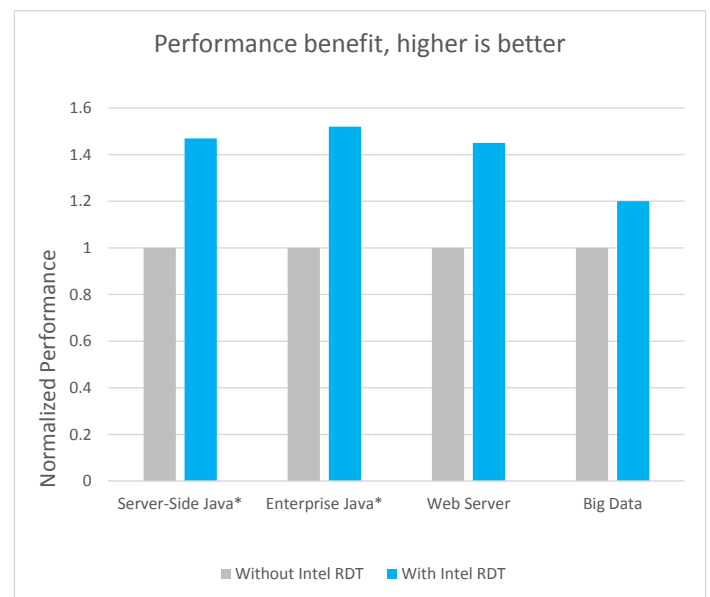


Figure 8. Relative performance gains on different categories of applications

Conclusion

Cloud computing offers many advantages, including flexibility and economies of scale. But the benefits also come with some new headaches, including noisy neighbors and unpredictable performance. Intel RDT provides CSPs and other customers with new tools to detect and mitigate noisy neighbors by providing mechanisms to monitor and control the shared resources. Our experiment shows that Intel RDT can improve both application response times (user experience) and throughput.

Questions and Feedback

If you have any questions or feedback, please feel free to contact the authors, Sunil Raghavan (sunil.raghavan@intel.com) and Khun Ban (khun.ban@intel.com).

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit intel.com/benchmarks.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Intel, the Intel logo, and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

© 2017 Intel Corporation.

Printed in USA

Appendix A: Building an Enterprise Java* Workload Using Ticket Monster

Ticket Monster is an open source application that demonstrates the capabilities of Red Hat JBoss Enterprise Application Platform. We use it as a proxy Enterprise Java application for our cloud performance study. This document describes how we designed and built a workload around TicketMonster.

A quick overview

Ticket Monster documentation¹ describes the application as follows:

“TicketMonster is an example application that focuses on Java EE6 - JPA 2, CDI, EJB 3.1 and JAX-RS along with HTML5 and jQuery Mobile. It is a moderately complex application that demonstrates how to build modern web applications optimized for mobile & desktop. TicketMonster is representative of an online ticketing broker - providing access to events (e.g. concerts, shows, etc.) with an online booking application”

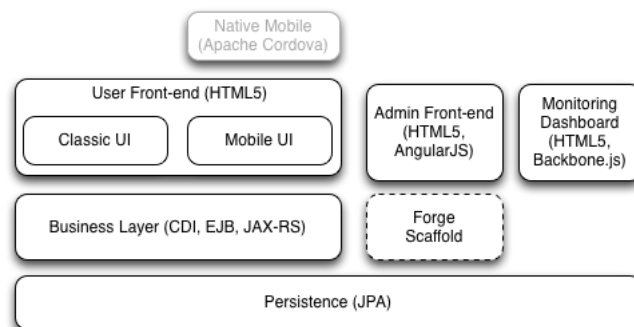


Figure 9 Ticket Monster architecture

It is a simple three tier web application offering rich mobile-ready front-end built with HTML5, AngularJS, and jQuery. The business layer uses JAX-RS APIs to provide a RESTful web service for the front-end and Java Persistence APIs (JPA) to connect to the database. The application is configured to use H2 Database Engine² running in-memory. A comprehensive documentation³ for Ticket Monster is available on the JBoss website that explains how each of these component was designed and developed.

Use Cases

Ticket Monster is basically a ticket booking web application. It builds around 'events' and 'venues' that host events. A 'show' is an event hosted on a venue. A 'performance' is a show at a specific time. Venues have 'sections', each section has an associated 'TicketPrice' at which seats are charged. 'Tickets' can be booked for a performance at a section on the venue.

There can be two types of users accessing the application – vendors who sell the tickets, consumers who buy tickets⁴. Vendors can add information on venues, events, shows etc. using the

'administration' interface. There is also a 'monitor' interface to help in real time monitoring of ticket sales. Consumers look for shows using the 'events' page or the 'venues' page. Both the webpages let the consumers browse through the events, find a specific show and book a ticket. They can also look at reserved tickets and choose to cancel.

Designing the workload

The workload is implemented using Apache jMeter⁵ framework. Our objective is to study the performance of Ticket Monster application server and not the front-end UI. The workload by-passes the front-end and communicates directly with the application server using REST APIs.

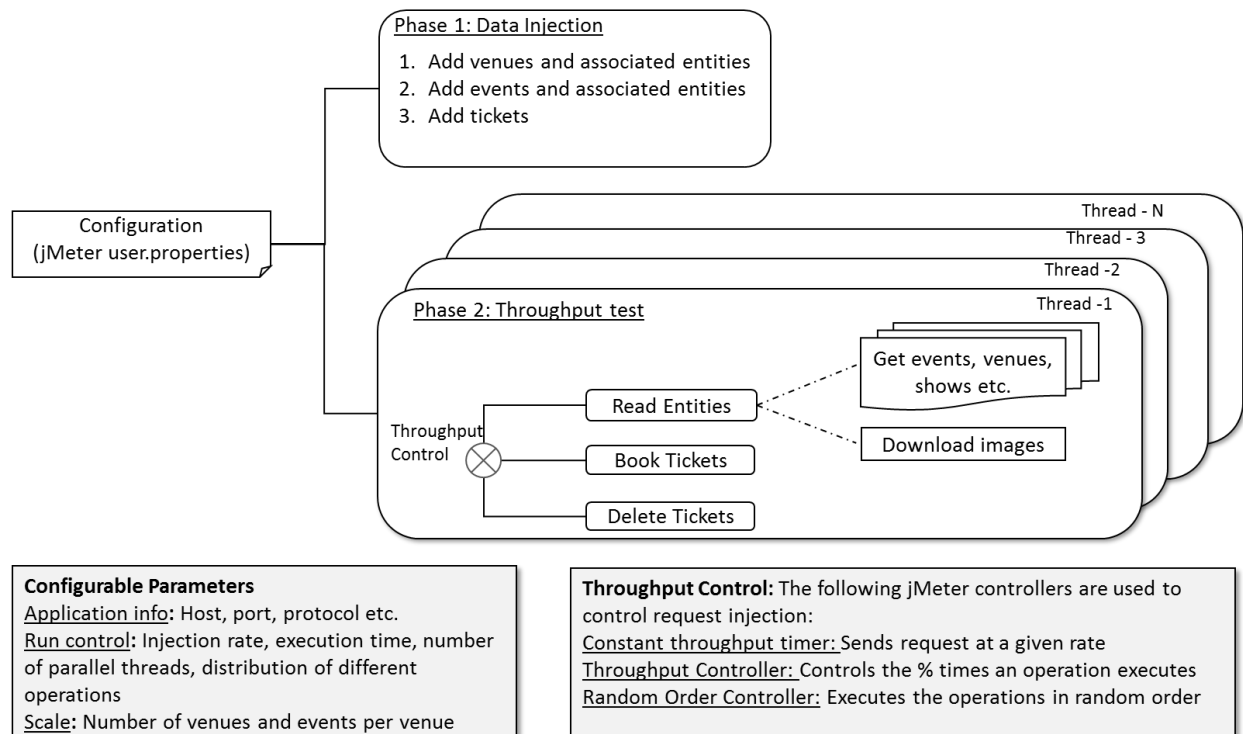


Figure 10 Workload design

The workload is highly customizable through configuration stored in jMeter properties file. It runs in two phases - data injection and throughput test. The phases execute sequentially and are implemented as different 'Thread Groups' in jMeter.

In the data injection phase the application is loaded with initial data – venues, events, shows, tickets etc. The number of events and venues to populate can be configured. The workload assumes all events are hosted in all venues and each venue has one section that can host 10000 seats. We typically use 100 venues and 100 events which results in 10,000 shows and 100 million potential seats that can be reserved.

Note: By default Ticket Monster populates default values for all the entities. We modified 'import.sql' in Ticket Monster and removed these default entries. For more information on this file can be found in Ticket Monster documentation.

The throughput test is designed to emulate the interactions between the front-end UI and application server. We emulate users sending requests for different operations simultaneously to the application server. The number of users and the rate of requests are configurable. The following assumptions are made when modelling user requests:

- Read: Most common type of request sent from the front-end UI is the 'read' request. The front-end is designed to make periodic Ajax⁶ calls to retrieve the list of all the venues and events. Also the webpage puts up a carousel display of event pictures for which it periodically retrieves images from the server. It has to be noted that these APIs will be called even if the user doesn't do anything interactive on the webpage.
- Add and Delete Tickets: Booking and deleting tickets are the key operations on TicketMonster and often the take the longest time to perform. Note that TicketMonster has no user authentication support and hence any user can view or delete any ticket.
- The throughput test basically comprises of the above three operations. The distribution of requests between these three operations can be configurable. We typically split the requests as 50% read operations, 33% book and 17% delete.
- Another consideration in our design is using persistent HTTP connection (keep-alive). The workload will generate a separate set of requests that happen with keep-alive turned off. The percentage of HTTP requests without keep-alive is configurable. In general we assume most of the requests are sent over persistent HTTP connection than otherwise.
- We assume that the 'vendor' operations (such as adding entities, monitoring sales etc.) will not be as frequent as the 'consumer' operations. The throughput test focuses exclusively on consumer operations.

Please check the attached jMeter test plan for a closer look at the implementation.

Setting up the workload

Here are the quick steps to setup the workload:

i. Application Server:

TicketMonster application server should be installed on the system under test. Follow the instructions on the TicketMonster website (link below) to download and install TicketMonster.

http://www.jboss.org/ticket-monster/whatisticketmonster/#_how_can_you_run_it

In our tests we used TicketMonster 2.7.0-Final running on JBoss 7.1.1 and Java OpenJDK 1.7.0.

ii. Load generator:

- Download and install Apache jMeter on an external machine. Instructions for Apache jMeter can be found in the following location.

<http://jmeter.apache.org/usermanual/get-started.html>

In our tests we used Apache jMeter 3.0-r1.

- Copy the attached [Configuration file](#) to [apache-jMeter directory]/bin. This will overwrite the default user.properties file. The listing for the user.properties can also be found in [Appendix C](#).
- Download the attached [jMeter test plan](#). This can be opened from jMeter application or invoked from command line as explained in the next section. The listing for the test plan can also be found in [Appendix D](#).

Running the workload

The workload can be run using jMeter command passing essential configuration parameters. A sample invocation would look like the following:

```
$ ./jmeter -n -t monsterdr.jmx -l results.csv -JHOST=192.168.1.1 -JPORT=8080 -
JUSERS=500 -JIR=1000 -JRAMPUP=60 -JSTEADYSTATE=240
```

The above command would run the test on TicketMonster running at <http://192.168.1.1:8080> and will emulate 500 users running concurrently, sending requests at the rate of 1000 requests per second. The test will run for a total time of 5 minutes including 60 seconds of ramp up. [Appendix D](#) has the complete jMeter test plan (monsterdr.jmx).

