

# Vignette High Performance Delivery 7.0 Beta With Portal 7.4: Performance Snapshot

Version 0.9  
Vignette Performance Engineering  
February 6 2008



Vignette Corporation  
1301 South MoPac Expressway  
Austin, TX 78746  
Phone: 1.512.741.4300  
Fax: 1.512.741.1403  
<http://www.vignette.com>

"Copyright © 2008 Vignette Corporation. All rights reserved. Vignette, Vignette Portal and the Vignette logo are either trademarks or registered trademarks of Vignette Corporation. All other company names and product names may be trademarks or registered trademarks of their respective companies or owners."

## **Audience**

The document is intended for individuals interested in understanding the potential performance benefits of the Vignette High Performance Delivery (VHPD) product.

## **Disclaimer**

This document is provided for information purposes only and Vignette reserves the right to change the contents of this document and the features or functionalities of its products and services at any time without notice. Vignette does not warrant, guarantee, or make representations concerning the contents of this document. All information is provided "AS-IS," without express or implied warranties of any kind including, without limitation, the warranties of merchantability, fitness for a particular purpose, quality and title. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the prior written consent of Vignette.

Performance tests and results described in this document were obtained with a beta version of Vignette High Performance Delivery and a development build of Vignette Portal 7.4 using a specific configuration of computers systems and components. Testing was performed by Vignette personnel on Vignette systems in the Vignette Performance Laboratory in Austin, Texas. Actual results may vary depending on a broad range of implementation factors, including, but not limited to, hardware platform, software configuration, network parameters and system use patterns. You may not achieve the same or similar results or benefits even if you use the same or similar equipment and/or software applications. Nothing in this document is considered to be part of any product documentation or specification for any purpose.

## Table Of Contents

<b>Audience .....</b>	<b>2</b>
<b>Disclaimer .....</b>	<b>2</b>
<b>1. Executive Summary.....</b>	<b>4</b>
<b>2. Test Overview .....</b>	<b>6</b>
<b>2.1 Test Environment Topology.....</b>	<b>6</b>
<b>3. Results .....</b>	<b>10</b>
<b>3.1 Test Methodology.....</b>	<b>10</b>
<b>3.2 Varying the percentage of fully-cached pages (SiteA, Plugin mode) ..</b>	<b>10</b>
<b>3.3 Separately cached portlets using SSI (SiteA, Plugin mode) .....</b>	<b>13</b>
<b>3.4 Dynamically-generated portlets using SSI (SiteA, Plugin mode).....</b>	<b>16</b>
<b>3.5 Dynamically-generated AJAX portlets (SiteA, Plugin mode).....</b>	<b>19</b>
<b>3.6 Sending requests to the application server directly (SiteA, Plugin mode vs. Servlet Filter mode).....</b>	<b>22</b>
<b>3.7 Varying the processing capability of the application server .....</b>	<b>25</b>
<b>3.8 Varying page complexity .....</b>	<b>28</b>
<b>4.0 Caveats .....</b>	<b>30</b>

# 1. Executive Summary

Vignette High Performance Delivery 7.0 (VHPD) is designed to provide highly flexible and configurable caching at the page level for a number of different back-end page generators, including the Vignette Portal. This document presents a snapshot of performance results obtained using simple browsing tests on the Beta build of VHPD and a corresponding development build of Vignette Portal 7.4 (referred to from here on as "Portal") during January 2008.

The primary objective of this document is to provide Vignette personnel with an indication of the **potential** performance benefits of VHPD under a variety of conditions. The tests used here are specially designed to easily support the ability to vary the proportion of cached and uncached content: for example, each Portal page has 10 portlets whose caching parameters can be set independently of each other, and each Portal site has exactly 100 pages, whose caching parameters can also be set independently of each other. To reduce the effort and the amount of time taken for testing while simulating fairly heavy workloads, the tests were run with zero think times between requests. This is in contrast to real-world use cases where users typically spend several seconds or even minutes between successive requests. Consequently, the **results provided here are not as realistic as tests that emulate user think times, and should not be used for capacity planning or sizing.** Furthermore, as with any product, the performance of both VHPD and Portal is likely to change over time as development and testing proceeds, so the results provided here should not be assumed to apply to the release version of either product. This document is also not intended to provide technical details of the VHPD product or its interactions with Portal.

In our tests, VHPD was configured in two modes. In one mode (referred to from here on as the "**Plugin**" mode), an Apache web server with a VHPD plugin was used as a front end. In the other mode (referred to from here on as the "**Servlet Filter**" mode), no front-end web server was used, and VHPD was deployed using servlet filters in the same servlet engine/application server (Tomcat 5.5.16) used for the Portal. The tests consisted of guest users browsing the pages of a Portal configuration. A number of different test scenarios were used for each mode, in which several parameters impacting performance were varied. Table 1 summarizes some of the parameter variations for which results are provided in this paper. Unless otherwise specified, the "baseline" values for each parameter set were used in our tests: so, for example, unless otherwise specified, the number of application server processor threads was 32.

Parameter	Range of Values Tested
Page Cache Settings	No pages cached (baseline against which other results are compared) Percentage of pages fully cached: 100%, 90%, 80%, 50%
Portlet Cache Settings	Number of separately cached portlets per cached page, rendered using SSI: 1, 2, 5 (baseline = 0) Number of dynamic portlets per cached page, rendered using SSI: 1, 2, 5 (baseline = 0) Number of dynamic portlets per cached page, rendered using AJAX: 1, 2, 5 (baseline = 0)
VHPD Architecture	Plugin mode: Front-end web server with VHPD plugin Servlet Filter mode: No front-end web server; requests handled directly by application server
Application Server Processing Power	Number of hardware threads of execution: 32 (8 cores, baseline), 16 (4 cores), 8 (2 cores)
Portal Page Complexity	SiteA: Nominal single-user response time for an uncached page is 0.25 second, CPU intensive (baseline)

	<p>SiteB: Nominal single-user response time for an uncached page is 1.0 second, CPU intensive</p> <p>SiteC: Nominal single-user response time for an uncached page is 2.0 second, includes 1 second of sleep and 1 second of CPU-intensive operations.</p> <p>SiteD: Nominal single-user response time for an uncached page is 0.1 second, CPU intensive</p>
--	--

**Table 1**

Note that not all parameter combinations listed in Table 1 are reported on here: for example, in the VHPD release we tested, SSI (Server Side Include) capability is only supported for configurations where a web server plugin is used, and so SSI-based portlet rendering was not tested for the Servlet Filter mode.

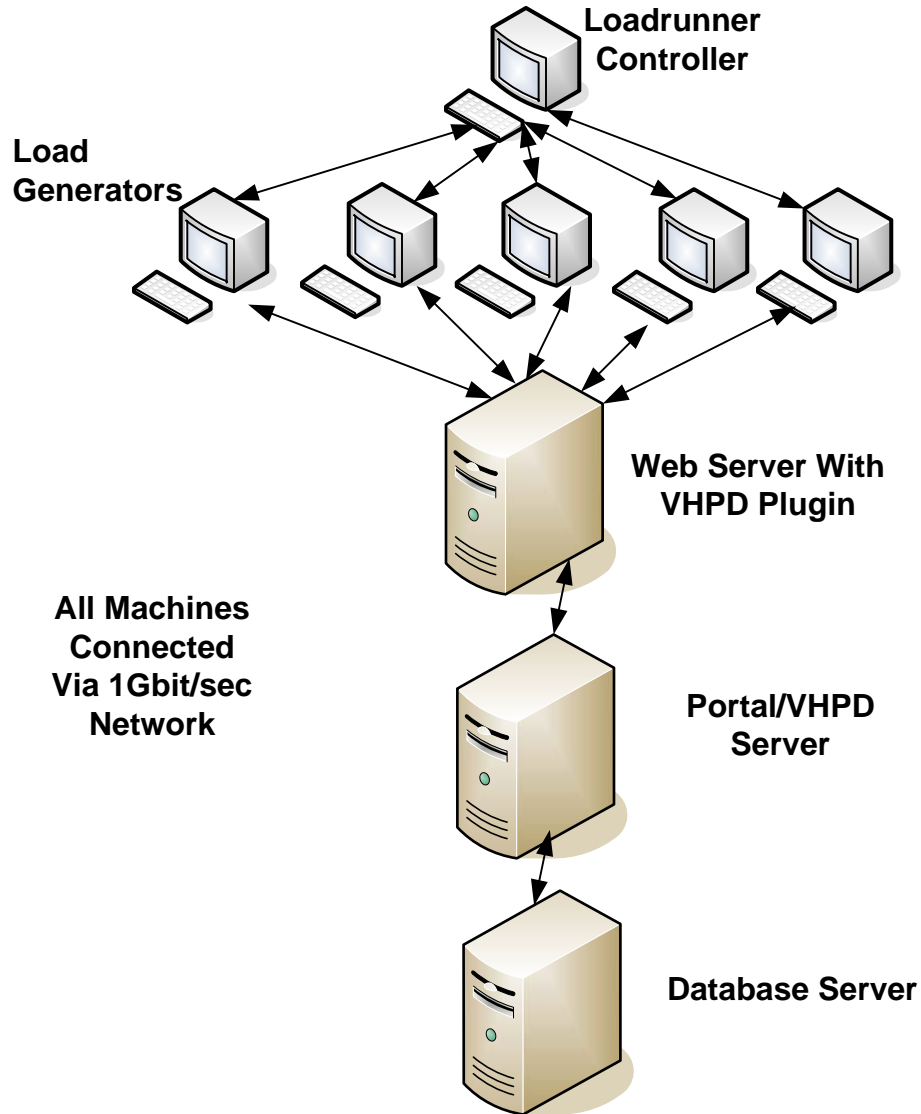
For all the tests, the page throughput and average page response times were measured as the number of virtual users was increased. To simplify the presentation, we also present some results in this paper as **throughput ratios** and **response time ratios** compared to the corresponding baseline values for the same offered virtual user load. For example, in answer to the question, “How much more throughput, compared to the fully uncached configuration, is supported when 90% of the pages are fully cached and the Plugin mode is used?” we provide the ratio of the throughput measured with 90% pages cached and 32 zero-think-time virtual users to the throughput measured when none of the pages are cached and 32 zero-think-time virtual users load the system. The number of virtual users for which the throughput and response time ratios are provided is selected based on resource utilization levels: typically, the bottleneck resource (e.g., the application server processors) is about 90% busy for the baseline case at the user load selected for computing the ratios.