Performance Data

Typically application performance is measured by its throughput - the rate at which the application handles requests, and by response time - time taken for the application to service the request and response back to the users (a.k.a. latency). By making the injection rate and number of users configurable, the workload allows the flexibility to setup the test to suit different needs.

For collecting the results we use the jMeter default summarizer and 'Summary Report' listener. The run results are generated in a csv file capturing each request URL, response code and latency among other parameters. A sample csv results file would look like the following:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
timeStamp	elapsed	label	responseCode	threadName	dataType	success	failureMbytes	grpThread:	allThreads	URL	Latency	IdleTime	Connect	
31:43.6	4	Get 1 show	200	Throughput test 2-372	text	TRUE		1129	500	500	http://192.168.250.229:8080/ticket-mc	4	0	0
31:43.6	5	Get 1 ticketprice	200	Throughput test 2-365	text	TRUE		475	500	500	http://192.168.250.229:8080/ticket-mc	5	0	0
31:43.6	5	Add booking	200	Throughput test 2-442	text	TRUE		2835	500	500	http://192.168.250.229:8080/ticket-mc	5	0	0
31:43.6	3	Get cached image	200	Throughput test 2-186	text	TRUE		192955	500	500	http://192.168.250.229:8080/ticket-mc	1	0	0
31:43.7	5	Get cached image	200	Throughput test 2-369	text	TRUE		192955	500	500	http://192.168.250.229:8080/ticket-mc	2	0	0
31:43.7	5	Get cached image	200	Throughput test 2-357	text	TRUE		192955	500	500	http://192.168.250.229:8080/ticket-mc	2	0	1
31:43.7	4	Get cached image	200	Throughput test 2-183	text	TRUE		192955	500	500	http://192.168.250.229:8080/ticket-mc	1	0	0
31:43.7	3	Get 1 performance	200	Throughput test 2-397	text	TRUE		294	500	500	http://192.168.250.229:8080/ticket-mc	3	0	1
31:43.7	2	Get 1 ticketprice	200	Throughput test 2-180	text	TRUE		475	500	500	http://192.168.250.229:8080/ticket-mc	2	0	0
31:43.6	15	Get venues Keepali	200	Throughput test 2-274	text	TRUE		41711	500	500	http://192.168.250.229:8080/ticket-mc	15	0	0
31:43.7	4	Get cached image	200	Throughput test 2-407	text	TRUE		192955	500	500	http://192.168.250.229:8080/ticket-mc	1	0	0
31:43.7	2	Get 1 show	200	Throughput test 2-318	text	TRUE		1127	500	500	http://192.168.250.229:8080/ticket-mc	2	0	0
31:43.7	6	Get 1 show	200	Throughput test 2-76	text	TRUE		1127	500	500	http://192.168.250.229:8080/ticket-mc	6	0	0

Figure 11 Sample results in csv file (opened as a spreadsheet)

The summary information logged in jMeter.log file also provides throughput information:

```
2017/03/13 11:30:00 INFO - jmeter.reporters.Summariser: summary = 113570 in 00:02:37 = 721.1/s Avg:
8 Min: 0 Max: 563 Err: 1 (0.00%)

2017/03/13 11:30:30 INFO - jmeter.reporters.Summariser: summary + 30080 in 00:00:30 = 1002.7/s Avg:
14 Min: 1 Max: 404 Err: 0 (0.00%) Active: 500 Started: 500

1 Finished: 1
```

Figure 12 Sample summary information collected in jMeter.log

The following plot⁷ shows average response time of different operations in a sample run (generated from the csv output):

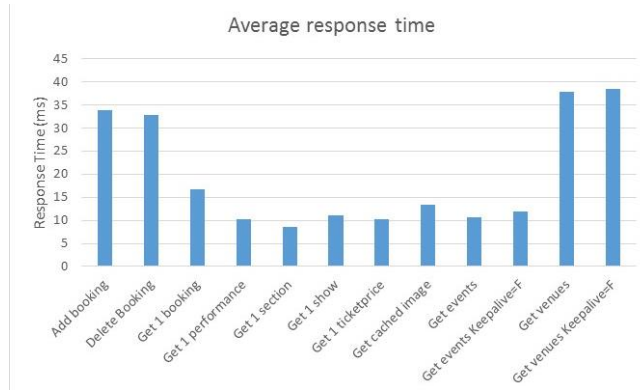


Figure 13 Average response time plotted using the results csv file

Next Steps

So far we have created a very simple workload using TicketMonster with minimal change to the default application configuration. We are exploring ways to expand this by separating the database layer and scaling the application to cloud scale. There are potential scope of improvement on the security aspects such as support for secure http and enhanced user authentication / authorization to mimic real world applications.

Attachments

- i. jMeter test plan



monsterdr.jmx

- ii. Configuration file



user.properties

Appendix B: Using Intel Resource Director Technology on Xen*

Intel Resource Director Technology (Intel RDT⁸) is an advanced resource monitoring and control feature that is designed to improve visibility into and control over usage of shared platform resources such as memory bandwidth and last-level cache⁹. It is featured on the Intel[®] Xeon[®] processor family as part of Intel's multi-year initiative to provide more insight and granular control over shared resources.

In this document we are going to look at how Intel RDT can be used in Xen environments. We use a simple example to explain the application of RDT¹⁰.

The Problem

In hosted platforms such as Xen, we have multiple VMs sharing the same resources including the processor. VMs sharing the same processor, assuming the processor has enough cores to host all the VCPUs, compete for shared processor resources such as last level cache (LLC) and memory bandwidth. It is important to be able to monitor and manage these resources to avoid starvation scenarios that could lead to significant performance penalties. Intel RDT supports such a mechanism to monitor and control access to last level cache and memory bandwidth.

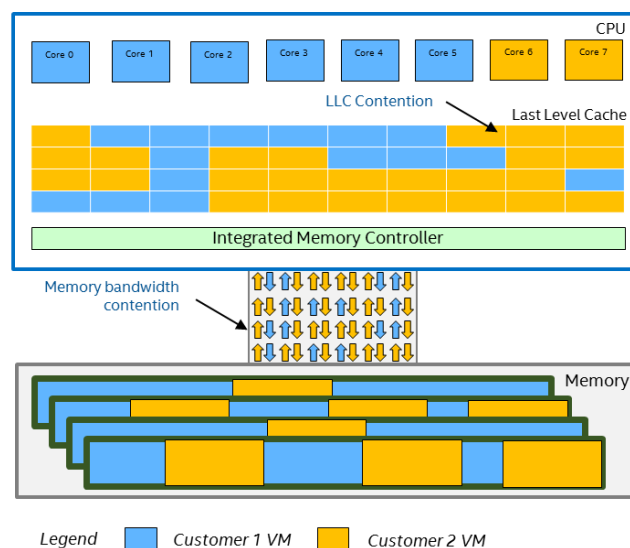


Figure 14. Illustration of LLC and memory bandwidth contention between two VMs on a hypothetical 8 core processor

Support on Xen platforms

Intel RDT features are supported on Xen platform incrementally¹¹ from Xen 4.5 and accessible to the xl toolstack of Xen. The following features are supported:

- Cache Monitoring Technology (CMT) :
 - Monitor last level cache usage of a VM

- Memory Bandwidth Monitoring (MBM) :
 - Monitor local and total memory bandwidth usage of a VM
- Cache Allocation Technology (CAT):
 - Control last level cache access of a VM
- Code and Data Prioritization (CDP):
 - Control last level cache access for code and data portions of a VM separately

```

root@localhost:~ (on ...)
File Edit View Search Terminal Help
[root@localhost ~]# xl help | grep psr
psr-hwinfo          Show hardware information for Platform Shared Resource
psr-cmt-attach      Attach Cache Monitoring Technology service to a domain
psr-cmt-detach      Detach Cache Monitoring Technology service from a domain
psr-cmt-show        Show Cache Monitoring Technology information
psr-cat-cbm-set     Set cache capacity bitmasks(CBM) for a domain
psr-cat-show        Show Cache Allocation Technology information
  
```

Figure 15. The xl commands for RDT are prefixed 'psr' for Platform Shared Resources

Enabling RDT support

To enable support of RDT we need to set the 'psr' Xen command line option accordingly. Using this option users can individually enable or disable RDT features – CMT, CAT and CDP. (Enabling CMT automatically enables memory bandwidth monitoring.) It also allows configuration of RDT parameters such as RMID and COS (explained later). Please check <https://xenbits.xenproject.org/docs/unstable/misc/xen-command-line.html> for a complete documentation.

psr (Intel)

= List of (cmt:<boolean> | rmid_max:<integer> | cat:<boolean> | cos_max:<integer> | cdp:<boolean>)

Default: psr=cmt:0,rmid_max:255,cat:0,cos_max:255,cdp:0

Figure 16. Configuration of 'psr' Xen command line option (from documentation)

Example Setup

In the rest of the document we are going to discuss RDT commands using a simple example. We use a two-socket Intel Xen v4 2699 server running Centos 7.2 with Xen 4.6 hypervisor.

The DOM0 is configured to have 8 VCPUs that are pinned and allocated 16 GB of memory. We run a bzip2 program to compress random ~1.3 GB of data generated using the following command.

```
openssl rand -out /dev/shm/test1 -base64 $((2**30))
```

Figure 17 Generates 1 GB of random data. Base64 encoding makes the file size slightly larger.

We see that bzip2 takes about 1min 46 secs to compress this data.

```
root@localhost:~/bz (on ...)
[root@localhost bz]# time bzip2 -c9 < /dev/shm/test1 > /dev/null

real    1m46.386s
user    1m45.967s
sys     0m0.414s
```

Figure 18. Compress the random data using bzip2. The compressed content is redirected to /dev/null.

Now we add a neighbor VM to DOM0, with 8 VCPUs and 16 GB of memory running Centos 7.2. This VM is configured to have hard CPU affinity to the same processor on which DOM0 runs. We run a special program called 'STREAM'¹² in this VM to simulate a program that is an aggressive cache user - an application that occupies a large amount of cache but doesn't reuse the cache effectively. More information about this benchmark and the source code can be found in <http://www.cs.virginia.edu/stream/ref.html>

```
while true; do ./stream; done 2>&1 >/dev/null &
```

Figure 19. STREAM is run in an infinite loop to maintain constant pressure on cache through the experiment. We run 4 such instances of this benchmark.

Monitoring shared resources

As mentioned before RDT supports monitoring of last level cache and memory bandwidth usage. It allows operating systems and hypervisors to define a Resource Monitoring ID (RMID) and associate it with entities they monitor – such as processors, VMs etc. There is a finite number of RMTDs supported on a processor and hence limits the number of entities that can be monitored.

```
root@localhost:~/ (on ...)
File Edit View Search Terminal Help
[root@localhost ~]# xl psr-hwinfo
Cache Monitoring Technology (CMT):
Enabled      : 1
Total RMID   : 175
Supported monitor types:
cache-occupancy
total-mem-bandwidth
local-mem-bandwidth
Cache Allocation Technology (CAT):
Socket ID    : 0
L3 Cache     : 56320KB
Maximum COS  : 15
CBM length   : 20
Default CBM  : 0xffff
Socket ID    : 1
L3 Cache     : 56320KB
Maximum COS  : 15
CBM length   : 20
Default CBM  : 0xffff
```

Figure 20. The above command gives the RDT support information on the platform. In our case the processor supports 175 RMTDs.

Xen abstracts RMTDs and provide an interface to the users to monitor VMs. However user needs to explicitly attach and detach VMs to CMT before monitoring which under the hood will allocate / deallocate RMTDs.

```

root@localhost:~/bz (on ...)
[root@localhost bz]# xl psr-cmt-attach 0
[root@localhost bz]# xl psr-cmt-attach 1
[root@localhost bz]# xl psr-cmt-show cache-occupancy
Total RMID: 175
Name                ID      Socket 0      Socket 1
Total L3 Cache Size          56320 KB      56320 KB
Domain-0                0          1584 KB         0 KB
str1                    1          54560 KB         0 KB
[root@localhost bz]# xl psr-cmt-show total-mem-bandwidth
Total RMID: 175
Name                ID      Socket 0      Socket 1
Domain-0                0          68568 KB/s         0 KB/s
str1                    1          27888621 KB/s         0 KB/s

```

Figure 21. We pin both our VMs on Socket#0. Hence they cache occupancy on Socket#1 is OKB.

As we can see, STREAM occupies almost all of the cache and DOM0 is almost starved of last level cache. Now if we run the bzip2 command again, we see it takes 2 min 21 secs to finish due to this.

```

root@localhost:~/bz (on ...)
[root@localhost bz]# time bzip2 -c9 < /dev/shm/test1 > /dev/null
real    2m21.372s
user    2m20.838s
sys     0m0.532s

```

Figure 22. bzip2 test with STREAM running on a neighbor VM.

Note that the xen command only gives instantaneous values of cache occupancy and memory bandwidth. We can write a simple script around these commands to collect this over time.

```

get_cache_occ()
{
    outfile=$1
    while true; do
        #Get cache occupancy - Assmng only two domains
        xl psr-cmt-show cache-occupancy | awk '{print $1,$3}' | tail -n2 >> $outfile
        sleep 1
    done
}
#Attach Domains for CMT monitoring - Assuming domains 0 and 1
xl psr-cmt-attach 0
xl psr-cmt-attach 1

get_cache_occ $@ &
pid=$!
time bzip2 -c9 < /dev/shm/test1 > /dev/null
kill $pid

#Detach domains from CMT monitoring - - Assuming domains 0 and 1
xl psr-cmt-detach 0
xl psr-cmt-detach 1

```

Figure 23. This script collects LLC occupancy every 1 second when bzip2 is run. Note that it attaches before and detaches after the data collection.

We plot the data collected and can see that all along the run DOM0 is suppressed. We can use such a technique in real world to detect a 'noisy' neighbor such as STREAM that hogs the last level cache and starves other VMs.

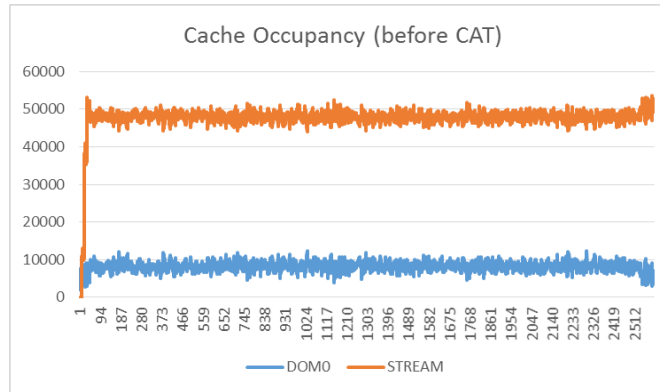


Figure 24. Plot showing cache occupancy of the two VMs. Usage of CAT is explained in the section below.

Managing shared resources using RDT

Intel RDT allows controlling access to last level cache through Cache Allocation Technology (CAT). It is modelled around the concept of 'Class of Service' (COS) where each class offers a certain portion of cache for its members. Operating systems and hypervisors can define multiple classes of services and associate VMs and applications with a COS. There can be more than one application or VM associated to same class of service.

Xen abstracts the concept of COS and provides a convenient interface to the user to directly allocate a cache portion to the VM. Under the hood it takes care of defining the COS and associating with the VM. Also note that the number of classes of service that can be defined is finite. The hardware info command, shown in Figure 20, can be used to get the number of COS supported.

Allocating Cache

The other concept used in CAT is the control bit mask (CBM). RDT splits the last level cache into equal sized logical partitions. Allocation of cache is achieved through a control bit mask where each bit indicates access to a cache partition. Defining a COS is basically assigning a control bit mask in the COS register. This model is very flexible and allows dividing cache in a shared, exclusive or overlapping fashion¹³.

Let us see how this works in our experiment. Looking at the hardware info, shown in Figure 20, shows that the total cache size is 55MB on this platform and control bit mask is 20 bit long. This means each bit in the CBM corresponds to 2.75 MB of cache. The default CBM is 0xfffff, which means by default the VMs have access to the whole cache.

We can get the current cache allocation as shown below.

```
root@localhost:~/bz (on ...)
[root@localhost bz]# xl psr-cat-show
Socket ID      : 0
L3 Cache      : 56320KB
Default CBM   : 0xffffffff
  ID          NAME          CBM
  0           Domain-0     0xffffffff
  1           str1         0xffffffff
Socket ID      : 1
L3 Cache      : 56320KB
Default CBM   : 0xffffffff
  ID          NAME          CBM
  0           Domain-0     0xffffffff
  1           str1         0xffffffff
```

Figure 25. xl psr-cat-show command helps to look at the current cache allocation.

Now let us try to restrict the cache accessible to Domain#1 where STREAM is running and see if it helps mitigating the impact on bzip2 that will be run on Domain#0. We restrict Domain#1 to access only 5.5MB of the cache.

```
root@localhost:~/bz (on ...)
[root@localhost bz]# xl psr-cat-cbm-set 1 0x3
```

Figure 26. CBM of 0x3 means Domain#1 will have access to only the last two cache ways and hence 5.5 MB of cache

Now let us run bzip2 test on DOM0.

```
root@localhost:~/bz (on ...)
[root@localhost bz]# time bzip2 -c9 < /dev/shm/test1 > /dev/null

real    1m58.695s
user    1m58.161s
sys     0m0.532s
```

Figure 27 Results of bzip2 test after CAT

We see that bzip2 has indeed benefited with CAT and the time to completion has significantly improved. We can observe the effect of CAT using the cache monitoring script shown in Figure 23.

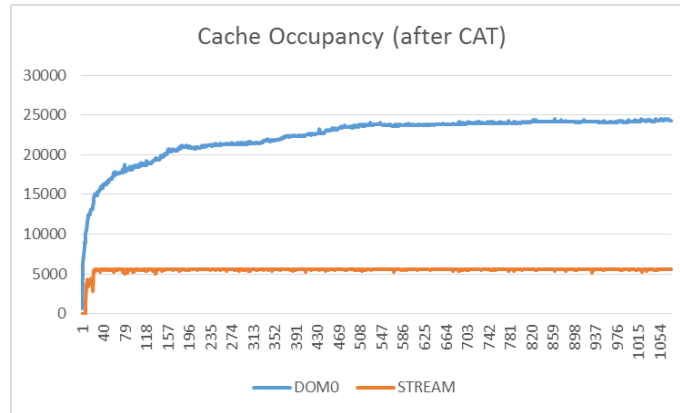


Figure 28. Cache occupancy after CAT

Note that we didn't change the bitmask of DOM0. This means DOM0 still has access to all the 20 cache ways out of which 2 are shared with Domain#1. We find that the bzip2 command we ran uses about 23 MB of last level cache.

The right cache allocation scheme is very much dependent on number of VMs and the applications running in them. We have found that some applications perform better with such overlapped cache allocation while others prefer no overlap with a noisy VM.

Other features of RDT

The focus in this example was around the last level cache. Memory bandwidth monitoring is another related and extremely useful feature to explore that can be used to detect noisy VMs. We also encourage exploring Code and Data Protection (CDP) which is similar to CAT but gives flexibility to treat code and data portions separately. This will be useful especially with workloads with very large code footprint. Memory bandwidth allocation¹⁴ (MBA) is a feature to look forward to which will be supported in future Intel Xeon platforms.

Appendix C: jMeter* Configuration File

```
# Sample user.properties file

#

## Licensed to the Apache Software Foundation (ASF) under one or more
## contributor license agreements. See the NOTICE file distributed with
## this work for additional information regarding copyright ownership.
## The ASF licenses this file to You under the Apache License, Version 2.0
## (the "License"); you may not use this file except in compliance with
## the License. You may obtain a copy of the License at
##
##      http://www.apache.org/licenses/LICENSE-2.0
##
## Unless required by applicable law or agreed to in writing, software
## distributed under the License is distributed on an "AS IS" BASIS,
## WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
## See the License for the specific language governing permissions and
## limitations under the License.

#-----
# Classpath configuration
#-----

#
# List of paths (separated by ;) to search for additional JMeter plugin classes,
# for example new GUI elements and samplers.
# A path item can either be a jar file or a directory.
# Any jar file in such a directory will be automatically included,
# jar files in sub directories are ignored.
# The given value is in addition to any jars found in the lib/ext directory.
# Do not use this for utility or plugin dependency jars.
#search_paths=/app1/lib;/app2/lib

# List of paths that JMeter will search for utility and plugin dependency classes.
# Use your platform path separator (java.io.File.separatorChar in Java) to separate multiple paths.
# A path item can either be a jar file or a directory.
# Any jar file in such a directory will be automatically included,
# jar files in sub directories are ignored.
# The given value is in addition to any jars found in the lib directory.
# All entries will be added to the class path of the system class loader
# and also to the path of the JMeter internal loader.
# Paths with spaces may cause problems for the JVM

#Example for windows (; separator)
#user.classpath=./classes;./lib;./app1/jar1.jar;./app2/jar2.jar
```

```
#Example for linux (:separator)

#user.classpath=./classes:./lib:./app1/jar1.jar:./app2/jar2.jar

# List of paths (separated by ;) that JMeter will search for utility
# and plugin dependency classes.
# A path item can either be a jar file or a directory.
# Any jar file in such a directory will be automatically included,
# jar files in sub directories are ignored.
# The given value is in addition to any jars found in the lib directory
# or given by the user.classpath property.
# All entries will be added to the path of the JMeter internal loader only.
# For plugin dependencies using plugin_dependency_paths should be preferred over
# user.classpath.
#plugin_dependency_paths=./dependencies/lib:./app1/jar1.jar:./app2/jar2.jar

#-----
# Logging configuration
#-----
#log_level.jorphan.reflect=DEBUG

# Warning: enabling the next debug line causes javax.net.ssl.SSLException: Received fatal alert: unexpected_message
# for certain sites when used with the default HTTP Sampler
#log_level.jmeter.util.HttpSSLProtocolSocketFactory=DEBUG
#log_level.jmeter.util.JsseSSLManager=DEBUG

# Enable Proxy request debug
#log_level.jmeter.protocol.http.proxy.HttpRequestHdr=DEBUG

#-----
# Reporting configuration
#-----

# If you want to debug reporting, uncomment this line
#log_level.jmeter.report=DEBUG

# Configure this property to change the report title
#jmeter.reportgenerator.report_title=Apache JMeter Dashboard

# Change this parameter if you want to change the granularity of over time graphs.
#jmeter.reportgenerator.overall_granularity=60000

# Change this parameter if you want to change the granularity of Response time distribution
# Set to 500 ms by default
#jmeter.reportgenerator.graph.responseTimeDistribution.property.set_granularity=500

# Change this parameter if you want to keep only some samples.
```

```
# Regular Expression which Indicates which samples to keep for graphs and statistics generation.

# Empty value means no filtering

#jmeter.reportgenerator.sample_filter=

# Change this parameter if you want to override the APDEX satisfaction threshold.

#jmeter.reportgenerator.apdex_satisfied_threshold=500

# Change this parameter if you want to override the APDEX tolerance threshold.

#jmeter.reportgenerator.apdex_tolerated_threshold=1500

# Indicates which graph series are filtered (regular expression)

# In the below example we filter on Search and Order samples

# Note that the end of the pattern should always include (-success|-failure)?

# TransactionsPerSecondGraphConsumer suffixes transactions with "-success" or "-failure" depending

# on the result

#jmeter.reportgenerator.exporter.html.series_filter=(Search|Order)(-success|-failure)?

# Indicates whether only controller samples are displayed on graphs that support it.

#jmeter.reportgenerator.exporter.html.show_controllers_only=false

# Custom parameters

HOST=192.168.1.1

PORT=8080

PROTOCOL=http

PERC_NEWCONN=17

PERC_READ=33

PERC_BOOK=33

PERC_DELETE=17

VENUES=100

EVENTS=100

USERS=500

RAMPUP=30

STEADYSTATE=80

IR=10000

DATAGEN=1
```