**Our results indicate that VHPD provides dramatic performance improvements under a wide variety of conditions. For example, for several of the sites that we tested, if 80% of the pages are cached, VHPD provided an improvement of approximately 450-500% in peak sustainable throughput. With 90% of the pages cached, the throughput improvement is even higher, often exceeding 800%. Similar improvements were seen even when a small number of components (portlets) of each cached page are generated dynamically: when two portlets of each page are generated dynamically using server-side includes, for example, a throughput gain of about 400% was measured.** The overall result varies based on a number of factors, such as the baseline cost of generating an uncached page, the proportion of uncached content, and so on.

The remainder of this document describes the test architecture and methodology and details of the results.

## 2. Test Overview

### 2.1 Test Environment Topology



**Figure 1: Test Network**

Figure 1 illustrates the servers used during the testing. A single instance of Portal 7.4 was deployed on a Tomcat5.5.16 servlet engine running on a Sun T2000 server. In the Plugin mode, the VHPD plugin and a cache manager resided on the web server, and the remaining subcomponents of VHPD resided on the same server as the portal. In the Servlet Filter mode, all the subcomponents of VHPD resided on the same server as the portal (the web server was not used, and page requests were sent directly from the load generators to the portal/VHPD server). The database tables for the Portal and for VHPD were located on the Oracle 9.2 database server. The load was generated with Loadrunner 8.5 using up to six Windows machines, with one serving as the Loadrunner controller and the remaining as load

generators. All the machines were on the same subnet and were connected to a 1Gbit/sec switch; the load generators, the web server, and the Portal/VHPD server each had Gigabit Ethernet cards.

Key specifications of the various machines used are shown in Table 2.

Server Role	Processors/Cores	Memory	OS/Software
Portal/VHPD Server	Sun T2000, 8x 1.2 GHz cores, 32 threads of hardware execution (8 cores, 4 threads each)	32 GB	Solaris 10, single default zone; Tomcat 5.5.16 using Sun JDK 1.5.0_08; Vignette Portal 7.4 and VHPD Beta Builds from December 2007
Database Server	SunFire V440, 4x 1 GHz processors	8 GB	Solaris 9; Oracle 9.2
Web Server	Intel Xeon, 4x 3 GHz processors	3.8 GB	Windows Server 2003; Apache 2.2; VHPD Plugin
LoadRunner Controller	Intel Xeon, 2x 3GHz processors	3.8 MB	Windows Server 2000; Loadrunner 8.5
Load generators (five machines)	Intel Xeon, 4x 1.9GHz or 2x 3GHz processors	3.8 GB	Windows Server 2003; Loadrunner 8.5

**Table 2**

## 2.2 Portal Deployment

Some of the key portal deployment details are listed below.

- *System Architecture:* A single instance of the portal was installed and deployed; clustering was not used.
- *Users and Groups:* All the tests used guest users.
- *Portal Sites and Pages:* Each test used a single Portal site with 100 navigation items and 100 pages. Four different sites were used, with different levels of processing involved in each portlet as described below. The average size of the HTML corresponding to each page for all the sites was about 70 KB. Thus, with 100 total pages and only guest users accessing the site, the total amount of data cached by VHPD in any given test was about 7-8 MB. All pages were locked.
- *Portlets:* All pages included 10 JSR168 portlets. The portlets were shared across the pages: i.e., each page in a site had the same 10 portlets. Each portlet consisted of a small amount of text and some JSP code involving integer processing using random numbers. By varying the number of random numbers used, the amount of time needed to generate each portlet was changed. Details of the different portlets used in various sites are provided in Section 3.8.

## 2.3 User Operations Emulated During Testing

During each test, the load generator scripts (implemented using Loadrunner 8.5 from Hewlett Packard) emulated the following sequence of operations:

```

loop {
    start a new user session as a guest user;
    for (i=0; i < 10; i++){
        request a randomly-selected page from among the 100 pages of the site;
    }
}

```

Note that all these operations are read-only, and that there were no cache invalidation operations during our testing. That is, once a page was cached by VHPD, it stayed cached for the duration of our tests.

## 2.4 Performance Metrics and Terminology

Table 3 defines some of the terms used in describing the tests and results.

<b>Virtual Users</b>	The number of LoadRunner virtual users that are used in the test. In general, one virtual user may represent a load corresponding to many real-world users, depending on the test and the think times between successive requests.
<b>Think Time</b>	Think time is the time that a virtual user spends before submitting a request for a subsequent page. Think time was set to zero for all our tests.
<b>Average Page Response Time (sec)</b>	For pages that do not use AJAX, the total time to load a Portal page with all its elements in seconds. <b><u>For portlets whose content is dynamically generated via AJAX, while the surrounding page skeleton is cached, the reported response time is the time taken to return the cached page skeleton. The content of each AJAX-configured portlet is requested separately from the request for the page skeleton in our script, and the response times for these portlets is not included in the reported response time for the page.</u></b>
<b>Page Views per second (PV/sec)</b>	The average number of page views (requests) processed by the system every second, representing the throughput of the system.
<b>CPU Utilization (%)</b>	The total percentage of time that the processing elements (CPUs, cores or threads) were busy (includes user, system and all other non-idle time). When there are multiple processing elements per server, this is the average utilization across the elements.

**Table 3**

## 2.5 Repeatability of Results

For each number of virtual users, a constant load was applied for ten minutes. The average throughput and response time, obtained from the middle eight minutes of each run, is reported here.

In general we saw relatively minor variability in the results for a particular set of parameters, either within a given run or across different runs (in some cases we tried multiple runs). A special "caching" run was executed prior to each test where content was cached. The caching run consisted of one guest user accessing each of the pages of the Portal site in sequence, so that all the pages are cached by VHPD and the cache is "warmed up".

## 2.6 Tuning Parameters

Since our main goal was to compare the performance for various levels of caching on identical hardware/software platforms, we did not experiment with a large number of tuning parameters. Some of the key parameters we used include:

Tomcat JVM Heap Size: 3 Gigabytes (-Xms = -Xmx = 3g)

Tomcat Threads: maxThreads=500, minSpareThreads=25, maxSpareThreads=75

Portal Database Connection Pool Size: 100 (both minimum and maximum)

VHPD Cache Size: unlimited (both Level 1 and Level 2)

VHPD Default TTL: -1 (unlimited) (used for .gif, .css, .js files)

Apache 2.2 Non-default Configuration Parameters:

ThreadsPerChild 250

MaxRequestsPerChild 50000

Win32DisableAcceptEx

KeepAlive On

MaxKeepAliveRequests 0

## 3. Results

### 3.1 Test Methodology

In each of our tests, user requests were directed to a single Portal site. For the majority of the tests, the same site (SiteA) is used, where the baseline, single-user response time for a page that is generated dynamically is about 0.25 seconds.

In each of the following sections, we vary one set of related parameters, and provide metrics such as throughput and response time that illustrate the impact of varying those parameters. Each section begins by briefly describing the objectives of the corresponding set of tests. Then, the tests and results are described, followed by analysis of the results. The section ends with a brief conclusion based on the results.

### 3.2 Varying the percentage of fully-cached pages (SiteA, Plugin mode)

#### Objective

The objective here is to see how the system performs for the simplest parameter variation, when different numbers of pages of the test site are either fully cached or fully dynamically generated. Thus, portlets within a given page are either all cached (as part of the cached page), or all dynamically generated.

#### Test Description and Results

In these tests, the number of pages that were fully cached was varied. First, for a baseline measurement against which the other measurements were compared, the Portal site was set to be uncached (this is the default setting for Portal sites), so that each page was generated dynamically. Then, the Portal site was configured to be cached via VHPD. The percentage of pages within the site for which the HPD caching was turned off (i.e., the site cache setting was overridden so that the corresponding page was not cached) was varied from zero to 10, 20 and 50. Thus, the percentage of fully-cached pages varied from 100% to 90%, 80% and 50%.

Figure 2 shows the throughput measured as the number of zero-think-time virtual users (which will be referred from here on as ZTT users) was increased from 1 to 64 for the different percentages of fully-cached pages, and Figure 3 shows the corresponding average page response times. The upper limit of 64 ZTT users was chosen because we found that throughput had typically leveled off by the time this user level was reached on our hardware.