Appendix D: jMeter* Test Plan

```
<?xml version="1.0" encoding="UTF-8"?>
<jmeterTestPlan version="1.2" properties="2.9" jmeter="3.0 r1743807">
  <hashTree>
    <TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="Test Plan" enabled="true">
      <stringProp name="TestPlan.comments"></stringProp>
      <boolProp name="TestPlan.functional_mode">false</boolProp>
      <boolProp name="TestPlan.serialize_threadgroups">true</boolProp>
      <elementProp name="TestPlan.user_defined_variables" elementType="Arguments" guiclass="ArgumentsPanel" testclass="Arguments" testname="User Defined Variables"
        enabled="true">
        <collectionProp name="Arguments.arguments"/>
      </elementProp>
      <stringProp name="TestPlan.user_define_classpath"></stringProp>
    </TestPlan>
  </hashTree>
  <Arguments guiclass="ArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">
    <collectionProp name="Arguments.arguments">
      <elementProp name="HOST" elementType="Argument">
        <stringProp name="Argument.name">HOST</stringProp>
        <stringProp name="Argument.value">${__P(HOST)}</stringProp>
        <stringProp name="Argument.metadata">=</stringProp>
      </elementProp>
      <elementProp name="PORT" elementType="Argument">
        <stringProp name="Argument.name">PORT</stringProp>
        <stringProp name="Argument.value">${__P(PORT)}</stringProp>
        <stringProp name="Argument.metadata">=</stringProp>
      </elementProp>
      <elementProp name="PROTOCOL" elementType="Argument">
        <stringProp name="Argument.name">PROTOCOL</stringProp>
        <stringProp name="Argument.value">${__P(PROTOCOL)}</stringProp>
        <stringProp name="Argument.metadata">=</stringProp>
      </elementProp>
      <elementProp name="PERC_NEWCONN" elementType="Argument">
        <stringProp name="Argument.name">PERC_NEWCONN</stringProp>
        <stringProp name="Argument.value">${__P(PERC_NEWCONN)}</stringProp>
        <stringProp name="Argument.metadata">=</stringProp>
      </elementProp>
      <elementProp name="PERC_READ" elementType="Argument">
        <stringProp name="Argument.name">PERC_READ</stringProp>
        <stringProp name="Argument.value">${__P(PERC_READ)}</stringProp>
        <stringProp name="Argument.metadata">=</stringProp>
      </elementProp>
      <elementProp name="PERC_BOOK" elementType="Argument">

```

```
<stringProp name="Argument.name">PERC_BOOK</stringProp>
<stringProp name="Argument.value">${__P(PERC_BOOK)}</stringProp>
<stringProp name="Argument.metadata">=</stringProp>
</elementProp>
<elementProp name="PERC_DELETE" elementType="Argument">
  <stringProp name="Argument.name">PERC_DELETE</stringProp>
  <stringProp name="Argument.value">${__P(PERC_DELETE)}</stringProp>
  <stringProp name="Argument.metadata">=</stringProp>
</elementProp>
<elementProp name="VENUES" elementType="Argument">
  <stringProp name="Argument.name">VENUES</stringProp>
  <stringProp name="Argument.value">${__P(VENUES)}</stringProp>
  <stringProp name="Argument.metadata">=</stringProp>
</elementProp>
<elementProp name="EVENTS" elementType="Argument">
  <stringProp name="Argument.name">EVENTS</stringProp>
  <stringProp name="Argument.value">${__P(EVENTS)}</stringProp>
  <stringProp name="Argument.metadata">=</stringProp>
</elementProp>
<elementProp name="USERS" elementType="Argument">
  <stringProp name="Argument.name">USERS</stringProp>
  <stringProp name="Argument.value">${__P(USERS)}</stringProp>
  <stringProp name="Argument.metadata">=</stringProp>
</elementProp>
<elementProp name="RAMPUP" elementType="Argument">
  <stringProp name="Argument.name">RAMPUP</stringProp>
  <stringProp name="Argument.value">${__P(RAMPUP)}</stringProp>
  <stringProp name="Argument.metadata">=</stringProp>
</elementProp>
<elementProp name="STEADYSTATE" elementType="Argument">
  <stringProp name="Argument.name">STEADYSTATE</stringProp>
  <stringProp name="Argument.value">${__P(STEADYSTATE)}</stringProp>
  <stringProp name="Argument.metadata">=</stringProp>
</elementProp>
<elementProp name="IR" elementType="Argument">
  <stringProp name="Argument.name">IR</stringProp>
  <stringProp name="Argument.value">${__P(IR)}</stringProp>
  <stringProp name="Argument.metadata">=</stringProp>
</elementProp>
<elementProp name="DATAGEN" elementType="Argument">
  <stringProp name="Argument.name">DATAGEN</stringProp>
  <stringProp name="Argument.value">${__P(DATAGEN)}</stringProp>
  <stringProp name="Argument.metadata">=</stringProp>
</elementProp>
</collectionProp>
```

```

</Arguments>
<hashTree/>
<HeaderManager guiclass="HeaderPanel" testclass="HeaderManager" testname="HTTP Header Manager" enabled="true">
  <collectionProp name="HeaderManager.headers">
    <elementProp name="Content-Type" elementType="Header">
      <stringProp name="Header.name">Content-Type</stringProp>
      <stringProp name="Header.value">application/json;charset=utf-8</stringProp>
    </elementProp>
    <elementProp name="Accept-Language" elementType="Header">
      <stringProp name="Header.name">Accept-Language</stringProp>
      <stringProp name="Header.value">en-US,en;q=0.5</stringProp>
    </elementProp>
    <elementProp name="Accept" elementType="Header">
      <stringProp name="Header.name">Accept</stringProp>
      <stringProp name="Header.value">application/json, text/plain, */*</stringProp>
    </elementProp>
    <elementProp name="User-Agent" elementType="Header">
      <stringProp name="Header.name">User-Agent</stringProp>
      <stringProp name="Header.value">Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0</stringProp>
    </elementProp>
    <elementProp name="Accept-Encoding" elementType="Header">
      <stringProp name="Header.name">Accept-Encoding</stringProp>
      <stringProp name="Header.value">gzip, deflate</stringProp>
    </elementProp>
  </collectionProp>
</HeaderManager>
<hashTree/>
<ConfigTestElement guiclass="HttpDefaultsGui" testclass="ConfigTestElement" testname="HTTP Request Defaults" enabled="true">
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" testname="User Defined Variables"
  enabled="true">
    <collectionProp name="Arguments.arguments"/>
  </elementProp>
  <stringProp name="HTTPSampler.domain">${HOST}</stringProp>
  <stringProp name="HTTPSampler.port">${PORT}</stringProp>
  <stringProp name="HTTPSampler.connect_timeout"></stringProp>
  <stringProp name="HTTPSampler.response_timeout"></stringProp>
  <stringProp name="HTTPSampler.protocol">${PROTOCOL}</stringProp>
  <stringProp name="HTTPSampler.contentEncoding"></stringProp>
  <stringProp name="HTTPSampler.path"></stringProp>
  <stringProp name="HTTPSampler.implementation">HttpClient4</stringProp>
  <stringProp name="HTTPSampler.concurrentPool">6</stringProp>
</ConfigTestElement>
<hashTree/>
<ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup" testname="Data Injection" enabled="true">
  <stringProp name="ThreadGroup.on_sample_error">stoptest</stringProp>

```

```

<elementProp name="ThreadGroup.main_controller" elementType="LoopController" guiclass="LoopControlPanel" testclass="LoopController" testname="Loop Controller"
enabled="true">

  <boolProp name="LoopController.continue_forever">false</boolProp>

  <stringProp name="LoopController.loops">1</stringProp>

</elementProp>

<stringProp name="ThreadGroup.num_threads">1</stringProp>
<stringProp name="ThreadGroup.ramp_time">1</stringProp>
<longProp name="ThreadGroup.start_time">1479414441000</longProp>
<longProp name="ThreadGroup.end_time">1479414441000</longProp>
<boolProp name="ThreadGroup.scheduler">false</boolProp>
<stringProp name="ThreadGroup.duration"></stringProp>
<stringProp name="ThreadGroup.delay"></stringProp>

</ThreadGroup>

<hashTree>

<IfController guiclass="IfControllerPanel" testclass="IfController" testname="If Controller" enabled="true">

  <stringProp name="IfController.condition">${DATAGEN}==&quot;1&quot;</stringProp>

  <boolProp name="IfController.evaluateAll">false</boolProp>

</IfController>

<hashTree>

<Arguments guiclass="ArgumentsPanel" testclass="Arguments" testname="User Defined Variables" enabled="true">

  <collectionProp name="Arguments.arguments">

    <elementProp name="ENTITY1" elementType="Argument">

      <stringProp name="Argument.name">ENTITY1</stringProp>

      <stringProp name="Argument.value">bookings</stringProp>

      <stringProp name="Argument.metadata">=</stringProp>

    </elementProp>

    <elementProp name="ENTITY2" elementType="Argument">

      <stringProp name="Argument.name">ENTITY2</stringProp>

      <stringProp name="Argument.value">ticketprices</stringProp>

      <stringProp name="Argument.metadata">=</stringProp>

    </elementProp>

    <elementProp name="ENTITY3" elementType="Argument">

      <stringProp name="Argument.name">ENTITY3</stringProp>

      <stringProp name="Argument.value">performances</stringProp>

      <stringProp name="Argument.metadata">=</stringProp>

    </elementProp>

    <elementProp name="ENTITY4" elementType="Argument">

      <stringProp name="Argument.name">ENTITY4</stringProp>

      <stringProp name="Argument.value">forge/shows</stringProp>

      <stringProp name="Argument.metadata">=</stringProp>

    </elementProp>

    <elementProp name="ENTITY5" elementType="Argument">

      <stringProp name="Argument.name">ENTITY5</stringProp>

      <stringProp name="Argument.value">forge/events</stringProp>

      <stringProp name="Argument.metadata">=</stringProp>

    </elementProp>

  </collectionProp>

</Arguments>

</hashTree>