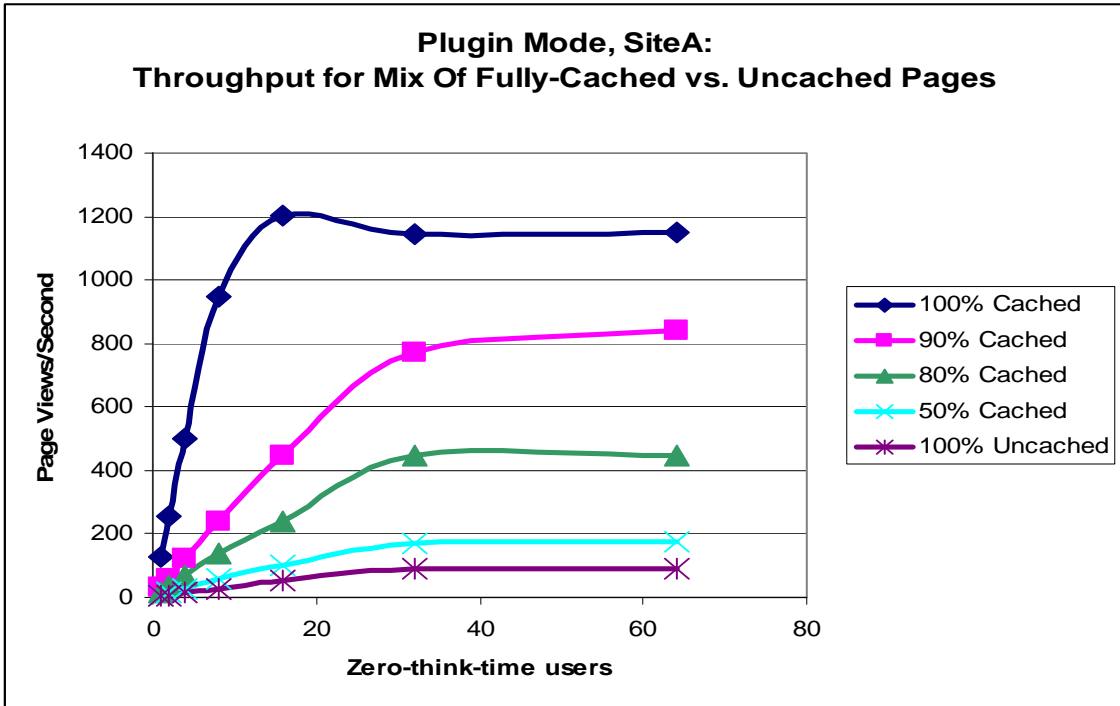


Figure 2

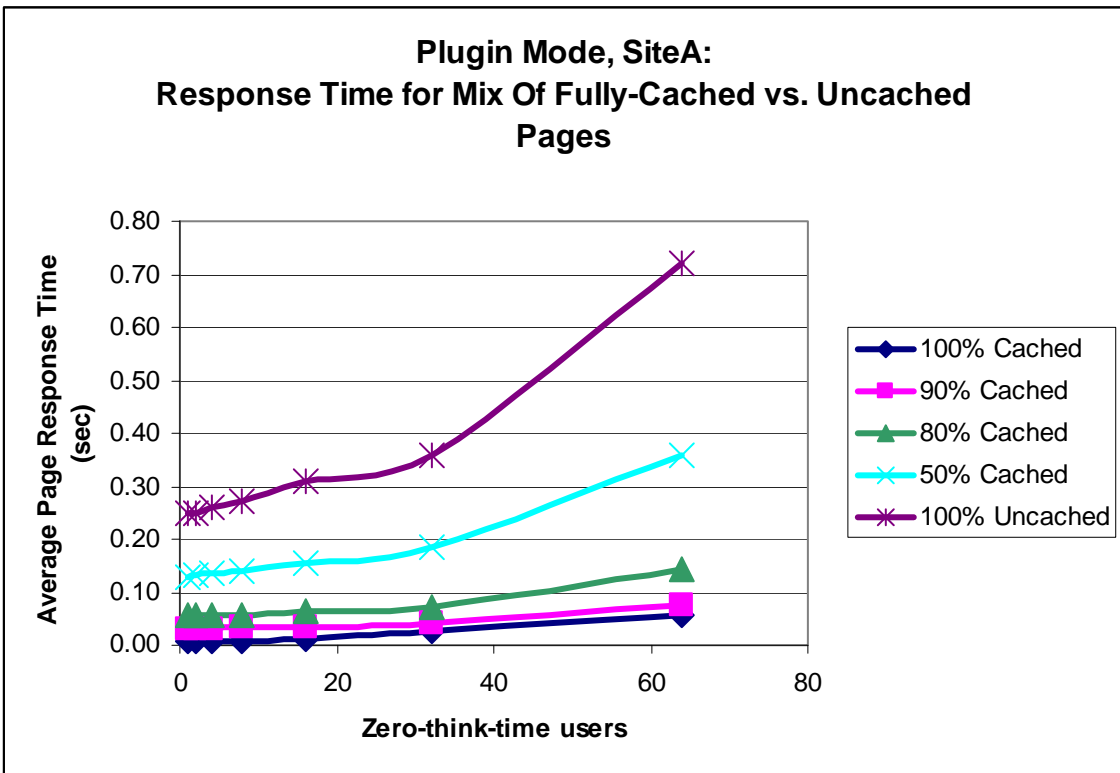


Figure 3

In order to summarize the results for these different scenarios, we picked a particular load level (corresponding to 32 ZTT users) representing a reasonably heavy load, and determined the ratio of the measured throughputs obtained for that load for the different mixes of cached pages, to the throughput for that load level in the 100% uncached baseline case. Figure 4 shows this throughput ratio (in effect, the improvement factor for throughput provided by VHPD) for 90% fully-cached, 80% full-cached, and 50% fully-cached pages. For throughput ratios, a higher value represents a better result than a lower value: for example, with 90% of the pages fully cached, the throughput was 8.7X (870%) that of the uncached scenario, while with just 50% of the pages cached, the throughput was 1.9X (190%) that of the uncached scenario.

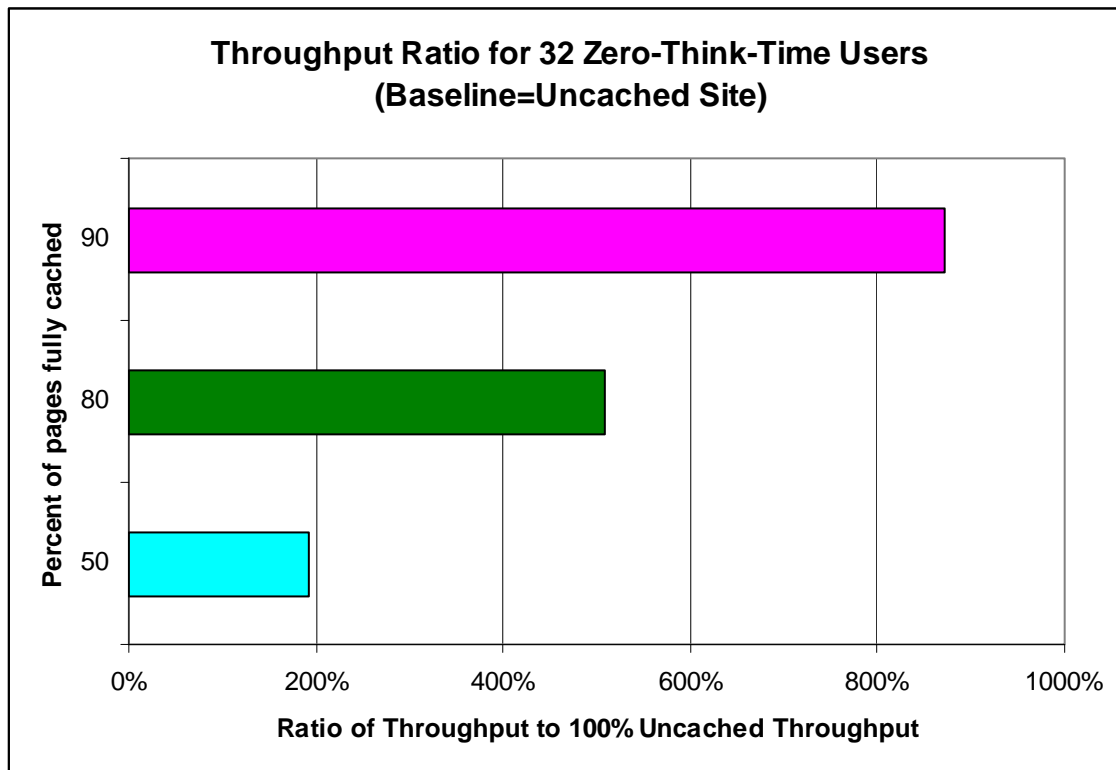


Figure 4

In Figure 5, we present the response time ratios for 32 ZTT users for the 90%, 80% and 50% cached scenarios: that is, the ratio of the response time measured for 32 ZTT users in each case to the response time measured in the baseline, uncached case. Here, a lower ratio indicates better performance.

### Analysis

As part of our tests, we also obtained measurements of CPU utilization at each of the servers used (including the load generators) and the network bandwidth in use between the load generators and the web server. This enabled us to identify the bottleneck resource in each of the test cases.

Our data shows that the bottleneck resource in the 100% uncached baseline case is the processor cores at the application server where the Portal was run. In contrast, the bottleneck in the 100% cached case was the 1Gbit/sec network bandwidth limit (although the web server CPUs were also very heavily used, and were also close to being the bottleneck resource). In the intermediate cases, where between 90% and 50% of the pages were fully cached, the processor cores at the application server were once again the bottleneck resource. This is because the cost of dynamically generating a page is so much greater than the cost of serving a cached page that the page generator (Portal) becomes the

bottleneck resource even for relatively highly-cached sites, as long as some of the requested pages are generated dynamically.

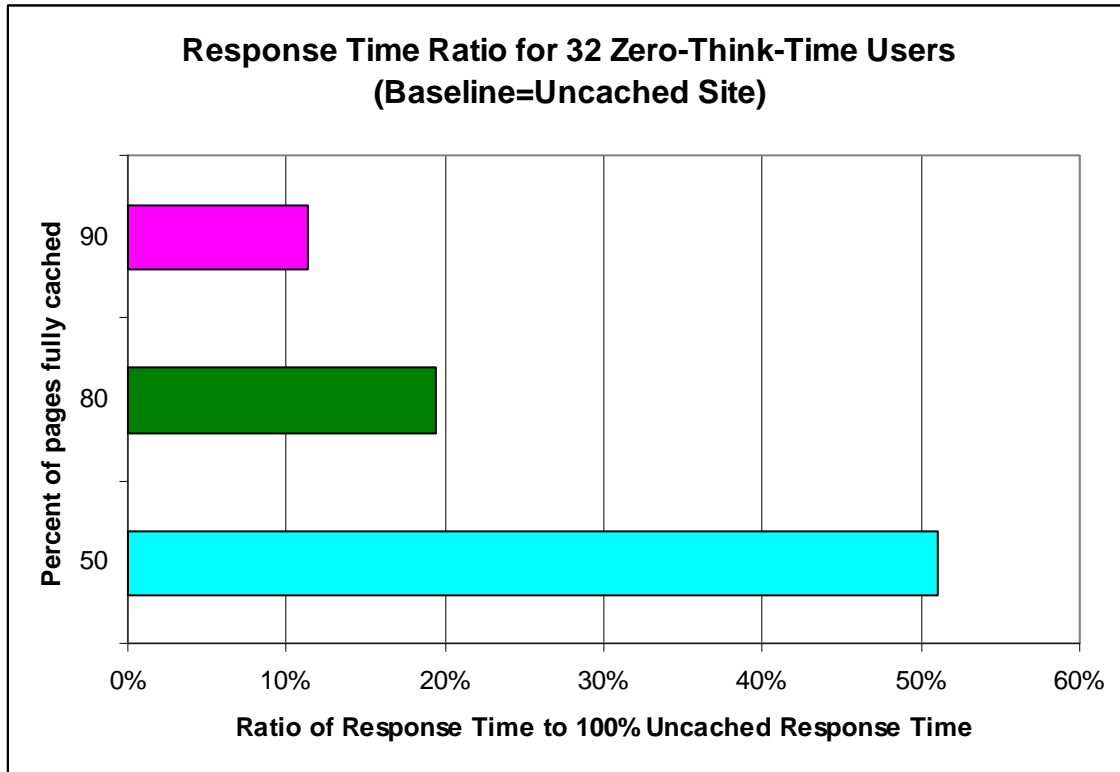


Figure 5

It is important to note that the relationship between throughput and percentage of pages cached is not linear: for example, the throughput for 32 ZTT users with 90% of the pages cached is about 67% of the throughput when everything is cached, not 90% as would be the case if the relationship were linear. Response time relationships are more linear: with 90% pages cached, the response time is close to 10% of the baseline response times; with 80% pages cached, the response time is close to 20% of the baseline response times, and so on.

### Conclusion

The benefits of VHPD caching are greatest when the proportion of cached pages is quite high. If the proportion of pages (or at least, the most heavily accessed or “hot” pages) that is cached is low, the performance improvements provided by VHPD may not be quite as dramatic.

### 3.3 Separately cached portlets using SSI (SiteA, Plugin mode)

#### Objective

The goal here is to see how the system performs when a subset of each page’s content is cached separately from the rest of the page, using sever side includes (SSI). A configuration like this may be used, for example, if there are a few cacheable components or portlets common to a large number of cacheable pages, and where these shared components may need to be updated from time to time, while the rest of the enclosing pages do not need to be updated as frequently. Note that (as in all the tests for which results are provided in this report) **no invalidations were performed during the tests.**

## Test Description and Results

In this set of tests, all the pages are configured to be cached, and some (1, 2, or 5) portlets within each page are cached separately from the rest of the page. Each page still contains a total of 10 portlets. Figure 6 shows the throughput for this set of parameters, and Figure 7 shows the response times. The baseline case (no pages cached) is also shown in each graph.

The throughput ratios for 32 ZTT users are shown in Figure 8, and the response time ratios are shown in Figure 9.

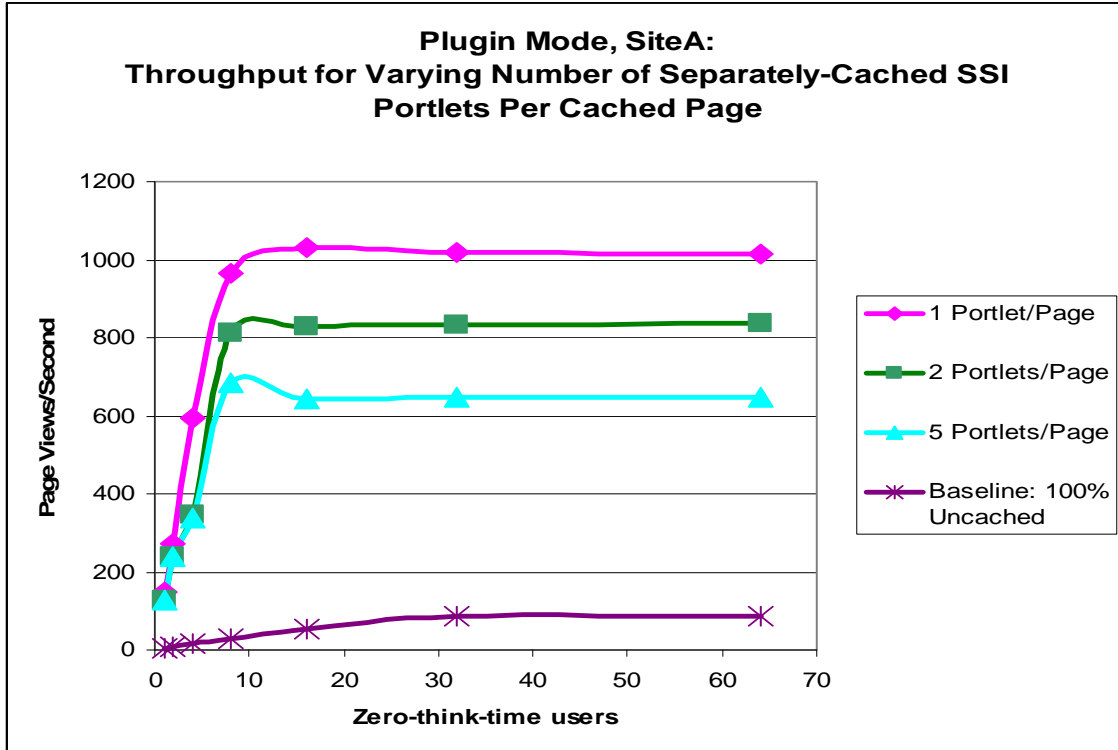


Figure 6

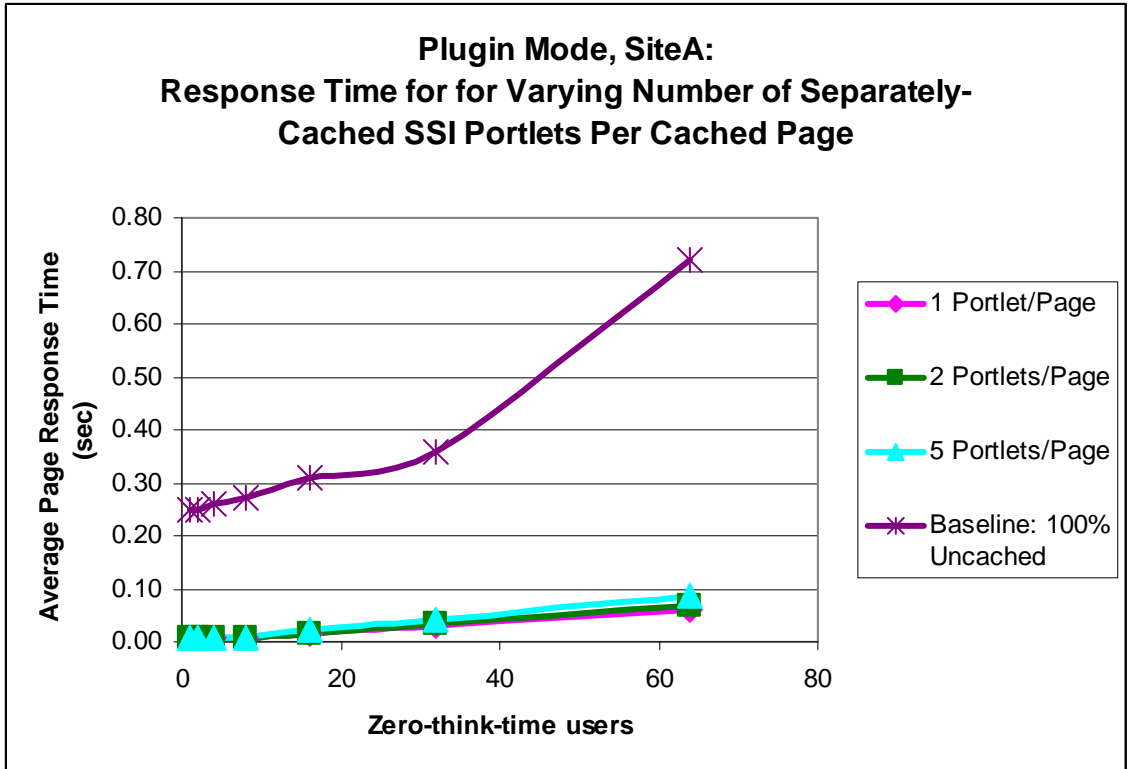


Figure 7

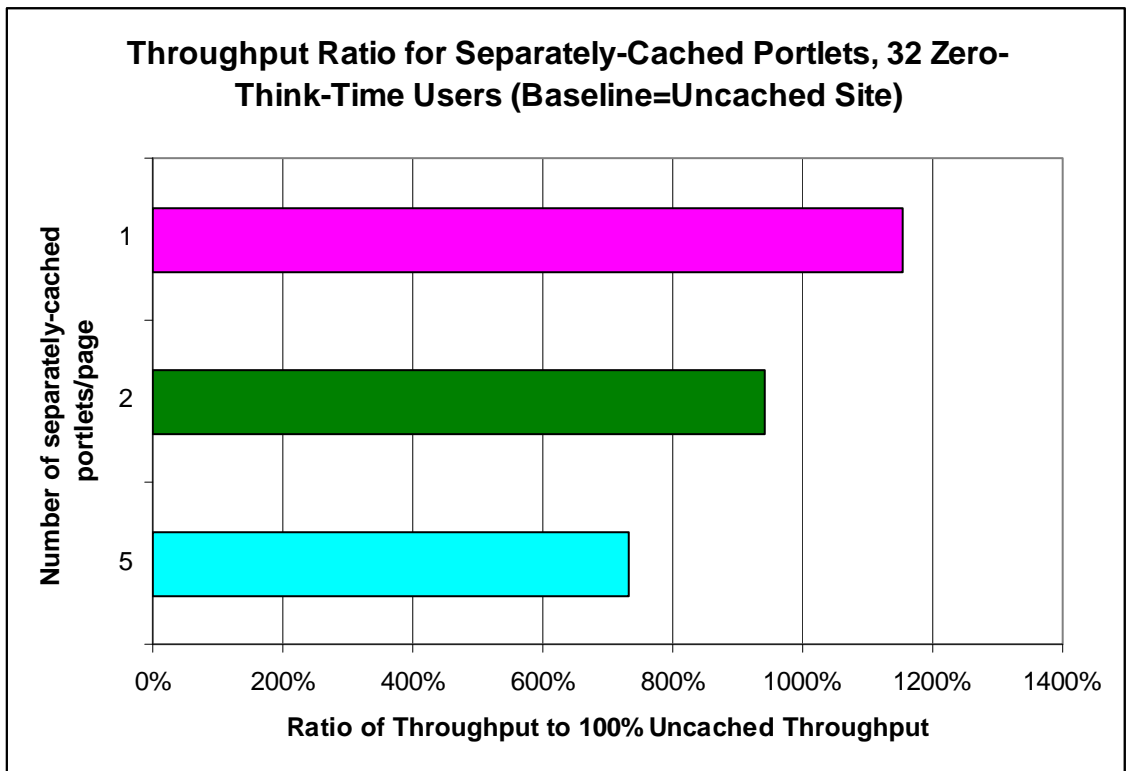
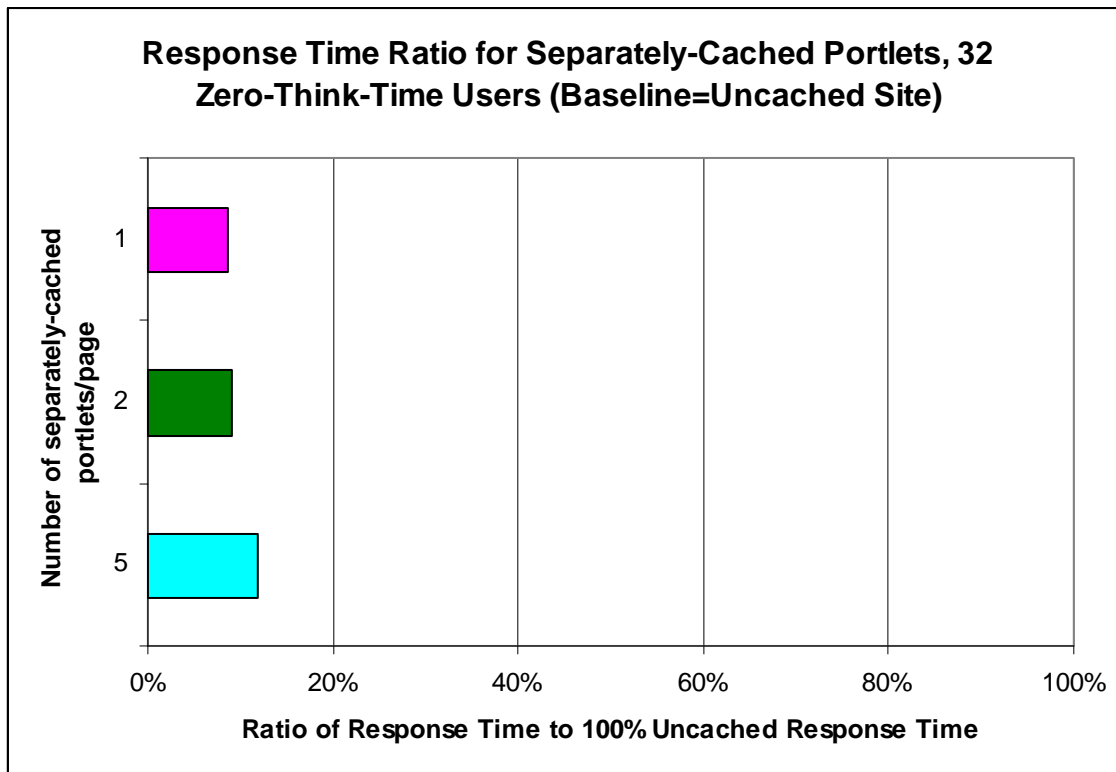


Figure 8



**Figure 9**

**Analysis**

The bottleneck resource for the tests with separately-cacheable SSI portlets is the processing power at the web server. Almost all the work is done at the web server, and the application server is largely idle. As the number of separately-cached portlets per page increases, the throughput is reduced somewhat, and the response time increase slightly, as the amount of work in assembling each page from its constituent components rises. Overall, though, as one would expect in a highly cached scenario, the performance is very good compared to the baseline uncached case.

**Conclusion**

VHPD can support very high performance for situations where subsets of cacheable pages are separately cached. As the number of separately cached portlets on a page is increased, performance falls off somewhat. If many portlets are cached separately, increasing the web server’s processing power could help increase performance further.

**3.4 Dynamically-generated portlets using SSI (SiteA, Plugin mode)**

**Objective**

The objective here is to determine the impact of including one or more dynamically-generated portlets (using SSI) on otherwise cached pages.

**Test Description and Results**

In this set of tests, all pages are configured to be cached, but some (1, 2 or 5) of the ten portlets within each page are dynamically generated using SSI. Figures 10 and 11 show the throughput and

response times, and Figures 12 and 13 show the throughput ratios and response time ratios for 32 ZTT users, respectively.