```

```

<elementProp name="ENTITY6" elementType="Argument">
  <stringProp name="Argument.name">ENTITY6</stringProp>
  <stringProp name="Argument.value">sections</stringProp>
  <stringProp name="Argument.metadata">=</stringProp>
</elementProp>
<elementProp name="ENTITY7" elementType="Argument">
  <stringProp name="Argument.name">ENTITY7</stringProp>
  <stringProp name="Argument.value">forge/venues</stringProp>
  <stringProp name="Argument.metadata">=</stringProp>
</elementProp>
<elementProp name="ENTITY8" elementType="Argument">
  <stringProp name="Argument.name">ENTITY8</stringProp>
  <stringProp name="Argument.value">mediaitems</stringProp>
  <stringProp name="Argument.metadata">=</stringProp>
</elementProp>
<elementProp name="DELR0WS" elementType="Argument">
  <stringProp name="Argument.name">DELR0WS</stringProp>
  <stringProp name="Argument.value">1</stringProp>
  <stringProp name="Argument.metadata">=</stringProp>
</elementProp>
</collectionProp>
</Arguments>
<hashTree/>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Add media item" enabled="true">
  <boolProp name="HTTPSampler.postBodyRaw">true</boolProp>
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments">
    <collectionProp name="Arguments.arguments">
      <elementProp name="" elementType="HTTPArgument">
        <boolProp name="HTTPArgument.always_encode">>false</boolProp>
        <stringProp name="Argument.value">{"&quot;mediaType&quot;:&quot;IMAGE&quot;,&quot;url&quot;:&quot;http://127.0.0.1:8080/ticket-
monster/resources/img/splash-ticketmonster.png&quot;}</stringProp>
        <stringProp name="Argument.metadata">=</stringProp>
      </elementProp>
    </collectionProp>
  </elementProp>
  <stringProp name="HTTPSampler.domain"></stringProp>
  <stringProp name="HTTPSampler.port"></stringProp>
  <stringProp name="HTTPSampler.connect_timeout"></stringProp>
  <stringProp name="HTTPSampler.response_timeout"></stringProp>
  <stringProp name="HTTPSampler.protocol">http</stringProp>
  <stringProp name="HTTPSampler.contentEncoding"></stringProp>
  <stringProp name="HTTPSampler.path">/ticket-monster/rest/mediaitems</stringProp>
  <stringProp name="HTTPSampler.method">POST</stringProp>
  <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
  <boolProp name="HTTPSampler.auto_redirects">>false</boolProp>
  <boolProp name="HTTPSampler.use_keepalive">true</boolProp>

```

```
<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
<boolProp name="HTTPSampler.monitor">false</boolProp>
<stringProp name="HTTPSampler.embedded_ur1_re"></stringProp>
</HTTPSamplerProxy>
<hashTree/>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Add Event Category" enabled="true">
  <boolProp name="HTTPSampler.postBodyRaw">true</boolProp>
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments">
    <collectionProp name="Arguments.arguments">
      <elementProp name="" elementType="HTTPArgument">
        <boolProp name="HTTPArgument.always_encode">false</boolProp>
        <stringProp name="Argument.value">{&quot;description&quot;: &quot;event event event&quot;}</stringProp>
        <stringProp name="Argument.metadata">=</stringProp>
      </elementProp>
    </collectionProp>
  </elementProp>
  <stringProp name="HTTPSampler.domain"></stringProp>
  <stringProp name="HTTPSampler.port"></stringProp>
  <stringProp name="HTTPSampler.connect_timeout"></stringProp>
  <stringProp name="HTTPSampler.response_timeout"></stringProp>
  <stringProp name="HTTPSampler.protocol"></stringProp>
  <stringProp name="HTTPSampler.contentEncoding"></stringProp>
  <stringProp name="HTTPSampler.path">/ticket-monster/rest/eventcategories</stringProp>
  <stringProp name="HTTPSampler.method">POST</stringProp>
  <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
  <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
  <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
  <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
  <boolProp name="HTTPSampler.monitor">false</boolProp>
  <stringProp name="HTTPSampler.embedded_ur1_re"></stringProp>
</HTTPSamplerProxy>
<hashTree/>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Add Ticket Category" enabled="true">
  <boolProp name="HTTPSampler.postBodyRaw">true</boolProp>
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments">
    <collectionProp name="Arguments.arguments">
      <elementProp name="" elementType="HTTPArgument">
        <boolProp name="HTTPArgument.always_encode">false</boolProp>
        <stringProp name="Argument.value">{&quot;description&quot;: &quot;ticket ticket ticket&quot;}</stringProp>
        <stringProp name="Argument.metadata">=</stringProp>
      </elementProp>
    </collectionProp>
  </elementProp>
  <stringProp name="HTTPSampler.domain"></stringProp>
  <stringProp name="HTTPSampler.port"></stringProp>
```

```

<stringProp name="HTTPSampler.connect_timeout"></stringProp>
<stringProp name="HTTPSampler.response_timeout"></stringProp>
<stringProp name="HTTPSampler.protocol"></stringProp>
<stringProp name="HTTPSampler.contentEncoding"></stringProp>
<stringProp name="HTTPSampler.path">/ticket-monster/rest/ticketcategories</stringProp>
<stringProp name="HTTPSampler.method">POST</stringProp>
<boolProp name="HTTPSampler.follow_redirects">true</boolProp>
<boolProp name="HTTPSampler.auto_redirects">false</boolProp>
<boolProp name="HTTPSampler.use_keepalive">true</boolProp>
<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
<boolProp name="HTTPSampler.monitor">false</boolProp>
<stringProp name="HTTPSampler.embedded_url_re"></stringProp>
</HTTPSamplerProxy>
<hashTree/>
<LoopController guiclass="LoopControlPanel" testclass="LoopController" testname="Add all events" enabled="true">
  <boolProp name="LoopController.continue_forever">true</boolProp>
  <stringProp name="LoopController.loops">${EVENTS}</stringProp>
</LoopController>
<hashTree>
  <CounterConfig guiclass="CounterConfigGui" testclass="CounterConfig" testname="EventCounter" enabled="true">
    <stringProp name="CounterConfig.start">1</stringProp>
    <stringProp name="CounterConfig.end">${EVENTS}</stringProp>
    <stringProp name="CounterConfig.incr">1</stringProp>
    <stringProp name="CounterConfig.name">EVENT</stringProp>
    <stringProp name="CounterConfig.format"></stringProp>
    <boolProp name="CounterConfig.per_user">false</boolProp>
  </CounterConfig>
</hashTree/>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Add event" enabled="true">
  <boolProp name="HTTPSampler.postBodyRaw">true</boolProp>
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments">
    <collectionProp name="Arguments.arguments">
      <elementProp name="" elementType="HTTPArgument">
        <boolProp name="HTTPArgument.always_encode">false</boolProp>
        <stringProp
name="Argument.value">{&quot;name&quot;:&quot;&quot;,event${EVENT}&quot;,&quot;mediaItem&quot;:{&quot;id&quot;:1},&quot;category&quot;:{&quot;id&quot;:1},&quot;description&quot;:
:&quot;yada yada yada yada&quot;}</stringProp>
        <stringProp name="Argument.metadata">=</stringProp>
      </elementProp>
    </collectionProp>
  </elementProp>
  <stringProp name="HTTPSampler.domain"></stringProp>
  <stringProp name="HTTPSampler.port"></stringProp>
  <stringProp name="HTTPSampler.connect_timeout"></stringProp>
  <stringProp name="HTTPSampler.response_timeout"></stringProp>
  <stringProp name="HTTPSampler.protocol"></stringProp>

```

```

<stringProp name="HTTPSampler.contentEncoding"/></stringProp>
<stringProp name="HTTPSampler.path"/>/ticket-monster/rest/forge/events</stringProp>
<stringProp name="HTTPSampler.method">POST</stringProp>
<boolProp name="HTTPSampler.follow_redirects">true</boolProp>
<boolProp name="HTTPSampler.auto_redirects">false</boolProp>
<boolProp name="HTTPSampler.use_keepalive">true</boolProp>
<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
<boolProp name="HTTPSampler.monitor">false</boolProp>
<stringProp name="HTTPSampler.embedded_url_re"></stringProp>
</HTTPSamplerProxy>
<hashTree/>
</hashTree>
<LoopController guiclass="LoopControlPanel" testclass="LoopController" testname="Add all venues" enabled="true">
  <boolProp name="LoopController.continue_forever">true</boolProp>
  <stringProp name="LoopController.loops">${VENUES}</stringProp>
</LoopController>
<hashTree>
  <CounterConfig guiclass="CounterConfigGui" testclass="CounterConfig" testname="VenueCounter" enabled="true">
    <stringProp name="CounterConfig.start">1</stringProp>
    <stringProp name="CounterConfig.end">${VENUES}</stringProp>
    <stringProp name="CounterConfig.incr">1</stringProp>
    <stringProp name="CounterConfig.name">VENUE</stringProp>
    <stringProp name="CounterConfig.format"></stringProp>
    <boolProp name="CounterConfig.per_user">false</boolProp>
  </CounterConfig>
  <hashTree/>
  <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Add venue" enabled="true">
    <boolProp name="HTTPSampler.postBodyRaw">true</boolProp>
    <elementProp name="HTTPSampler.Arguments" elementType="Arguments">
      <collectionProp name="Arguments.arguments">
        <elementProp name="" elementType="HTTPArgument">
          <boolProp name="HTTPArgument.always_encode">false</boolProp>
          <stringProp name="Argument.value">{"name": "VENUE", "mediaItem": {"id": 1, "description": "yada yada yada", "capacity": 10000, "address": {"street": "venue street", "city": "venue city", "country": "venue country"}}}</stringProp>
          <stringProp name="Argument.metadata"></stringProp>
        </elementProp>
      </collectionProp>
    </elementProp>
    <stringProp name="HTTPSampler.domain"></stringProp>
    <stringProp name="HTTPSampler.port"></stringProp>
    <stringProp name="HTTPSampler.connect_timeout"></stringProp>
    <stringProp name="HTTPSampler.response_timeout"></stringProp>
    <stringProp name="HTTPSampler.protocol"></stringProp>
    <stringProp name="HTTPSampler.contentEncoding"></stringProp>
    <stringProp name="HTTPSampler.path">/ticket-monster/rest/forge/venues</stringProp>

```

```

<stringProp name="HTTPSampler.method">POST</stringProp>
<boolProp name="HTTPSampler.follow_redirects">true</boolProp>
<boolProp name="HTTPSampler.auto_redirects">false</boolProp>
<boolProp name="HTTPSampler.use_keepalive">true</boolProp>
<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
<boolProp name="HTTPSampler.monitor">false</boolProp>
<stringProp name="HTTPSampler.embedded_url_re"></stringProp>
</HTTPSamplerProxy>
<hashTree/>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Add section" enabled="true">
  <boolProp name="HTTPSampler.postBodyRaw">true</boolProp>
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments">
    <collectionProp name="Arguments.arguments">
      <elementProp name="" elementType="HTTPArgument">
        <boolProp name="HTTPArgument.always_encode">false</boolProp>
        <stringProp name="Argument.value">{"&name&quot;;&A&quot;;&description&quot;;&quot;;yada yada yada yada
yada&quot;;&quot;;numberOfRows&quot;;100,&quot;;rowCapacity&quot;;100,&quot;;capacity&quot;;50000,&quot;;venue&quot;;{"&id&quot;;${VENUE}}}</stringProp>
        <stringProp name="Argument.metadata">=</stringProp>
      </elementProp>
    </collectionProp>
  </elementProp>
  <stringProp name="HTTPSampler.domain"></stringProp>
  <stringProp name="HTTPSampler.port"></stringProp>
  <stringProp name="HTTPSampler.connect_timeout"></stringProp>
  <stringProp name="HTTPSampler.response_timeout"></stringProp>
  <stringProp name="HTTPSampler.protocol"></stringProp>
  <stringProp name="HTTPSampler.contentEncoding"></stringProp>
  <stringProp name="HTTPSampler.path">/ticket-monster/rest/sections</stringProp>
  <stringProp name="HTTPSampler.method">POST</stringProp>
  <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
  <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
  <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
  <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
  <boolProp name="HTTPSampler.monitor">false</boolProp>
  <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
</HTTPSamplerProxy>
<hashTree/>
<LoopController guiclass="LoopControlPanel" testclass="LoopController" testname="Add tickets" enabled="true">
  <boolProp name="LoopController.continue_forever">true</boolProp>
  <stringProp name="LoopController.loops">${EVENTS}</stringProp>
</LoopController>
<hashTree/>
<CounterConfig guiclass="CounterConfigGui" testclass="CounterConfig" testname="EventCounter" enabled="true">
  <stringProp name="CounterConfig.start">1</stringProp>
  <stringProp name="CounterConfig.end">${EVENTS}</stringProp>
  <stringProp name="CounterConfig.incr">1</stringProp>

```

```

<stringProp name="CounterConfig.name">EVENT</stringProp>
<stringProp name="CounterConfig.format"></stringProp>
<boolProp name="CounterConfig.per_user">>false</boolProp>
</CounterConfig>
<hashTree/>
<CounterConfig guiclass="CounterConfigGui" testclass="CounterConfig" testname="Ticket Counter" enabled="true">
  <stringProp name="CounterConfig.start">1</stringProp>
  <stringProp name="CounterConfig.end"></stringProp>
  <stringProp name="CounterConfig.incr">1</stringProp>
  <stringProp name="CounterConfig.name">TKT</stringProp>
  <stringProp name="CounterConfig.format"></stringProp>
  <boolProp name="CounterConfig.per_user">>false</boolProp>
</CounterConfig>
<hashTree/>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Add show" enabled="true">
  <boolProp name="HTTPSampler.postBodyRaw">>true</boolProp>
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments">
    <collectionProp name="Arguments.arguments">
      <elementProp name="" elementType="HTTPArgument">
        <boolProp name="HTTPArgument.always_encode">>false</boolProp>
        <stringProp name="Argument.value">{&quot;event&quot};{&quot;id&quot};{&quot;EVENT&quot}},&quot;venue&quot};{&quot;id&quot};{&quot;VENUE&quot}}</stringProp>
        <stringProp name="Argument.metadata">=</stringProp>
      </elementProp>
    </collectionProp>
  </elementProp>
  <stringProp name="HTTPSampler.domain"></stringProp>
  <stringProp name="HTTPSampler.port"></stringProp>
  <stringProp name="HTTPSampler.connect_timeout"></stringProp>
  <stringProp name="HTTPSampler.response_timeout"></stringProp>
  <stringProp name="HTTPSampler.protocol"></stringProp>
  <stringProp name="HTTPSampler.contentEncoding"></stringProp>
  <stringProp name="HTTPSampler.path">/ticket-monster/rest/forge/shows</stringProp>
  <stringProp name="HTTPSampler.method">POST</stringProp>
  <boolProp name="HTTPSampler.follow_redirects">>true</boolProp>
  <boolProp name="HTTPSampler.auto_redirects">>false</boolProp>
  <boolProp name="HTTPSampler.use_keepalive">>true</boolProp>
  <boolProp name="HTTPSampler.DO_MULTIPART_POST">>false</boolProp>
  <boolProp name="HTTPSampler.monitor">>false</boolProp>
  <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
</HTTPSamplerProxy>
<hashTree/>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Add performance" enabled="true">
  <boolProp name="HTTPSampler.postBodyRaw">>true</boolProp>
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments">
    <collectionProp name="Arguments.arguments">

```

```

    <elementProp name="" elementType="HTTPArgument">
      <boolProp name="HTTPArgument.always_encode">false</boolProp>
      <stringProp name="Argument.value">{"&quot;show&quot;:{"&quot;id&quot;:{"TKT"}},&quot;date&quot;:{"&quot;2020-11-13T06:00:00.000Z&quot;}}</stringProp>
      <stringProp name="Argument.metadata">=</stringProp>
    </elementProp>
  </collectionProp>
</elementProp>
<stringProp name="HTTPSampler.domain"></stringProp>
<stringProp name="HTTPSampler.port"></stringProp>
<stringProp name="HTTPSampler.connect_timeout"></stringProp>
<stringProp name="HTTPSampler.response_timeout"></stringProp>
<stringProp name="HTTPSampler.protocol"></stringProp>
<stringProp name="HTTPSampler.contentEncoding"></stringProp>
<stringProp name="HTTPSampler.path">/ticket-monster/rest/performances</stringProp>
<stringProp name="HTTPSampler.method">POST</stringProp>
<boolProp name="HTTPSampler.follow_redirects">true</boolProp>
<boolProp name="HTTPSampler.auto_redirects">false</boolProp>
<boolProp name="HTTPSampler.use_keepalive">true</boolProp>
<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
<boolProp name="HTTPSampler.monitor">false</boolProp>
<stringProp name="HTTPSampler.embedded_url_re"></stringProp>
</HTTPSamplerProxy>
<hashTree/>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Add ticketprice" enabled="true">
  <boolProp name="HTTPSampler.postBodyRaw">true</boolProp>
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments">
    <collectionProp name="Arguments.arguments">
      <elementProp name="" elementType="HTTPArgument">
        <boolProp name="HTTPArgument.always_encode">false</boolProp>
        <stringProp
name="Argument.value">{"&quot;show&quot;:{"&quot;id&quot;:{"TKT"}},&quot;section&quot;:{"&quot;id&quot;:{"VENUE"}},&quot;ticketCategory&quot;:{"&quot;id&quot;:1},&quot;price&quot;:200}</stringProp>
        <stringProp name="Argument.metadata">=</stringProp>
      </elementProp>
    </collectionProp>
  </elementProp>
  <stringProp name="HTTPSampler.domain"></stringProp>
  <stringProp name="HTTPSampler.port"></stringProp>
  <stringProp name="HTTPSampler.connect_timeout"></stringProp>
  <stringProp name="HTTPSampler.response_timeout"></stringProp>
  <stringProp name="HTTPSampler.protocol"></stringProp>
  <stringProp name="HTTPSampler.contentEncoding"></stringProp>
  <stringProp name="HTTPSampler.path">/ticket-monster/rest/ticketprices</stringProp>
  <stringProp name="HTTPSampler.method">POST</stringProp>
  <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
  <boolProp name="HTTPSampler.auto_redirects">false</boolProp>

```

```

    <boolProp name="HTTPSampler.use_keepalive">true</boolProp>

    <boolProp name="HTTPSampler.DO_MULTIPART_POST">>false</boolProp>

    <boolProp name="HTTPSampler.monitor">>false</boolProp>

    <stringProp name="HTTPSampler.embedded_url_re"></stringProp>

</HTTPSamplerProxy>

<hashTree/>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Add booking" enabled="true">

    <boolProp name="HTTPSampler.postBodyRaw">true</boolProp>

    <elementProp name="HTTPSampler.Arguments" elementType="Arguments">

        <collectionProp name="Arguments.arguments">

            <elementProp name="" elementType="HTTPArgument">

                <boolProp name="HTTPArgument.always_encode">>false</boolProp>

                <stringProp
name="Argument.value">{"&quot;ticketRequests&quot;:[{"&quot;ticketPrice&quot;:{"TKT"},&quot;quantity&quot;:2}],&quot;email&quot;:&quot;a@b.c&quot;,&quot;performance&quot;:&quot;uot;{"TKT}&quot;}</stringProp>

                <stringProp name="Argument.metadata">=</stringProp>

            </elementProp>

        </collectionProp>

    </elementProp>

    <stringProp name="HTTPSampler.domain"></stringProp>

    <stringProp name="HTTPSampler.port"></stringProp>

    <stringProp name="HTTPSampler.connect_timeout"></stringProp>

    <stringProp name="HTTPSampler.response_timeout"></stringProp>

    <stringProp name="HTTPSampler.protocol"></stringProp>

    <stringProp name="HTTPSampler.contentEncoding"></stringProp>

    <stringProp name="HTTPSampler.path">/ticket-monster/rest/bookings</stringProp>

    <stringProp name="HTTPSampler.method">POST</stringProp>

    <boolProp name="HTTPSampler.follow_redirects">true</boolProp>

    <boolProp name="HTTPSampler.auto_redirects">>false</boolProp>

    <boolProp name="HTTPSampler.use_keepalive">true</boolProp>

    <boolProp name="HTTPSampler.DO_MULTIPART_POST">>false</boolProp>

    <boolProp name="HTTPSampler.monitor">>false</boolProp>

    <stringProp name="HTTPSampler.embedded_url_re"></stringProp>

</HTTPSamplerProxy>

<hashTree/>

</hashTree>

</hashTree>

</hashTree>

</hashTree>

<ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup" testname="Throughput test" enabled="true">

    <stringProp name="ThreadGroup.on_sample_error">continue</stringProp>

    <elementProp name="ThreadGroup.main_controller" elementType="LoopController" guiclass="LoopControlPanel" testclass="LoopController" testname="Loop Controller"
enabled="true">

        <boolProp name="LoopController.continue_forever">>false</boolProp>

        <intProp name="LoopController.loops">-1</intProp>

    </elementProp>

    <stringProp name="ThreadGroup.num_threads">{USERS}</stringProp>