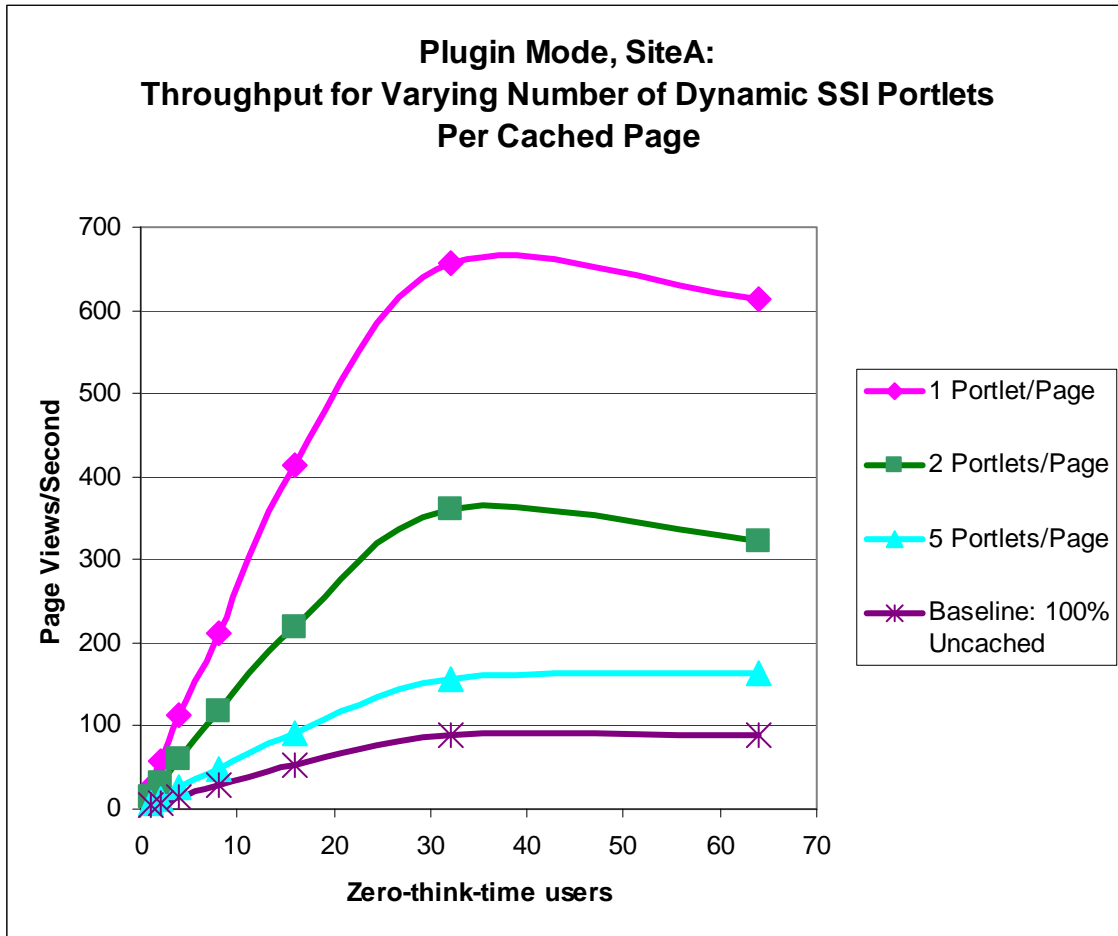


Figure 10

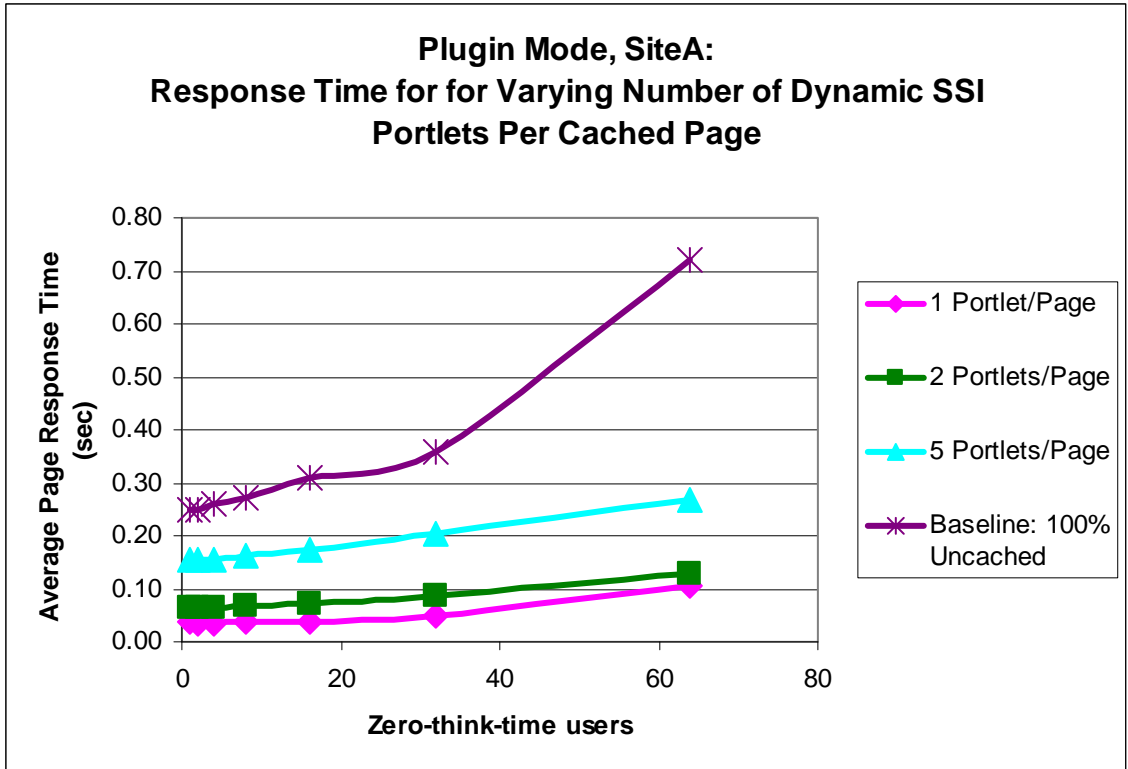


Figure 11

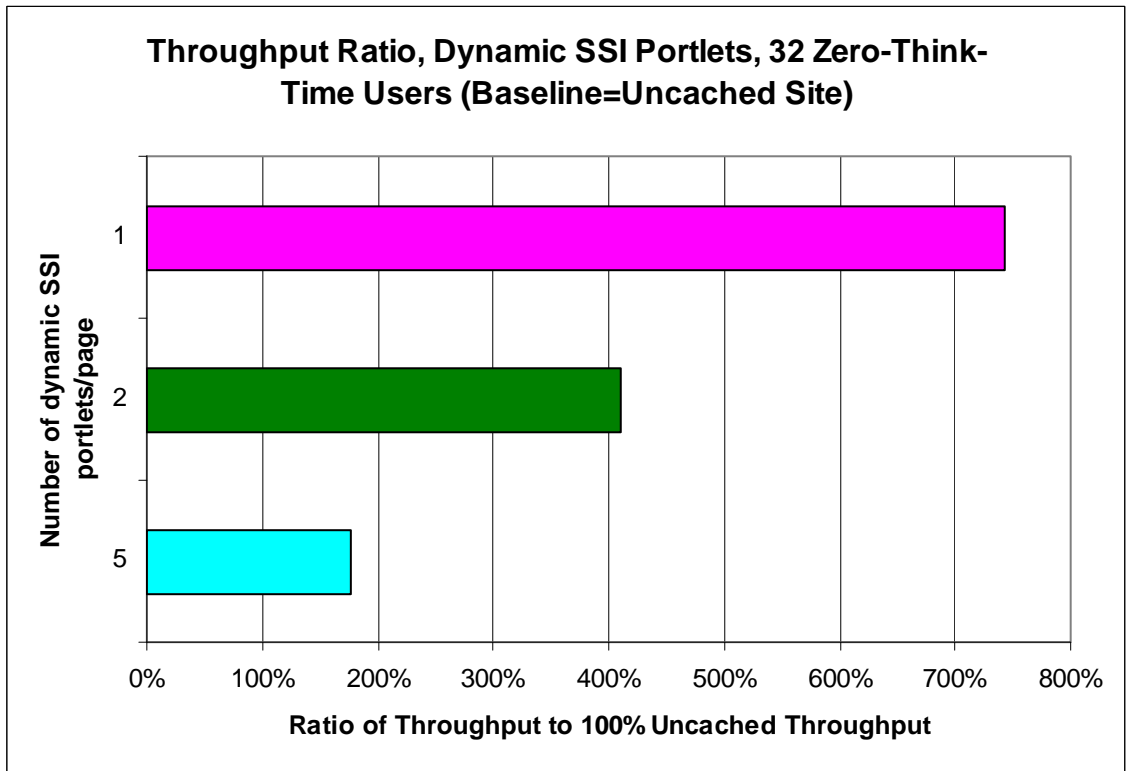
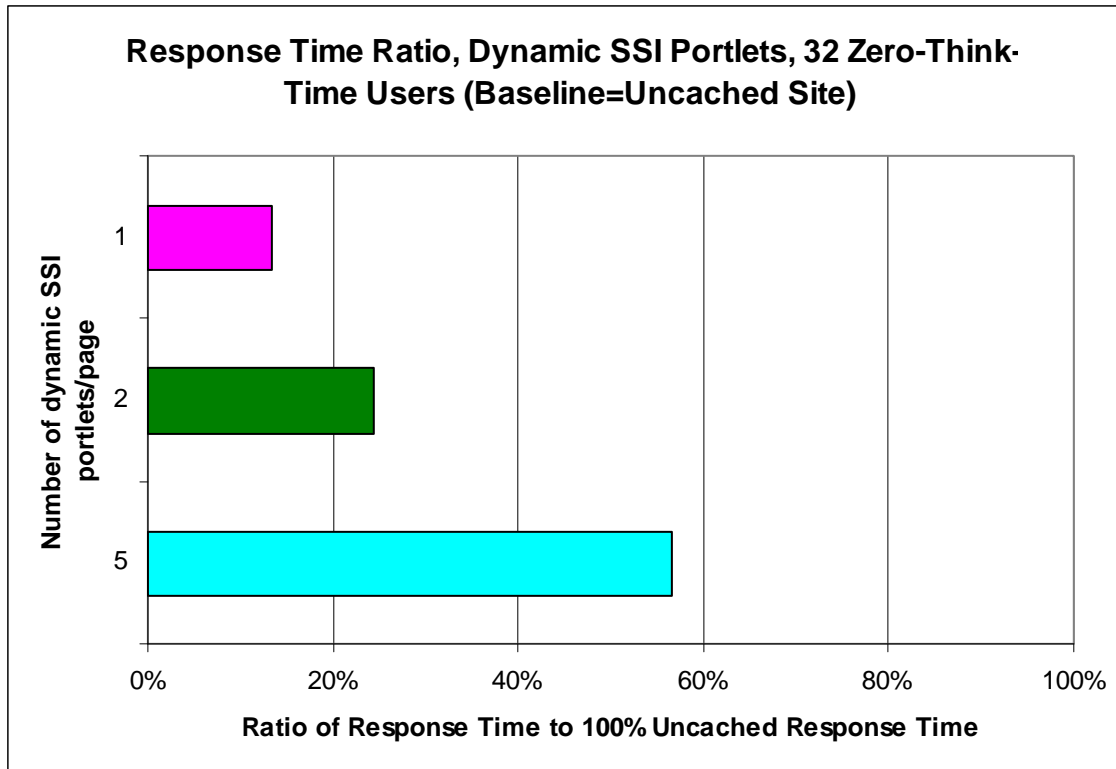


Figure 12



**Figure 13**

**Analysis**

The processor cores at the application server are the bottleneck resource for tests involving dynamic SSI portlets. Even though the enclosing page is cached, the amount of processing required to generate the dynamic portlets is much higher than the cost of returning the cached portion of the page, thus making the application server’s processing capability the bottleneck resource.

**Conclusion**

For small numbers of dynamic SSI portlets per page, VHPD provides excellent performance. If the number of dynamic portlets per page increases to more than a few, however, the performance improvements are not as impressive. Therefore, the number of dynamic portlets per page should ideally be kept as low as possible.

**3.5 Dynamically-generated AJAX portlets (SiteA, Plugin mode)**

**Objective**

The goal for these tests is determine the impact of including one or more dynamically-generated portlets configured to be rendered via AJAX on otherwise cached pages. The browser (being emulated by Loadrunner in our tests) issues separate requests for the page and then for each of the portlets configured to use AJAX. The page skeleton (i.e., the cached portion of the page, with “holes” where the content of the AJAX portlets would be) is returned to the browser first, and then the “holes” are filled when the browser requests the portlet content. One of the advantages of AJAX approach is that even if it takes a relatively long time to fill all the “holes” in the page, returning the page skeleton quickly makes the page response time appear more acceptable. In our results, therefore, the reported response time is the time taken for returning the page skeleton, and does not include the time taken to fill in the “holes”.

## Test Description and Results

All pages are configured to be cached, but some (1, 2 or 5) of the ten portlets within each page are dynamically rendered using AJAX. Figures 14 and 15 show the throughput and page skeleton response times, and Figures 16 and 17 show the throughput ratios and response time ratios for 32 ZTT users, respectively.

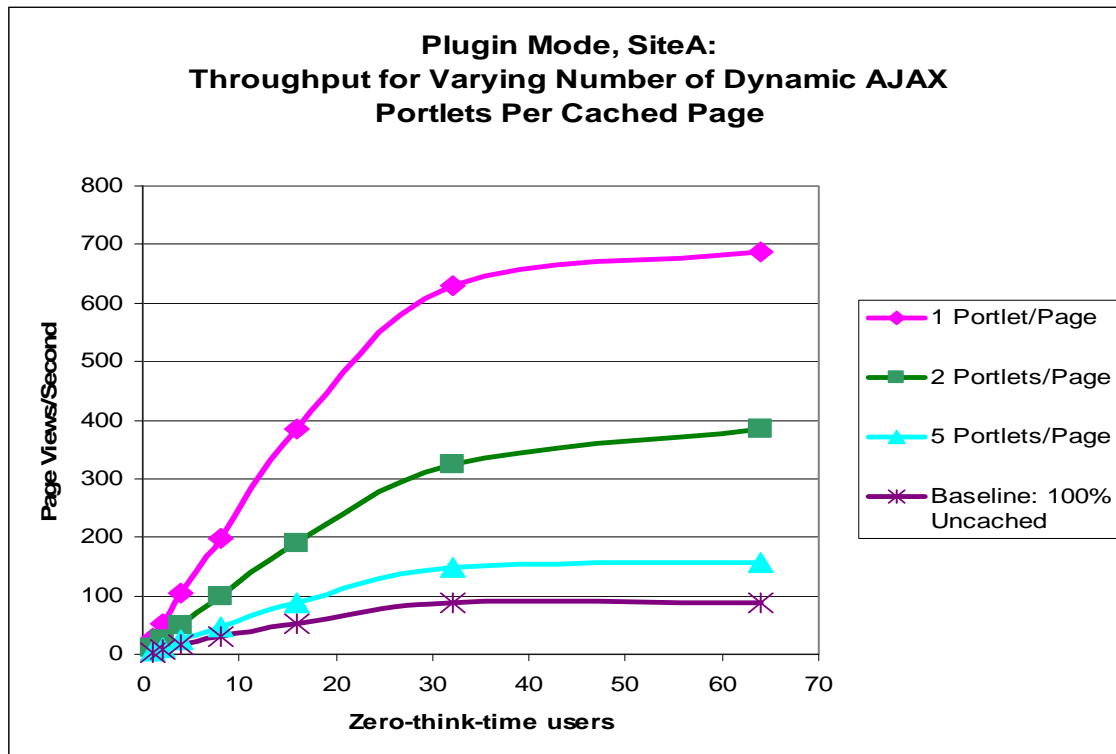


Figure 14

## Analysis

The processor cores at the application server are once again the bottleneck resource for tests involving AJAX. Since the page skeletons are cached, the apparent response time (and therefore the response time ratio) is very low. The throughput still flattens out because the application server has to do a lot of work to dynamically generate the content for the AJAX portlets. The overall performance is slightly worse than that for dynamic SSI portlets (described in a previous section), probably because each of the requests for the AJAX portlets has to go through the entire request path – from the browser through the web server to the application server and back.