```

```

<stringProp name="ThreadGroup.ramp_time">${RAMPUP}</stringProp>
<longProp name="ThreadGroup.start_time">1479582534000</longProp>
<longProp name="ThreadGroup.end_time">1479582534000</longProp>
<boolProp name="ThreadGroup.scheduler">true</boolProp>
<stringProp name="ThreadGroup.duration">${__BeanShell(${RAMPUP}+${STEADYSTATE})}</stringProp>
<stringProp name="ThreadGroup.delay"></stringProp>
</ThreadGroup>
<hashTree>
<ConstantThroughputTimer guiclass="TestBeanGUI" testclass="ConstantThroughputTimer" testname="Constant Throughput Timer" enabled="true">
  <stringProp name="throughput">${__BeanShell(${IR}*60)}</stringProp>
  <intProp name="calcMode">2</intProp>
</ConstantThroughputTimer>
<hashTree/>
<RandomOrderController guiclass="RandomOrderControllerGui" testclass="RandomOrderController" testname="Random Order Controller" enabled="true"/>
<hashTree>
<ThroughputController guiclass="ThroughputControllerGui" testclass="ThroughputController" testname="New Connection" enabled="true">
  <intProp name="ThroughputController.style">1</intProp>
  <boolProp name="ThroughputController.perThread">true</boolProp>
  <intProp name="ThroughputController.maxThroughput">1</intProp>
  <stringProp name="ThroughputController.percentThroughput">${PERC_NEWCONN}</stringProp>
</ThroughputController>
<hashTree>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Get venues Keepalive=F" enabled="true">
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" enabled="true">
    <collectionProp name="Arguments.arguments"/>
  </elementProp>
  <stringProp name="HTTPSampler.domain"></stringProp>
  <stringProp name="HTTPSampler.port"></stringProp>
  <stringProp name="HTTPSampler.connect_timeout"></stringProp>
  <stringProp name="HTTPSampler.response_timeout"></stringProp>
  <stringProp name="HTTPSampler.protocol"></stringProp>
  <stringProp name="HTTPSampler.contentEncoding"></stringProp>
  <stringProp name="HTTPSampler.path">/ticket-monster/rest/venues</stringProp>
  <stringProp name="HTTPSampler.method">GET</stringProp>
  <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
  <boolProp name="HTTPSampler.auto_redirects">>false</boolProp>
  <boolProp name="HTTPSampler.use_keepalive">>false</boolProp>
  <boolProp name="HTTPSampler.DO_MULTIPART_POST">>false</boolProp>
  <boolProp name="HTTPSampler.monitor">>false</boolProp>
  <stringProp name="HTTPSampler.embedded_ur_l_re"></stringProp>
</HTTPSamplerProxy>
<hashTree/>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Get events Keepalive=F" enabled="true">
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" enabled="true">
    <collectionProp name="Arguments.arguments"/>

```

```

</elementProp>
<stringProp name="HTTPSampler.domain"></stringProp>
<stringProp name="HTTPSampler.port"></stringProp>
<stringProp name="HTTPSampler.connect_timeout"></stringProp>
<stringProp name="HTTPSampler.response_timeout"></stringProp>
<stringProp name="HTTPSampler.protocol"></stringProp>
<stringProp name="HTTPSampler.contentEncoding"></stringProp>
<stringProp name="HTTPSampler.path">/ticket-monster/rest/events</stringProp>
<stringProp name="HTTPSampler.method">GET</stringProp>
<boolProp name="HTTPSampler.follow_redirects">true</boolProp>
<boolProp name="HTTPSampler.auto_redirects">false</boolProp>
<boolProp name="HTTPSampler.use_keepalive">false</boolProp>
<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
<boolProp name="HTTPSampler.monitor">false</boolProp>
<stringProp name="HTTPSampler.embedded_url_re"></stringProp>
</HTTPSamplerProxy>
<hashTree/>
</hashTree>
<ThroughputController guiclass="ThroughputControllerGui" testclass="ThroughputController" testname="Read Entities" enabled="true">
  <intProp name="ThroughputController.style">1</intProp>
  <boolProp name="ThroughputController.perThread">true</boolProp>
  <intProp name="ThroughputController.maxThroughput">1</intProp>
  <stringProp name="ThroughputController.percentThroughput">${PERC_READ}</stringProp>
</ThroughputController>
<hashTree>
  <RandomVariableConfig guiclass="TestBeanGUI" testclass="RandomVariableConfig" testname="EID" enabled="true">
    <stringProp name="maximumValue">${_BeanShell(${VENUES}*${EVENTS})}</stringProp>
    <stringProp name="minimumValue">1</stringProp>
    <stringProp name="outputFormat"></stringProp>
    <boolProp name="perThread">true</boolProp>
    <stringProp name="randomSeed"></stringProp>
    <stringProp name="variableName">EID</stringProp>
  </RandomVariableConfig>
  <hashTree/>
  <RandomVariableConfig guiclass="TestBeanGUI" testclass="RandomVariableConfig" testname="VENUE" enabled="true">
    <stringProp name="maximumValue">${VENUES}</stringProp>
    <stringProp name="minimumValue">1</stringProp>
    <stringProp name="outputFormat"></stringProp>
    <boolProp name="perThread">true</boolProp>
    <stringProp name="randomSeed"></stringProp>
    <stringProp name="variableName">VENUE</stringProp>
  </RandomVariableConfig>
  <hashTree/>
  <RandomOrderController guiclass="RandomOrderControllerGui" testclass="RandomOrderController" testname="Random Order Controller" enabled="true"/>
  <hashTree>

```

```
<LoopController guiclass="LoopControlPanel" testclass="LoopController" testname="Image download" enabled="true">
  <boolProp name="LoopController.continue_forever">true</boolProp>
  <stringProp name="LoopController.loops">10</stringProp>
</LoopController>
<hashTree>
  <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Get splash image" enabled="false">
    <elementProp name="HTTPSampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" enabled="true">
      <collectionProp name="Arguments.arguments"/>
    </elementProp>
    <stringProp name="HTTPSampler.domain"></stringProp>
    <stringProp name="HTTPSampler.port"></stringProp>
    <stringProp name="HTTPSampler.connect_timeout"></stringProp>
    <stringProp name="HTTPSampler.response_timeout"></stringProp>
    <stringProp name="HTTPSampler.protocol"></stringProp>
    <stringProp name="HTTPSampler.contentEncoding"></stringProp>
    <stringProp name="HTTPSampler.path">/ticket-monster/resources/img/splash-ticketmonster.png</stringProp>
    <stringProp name="HTTPSampler.method">GET</stringProp>
    <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
    <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
    <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
    <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
    <boolProp name="HTTPSampler.monitor">false</boolProp>
    <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
  </HTTPSamplerProxy>
</hashTree/>
  <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Get cached image" enabled="true">
    <elementProp name="HTTPSampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" enabled="true">
      <collectionProp name="Arguments.arguments"/>
    </elementProp>
    <stringProp name="HTTPSampler.domain"></stringProp>
    <stringProp name="HTTPSampler.port"></stringProp>
    <stringProp name="HTTPSampler.connect_timeout"></stringProp>
    <stringProp name="HTTPSampler.response_timeout"></stringProp>
    <stringProp name="HTTPSampler.protocol"></stringProp>
    <stringProp name="HTTPSampler.contentEncoding"></stringProp>
    <stringProp name="HTTPSampler.path">/ticket-monster/rest/media/1</stringProp>
    <stringProp name="HTTPSampler.method">GET</stringProp>
    <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
    <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
    <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
    <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
    <boolProp name="HTTPSampler.monitor">false</boolProp>
    <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
  </HTTPSamplerProxy>
</hashTree/>
```

```
</hashTree>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Get venues" enabled="true">
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" enabled="true">
    <collectionProp name="Arguments.arguments"/>
  </elementProp>
  <stringProp name="HTTPSampler.domain"></stringProp>
  <stringProp name="HTTPSampler.port"></stringProp>
  <stringProp name="HTTPSampler.connect_timeout"></stringProp>
  <stringProp name="HTTPSampler.response_timeout"></stringProp>
  <stringProp name="HTTPSampler.protocol"></stringProp>
  <stringProp name="HTTPSampler.contentEncoding"></stringProp>
  <stringProp name="HTTPSampler.path">/ticket-monster/rest/venues</stringProp>
  <stringProp name="HTTPSampler.method">GET</stringProp>
  <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
  <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
  <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
  <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
  <boolProp name="HTTPSampler.monitor">false</boolProp>
  <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
</HTTPSamplerProxy>

<hashTree/>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Get events" enabled="true">
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" enabled="true">
    <collectionProp name="Arguments.arguments"/>
  </elementProp>
  <stringProp name="HTTPSampler.domain"></stringProp>
  <stringProp name="HTTPSampler.port"></stringProp>
  <stringProp name="HTTPSampler.connect_timeout"></stringProp>
  <stringProp name="HTTPSampler.response_timeout"></stringProp>
  <stringProp name="HTTPSampler.protocol"></stringProp>
  <stringProp name="HTTPSampler.contentEncoding"></stringProp>
  <stringProp name="HTTPSampler.path">/ticket-monster/rest/events</stringProp>
  <stringProp name="HTTPSampler.method">GET</stringProp>
  <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
  <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
  <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
  <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
  <boolProp name="HTTPSampler.monitor">false</boolProp>
  <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
</HTTPSamplerProxy>

<hashTree/>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Get 1 show" enabled="true">
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" enabled="true">
    <collectionProp name="Arguments.arguments"/>
  </elementProp>
  <stringProp name="HTTPSampler.domain"></stringProp>
  <stringProp name="HTTPSampler.port"></stringProp>
  <stringProp name="HTTPSampler.connect_timeout"></stringProp>
  <stringProp name="HTTPSampler.response_timeout"></stringProp>
  <stringProp name="HTTPSampler.protocol"></stringProp>
  <stringProp name="HTTPSampler.contentEncoding"></stringProp>
  <stringProp name="HTTPSampler.path"></stringProp>
  <stringProp name="HTTPSampler.method"></stringProp>
  <boolProp name="HTTPSampler.follow_redirects"></boolProp>
  <boolProp name="HTTPSampler.auto_redirects"></boolProp>
  <boolProp name="HTTPSampler.use_keepalive"></boolProp>
  <boolProp name="HTTPSampler.DO_MULTIPART_POST"></boolProp>
  <boolProp name="HTTPSampler.monitor"></boolProp>
  <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
</HTTPSamplerProxy>

<hashTree/>
```

```
<stringProp name="HTTPSampler.domain"></stringProp>
<stringProp name="HTTPSampler.port"></stringProp>
<stringProp name="HTTPSampler.connect_timeout"></stringProp>
<stringProp name="HTTPSampler.response_timeout"></stringProp>
<stringProp name="HTTPSampler.protocol"></stringProp>
<stringProp name="HTTPSampler.contentEncoding"></stringProp>
<stringProp name="HTTPSampler.path">/ticket-monster/rest/shows/${EID}</stringProp>
<stringProp name="HTTPSampler.method">GET</stringProp>
<boolProp name="HTTPSampler.follow_redirects">true</boolProp>
<boolProp name="HTTPSampler.auto_redirects">false</boolProp>
<boolProp name="HTTPSampler.use_keepalive">true</boolProp>
<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
<boolProp name="HTTPSampler.monitor">false</boolProp>
<stringProp name="HTTPSampler.embedded_url_re"></stringProp>
</HTTPSamplerProxy>
<hashTree/>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Get 1 performance" enabled="true">
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" enabled="true">
    <collectionProp name="Arguments.arguments"/>
  </elementProp>
  <stringProp name="HTTPSampler.domain"></stringProp>
  <stringProp name="HTTPSampler.port"></stringProp>
  <stringProp name="HTTPSampler.connect_timeout"></stringProp>
  <stringProp name="HTTPSampler.response_timeout"></stringProp>
  <stringProp name="HTTPSampler.protocol"></stringProp>
  <stringProp name="HTTPSampler.contentEncoding"></stringProp>
  <stringProp name="HTTPSampler.path">/ticket-monster/rest/performances/${EID}</stringProp>
  <stringProp name="HTTPSampler.method">GET</stringProp>
  <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
  <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
  <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
  <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
  <boolProp name="HTTPSampler.monitor">false</boolProp>
  <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
</HTTPSamplerProxy>
<hashTree/>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Get 1 ticketprice" enabled="true">
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" enabled="true">
    <collectionProp name="Arguments.arguments"/>
  </elementProp>
  <stringProp name="HTTPSampler.domain"></stringProp>
  <stringProp name="HTTPSampler.port"></stringProp>
  <stringProp name="HTTPSampler.connect_timeout"></stringProp>
  <stringProp name="HTTPSampler.response_timeout"></stringProp>
  <stringProp name="HTTPSampler.protocol"></stringProp>
```

```

<stringProp name="HTTPSampler.contentEncoding"></stringProp>

<stringProp name="HTTPSampler.path">/ticket-monster/rest/ticketprices/${EID}</stringProp>

<stringProp name="HTTPSampler.method">GET</stringProp>

<boolProp name="HTTPSampler.follow_redirects">true</boolProp>

<boolProp name="HTTPSampler.auto_redirects">false</boolProp>

<boolProp name="HTTPSampler.use_keepalive">true</boolProp>

<boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>

<boolProp name="HTTPSampler.monitor">false</boolProp>

<stringProp name="HTTPSampler.embedded_url_re"></stringProp>

</HTTPSamplerProxy>

<hashTree/>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Get 1 section" enabled="true">

  <elementProp name="HTTPSampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" enabled="true">

    <collectionProp name="Arguments.arguments"/>

  </elementProp>

  <stringProp name="HTTPSampler.domain"></stringProp>

  <stringProp name="HTTPSampler.port"></stringProp>

  <stringProp name="HTTPSampler.connect_timeout"></stringProp>

  <stringProp name="HTTPSampler.response_timeout"></stringProp>

  <stringProp name="HTTPSampler.protocol"></stringProp>

  <stringProp name="HTTPSampler.contentEncoding"></stringProp>

  <stringProp name="HTTPSampler.path">/ticket-monster/rest/sections/${VENUE}</stringProp>

  <stringProp name="HTTPSampler.method">GET</stringProp>

  <boolProp name="HTTPSampler.follow_redirects">true</boolProp>

  <boolProp name="HTTPSampler.auto_redirects">false</boolProp>

  <boolProp name="HTTPSampler.use_keepalive">true</boolProp>

  <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>

  <boolProp name="HTTPSampler.monitor">false</boolProp>

  <stringProp name="HTTPSampler.embedded_url_re"></stringProp>

</HTTPSamplerProxy>

<hashTree/>

</hashTree>

</hashTree>

<ThroughputController guiclass="ThroughputControllerGui" testclass="ThroughputController" testname="Book Tickets" enabled="true">

  <intProp name="ThroughputController.style">1</intProp>

  <boolProp name="ThroughputController.perThread">true</boolProp>

  <intProp name="ThroughputController.maxThroughput">1</intProp>

  <stringProp name="ThroughputController.percentThroughput">${PERC_BOOK}</stringProp>

</ThroughputController>

<hashTree>

  <RandomVariableConfig guiclass="TestBeanGUI" testclass="RandomVariableConfig" testname="EID" enabled="true">

    <stringProp name="maximumValue">${__Beanshell(${VENUES}*${EVENTS})}</stringProp>

    <stringProp name="minimumValue">1</stringProp>

    <stringProp name="outputFormat"></stringProp>

    <boolProp name="perThread">true</boolProp>

```

```

    <stringProp name="randomSeed"></stringProp>

    <stringProp name="variableName">EID</stringProp>

</RandomVariableConfig>

<hashTree/>

<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Add booking" enabled="true">

    <boolProp name="HTTPSampler.postBodyRaw">true</boolProp>

    <elementProp name="HTTPSampler.Arguments" elementType="Arguments">

        <collectionProp name="Arguments.arguments">

            <elementProp name="" elementType="HTTPArgument">

                <boolProp name="HTTPArgument.always_encode">false</boolProp>

                <stringProp
name="Argument.value">{&quot;ticketRequests&quot;:[{&quot;ticketPrice&quot;:${EID},&quot;quantity&quot;:10}],&quot;email&quot;:&quot;a@b.c&quot;,&quot;performance&quot;:&
quot;${EID}&quot;}</stringProp>

                <stringProp name="Argument.metadata">=</stringProp>

            </elementProp>

        </collectionProp>

    </elementProp>

    <stringProp name="HTTPSampler.domain"></stringProp>

    <stringProp name="HTTPSampler.port"></stringProp>

    <stringProp name="HTTPSampler.connect_timeout"></stringProp>

    <stringProp name="HTTPSampler.response_timeout"></stringProp>

    <stringProp name="HTTPSampler.protocol"></stringProp>

    <stringProp name="HTTPSampler.contentEncoding"></stringProp>

    <stringProp name="HTTPSampler.path">/ticket-monster/rest/bookings</stringProp>

    <stringProp name="HTTPSampler.method">POST</stringProp>

    <boolProp name="HTTPSampler.follow_redirects">true</boolProp>

    <boolProp name="HTTPSampler.auto_redirects">false</boolProp>

    <boolProp name="HTTPSampler.use_keepalive">true</boolProp>

    <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>

    <boolProp name="HTTPSampler.monitor">false</boolProp>

    <stringProp name="HTTPSampler.embedded_ur_l_re"></stringProp>

</HTTPSamplerProxy>

<hashTree/>

</hashTree>

<ThroughputController guiclass="ThroughputControllerGui" testclass="ThroughputController" testname="Delete Tickets" enabled="true">

    <intProp name="ThroughputController.style">1</intProp>

    <boolProp name="ThroughputController.perThread">true</boolProp>

    <intProp name="ThroughputController.maxThroughput">1</intProp>

    <stringProp name="ThroughputController.percentThroughput">${PERC_DELETE}</stringProp>

</ThroughputController>

<hashTree>

<CounterConfig guiclass="CounterConfigGui" testclass="CounterConfig" testname="DELID" enabled="true">

    <stringProp name="CounterConfig.start">1</stringProp>

    <stringProp name="CounterConfig.end"></stringProp>

    <stringProp name="CounterConfig.incr">1</stringProp>

    <stringProp name="CounterConfig.name">DELID</stringProp>


```

```
<stringProp name="CounterConfig.format"></stringProp>
<boolProp name="CounterConfig.per_user">false</boolProp>
</CounterConfig>
<hashTree/>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Get 1 booking" enabled="true">
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" enabled="true">
    <collectionProp name="Arguments.arguments"/>
  </elementProp>
  <stringProp name="HTTPSampler.domain"></stringProp>
  <stringProp name="HTTPSampler.port"></stringProp>
  <stringProp name="HTTPSampler.connect_timeout"></stringProp>
  <stringProp name="HTTPSampler.response_timeout"></stringProp>
  <stringProp name="HTTPSampler.protocol"></stringProp>
  <stringProp name="HTTPSampler.contentEncoding"></stringProp>
  <stringProp name="HTTPSampler.path">/ticket-monster/rest/bookings/${DELID}</stringProp>
  <stringProp name="HTTPSampler.method">GET</stringProp>
  <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
  <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
  <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
  <boolProp name="HTTPSampler.DQ_MULTIPART_POST">false</boolProp>
  <boolProp name="HTTPSampler.monitor">false</boolProp>
  <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
</HTTPSamplerProxy>
<hashTree/>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Delete Booking" enabled="true">
  <elementProp name="HTTPSampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel" testclass="Arguments" enabled="true">
    <collectionProp name="Arguments.arguments"/>
  </elementProp>
  <stringProp name="HTTPSampler.domain"></stringProp>
  <stringProp name="HTTPSampler.port"></stringProp>
  <stringProp name="HTTPSampler.connect_timeout"></stringProp>
  <stringProp name="HTTPSampler.response_timeout"></stringProp>
  <stringProp name="HTTPSampler.protocol"></stringProp>
  <stringProp name="HTTPSampler.contentEncoding"></stringProp>
  <stringProp name="HTTPSampler.path">/ticket-monster/rest/bookings/${DELID}</stringProp>
  <stringProp name="HTTPSampler.method">DELETE</stringProp>
  <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
  <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
  <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
  <boolProp name="HTTPSampler.DQ_MULTIPART_POST">false</boolProp>
  <boolProp name="HTTPSampler.monitor">false</boolProp>
  <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
</HTTPSamplerProxy>
<hashTree/>
</hashTree>
```

```
</hashTree>
</hashTree>
<ResultCollector guiclass="SummaryReport" testclass="ResultCollector" testname="Summary Report" enabled="true">
  <boolProp name="ResultCollector.error_logging">false</boolProp>
  <objProp>
    <name>saveConfig</name>
    <value class="SampleSaveConfiguration">
      <time>true</time>
      <latency>true</latency>
      <timestamp>true</timestamp>
      <success>true</success>
      <label>true</label>
      <code>true</code>
      <message>false</message>
      <threadName>false</threadName>
      <dataType>false</dataType>
      <encoding>false</encoding>
      <assertions>false</assertions>
      <subresults>false</subresults>
      <responseData>false</responseData>
      <samplerData>false</samplerData>
      <xml>false</xml>
      <fieldNames>true</fieldNames>
      <responseHeaders>false</responseHeaders>
      <requestHeaders>false</requestHeaders>
      <responseDataOnError>false</responseDataOnError>
      <saveAssertionResultsFailureMessage>false</saveAssertionResultsFailureMessage>
      <assertionsResultsToSave>0</assertionsResultsToSave>
      <bytes>true</bytes>
      <url>true</url>
      <hostname>true</hostname>
      <threadCounts>true</threadCounts>
      <sampleCount>true</sampleCount>
      <idleTime>true</idleTime>
      <connectTime>true</connectTime>
    </value>
  </objProp>
  <stringProp name="filename">/root/run.csv</stringProp>
</ResultCollector>
<hashTree/>
</hashTree>
</hashTree>
</jmeterTestPlan>
```

¹ <http://www.jboss.org/ticket-monster/>

² <http://www.h2database.com/html/main.html>

³ <http://docs.jboss.org/jbossdeveloper/2.5/ticket-monster-2.5.0.Final.pdf>

⁴ 'Vendors' and 'Customers' are only logical roles we use to discuss use cases. The application doesn't support authentication or access control.

⁵ <http://jmeter.apache.org/>

⁶ Asynchronous Java and XML - [https://en.wikipedia.org/wiki/Ajax_\(programming\)](https://en.wikipedia.org/wiki/Ajax_(programming))

⁷ This is a sample chart from a random experiment, used to illustrate the results data collected. The results aren't intended to indicate actual performance.

⁸ More information on Intel Resource Director Technology can be found at

<http://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html>

⁹ The focus of this document is to discuss about last level cache. The term 'cache' without any qualifier always refers to last level cache in this document.

¹⁰ A more complex example for using Intel RDT, from the perspective of a Cloud Service Provider (CSP) can be found in [<link to whitepaper>](#)

¹¹ The following link provides more information Intel RDT support on different Xen versions -

https://wiki.xenproject.org/wiki/Intel_Platform_QoS_Technologies

¹² "The STREAM benchmark is a simple synthetic benchmark program that measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels.... STREAM benchmark is specifically designed to work with datasets much larger than the available cache on any given system, so that the results are (presumably) more indicative of the performance of very large, vector style applications." (Source: <http://www.cs.virginia.edu/stream/ref.html>)

¹³ Please check the Xen documentation for caveats around using CAT -

<http://xenbits.xen.org/docs/unstable/misc/xl-psr.html>

¹⁴ Support for Memory Bandwidth Allocation will be available in Intel Xeon scalable processor family. It is not yet integrated to Xen at the time of writing this document.