## Conclusion

The conclusions for throughput are similar to the conclusions for dynamic SSI portlets tests: for small numbers of dynamic AJAX portlets per page, VHPD again provides excellent performance. If the number of dynamic portlets per page increases to more than a few, however, the performance improvements are not as impressive. Therefore, the number of dynamic portlets per page should ideally be kept as low as possible. Page skeleton response times are very low with AJAX, which may help to increase user acceptance of system responsiveness even though it may take a while to render the complete page.

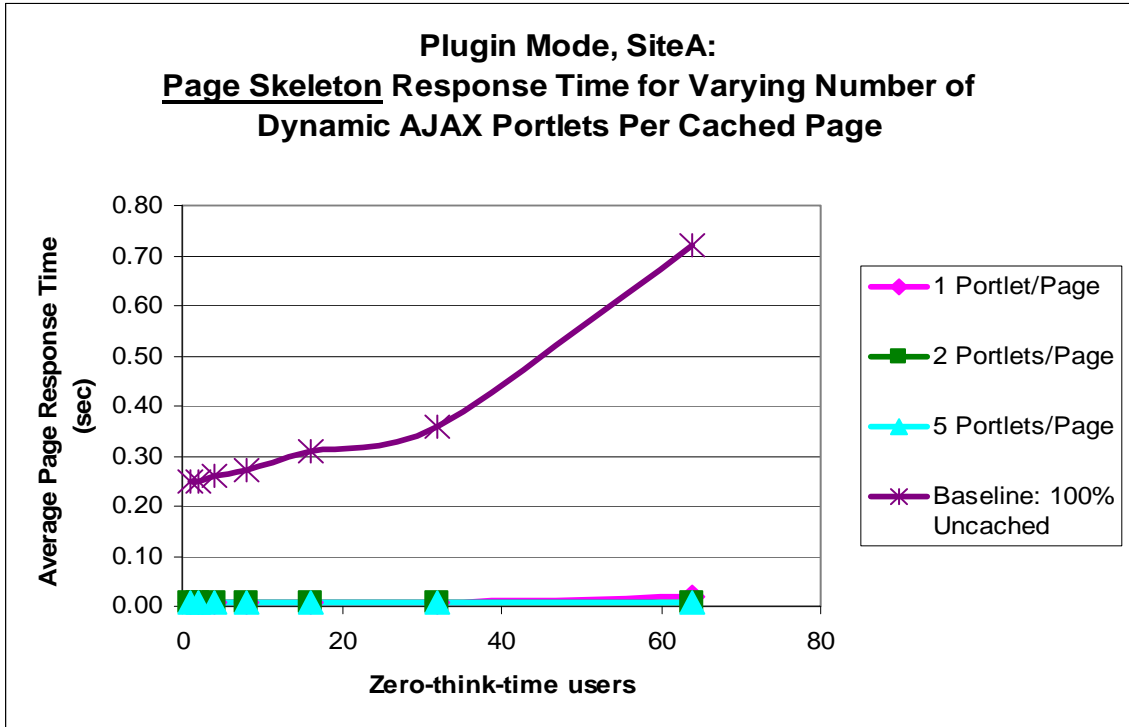


Figure 15

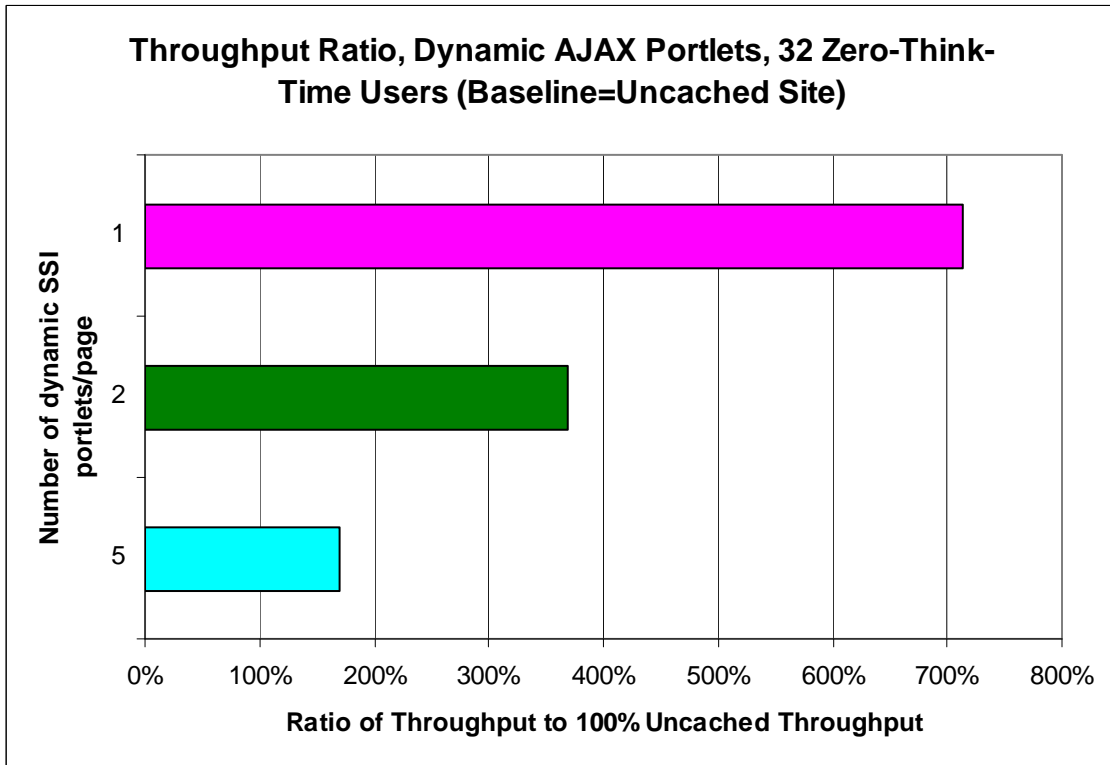


Figure 16

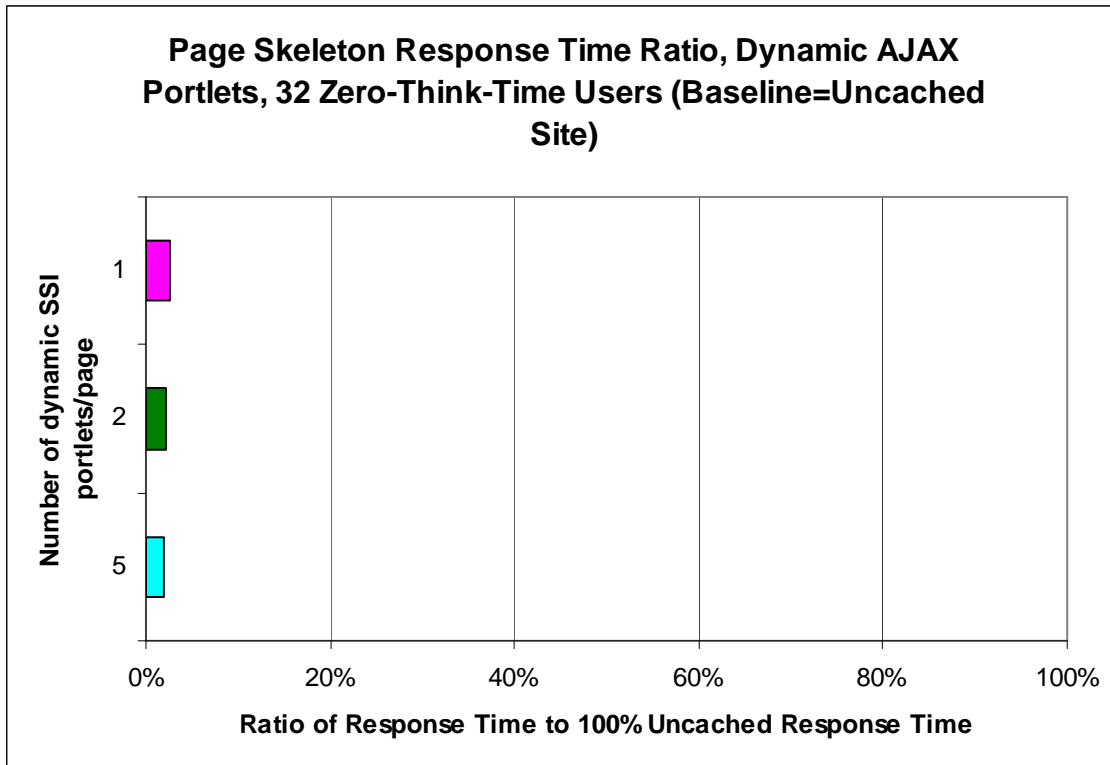


Figure 17

### 3.6 Sending requests to the application server directly (SiteA, Plugin mode vs. Servlet Filter mode)

#### Objective

In the tests described thus far in this report, user requests are sent to a web server configured with a VHPD plugin. The VHPD cache is maintained at the web server with the help of a remote cache manager, while other components of VHPD such as the dependency manager run at the page generator (Portal in our case) application server. However, VHPD also supports configurations where the cache is maintained at the same JVM as the page generator using servlet filters, requests are handled directly by the page generator and no plugin or web server is required. In this section, we compare the performance of the plugin mode of operation to the servlet filter mode for our hardware configuration. Note that the results in this section are especially sensitive to the relative computing capabilities of the web server and the application server. In our case, for example, the web server was a 4-CPU 3GHz Xeon machine running Windows with 3.8GB of memory, while the application server had 32 threads of hardware execution (in effect, providing 32-way hardware parallelism using 8 cores running at 1.2 GHz) and 32 GB memory, running Solaris 10. If the relative power of the web server and the application server were different, the results described here might change significantly.

#### Test Description and Results

The same tests as those described in Section 3.2 were run for both the Plugin and Servlet Filter modes of operation, i.e., a mix of fully-cached and fully-dynamic pages were requested in each mode. The throughput results are provided in Figure 18, and the response time results are in Figure 19.

### Throughput Comparison for Mix Of Fully-Cached and Uncached Pages, Plugin vs. Servlet Filter Mode

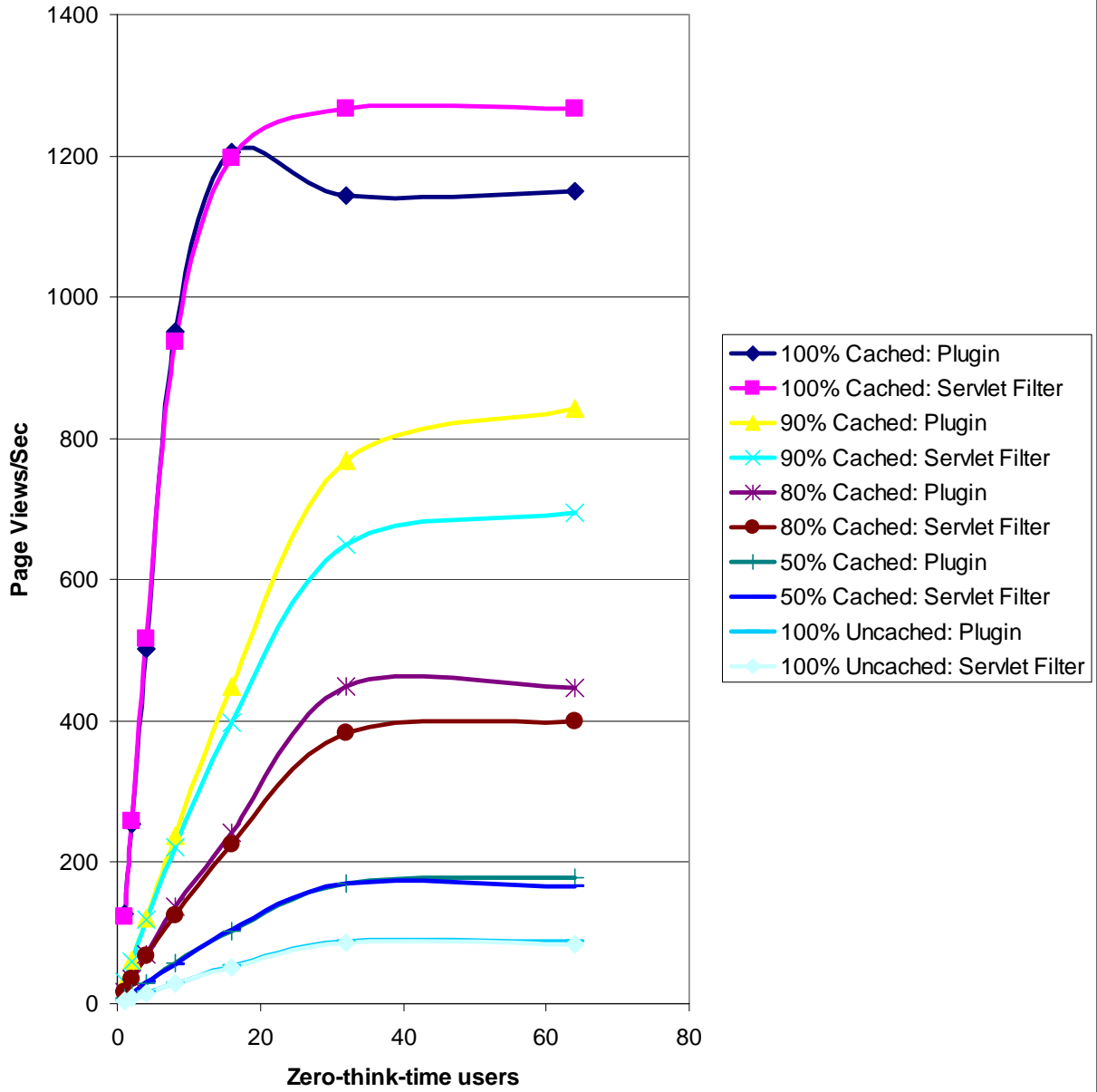


Figure 18

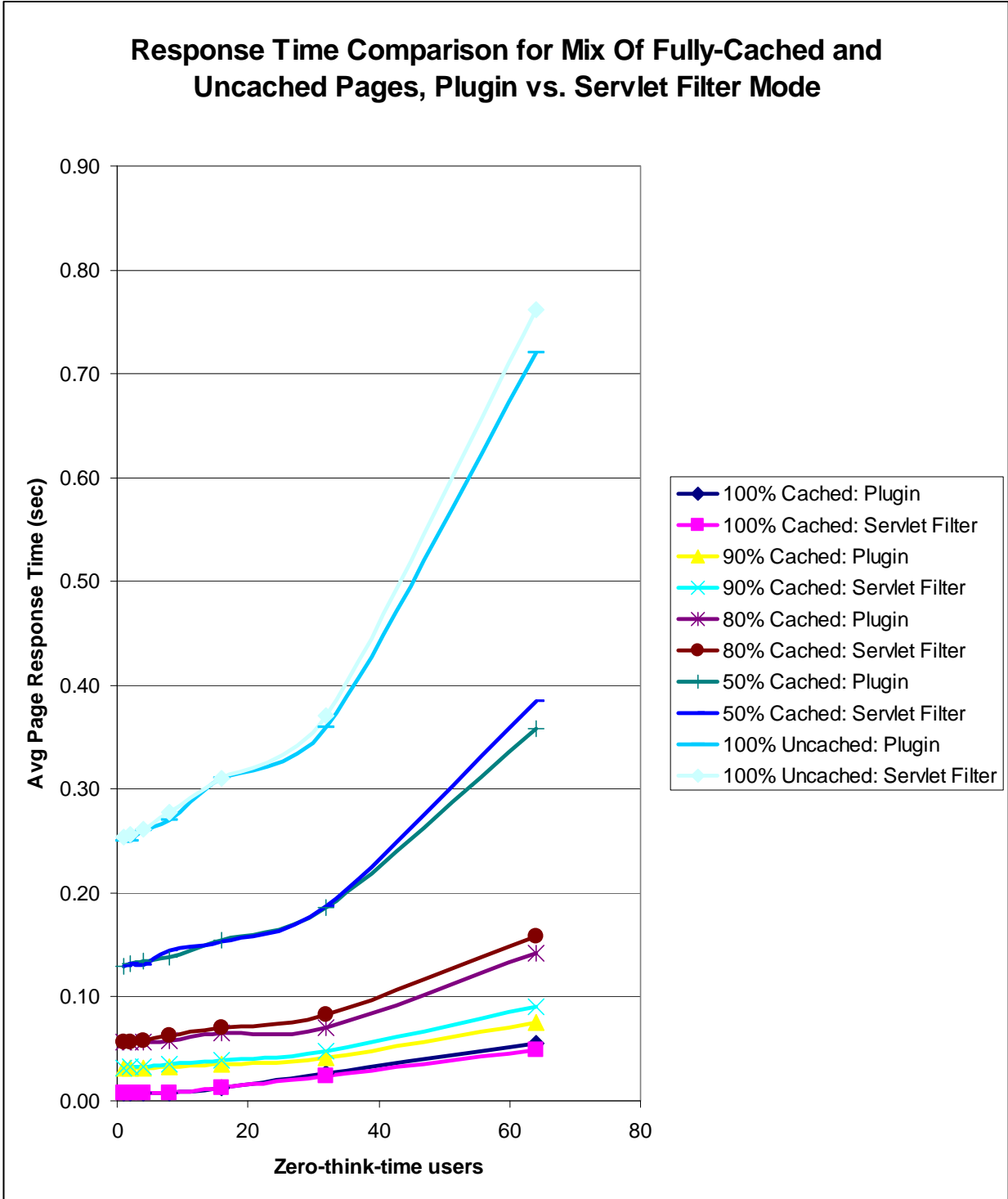


Figure 19

**Analysis**

The best way to explain the results comparing the Plugin mode to the Servlet filter mode is to begin by re-emphasizing the following basic facts: (1) the cost of generating a page dynamically is much higher than the cost of returning a cached page to the requester; (2) in the Servlet Filter mode, all the

work required for serving both cached and dynamic pages is done at the application server; and (3) our application server platform (like that in typical customer environments) is relatively more powerful than our web server platform.

Given these facts, we begin by analyzing with the 100% uncached results. There is virtually no difference in the performance for the Servlet Filter mode and the Plugin mode. This is because in either case, almost all the work is done at the page generator anyway; adding a web server to the picture doesn't offload much processing.

Next, we examine the 100% cached results. Here, somewhat surprisingly, the peak throughput is about 10% higher without the web server than with the web server. The reason is that, while in both modes the 1Gbit network bandwidth is close to being saturated at peak cached throughput, the CPU utilizations and parallelism capabilities are very different. In the Plugin mode, we have a 4-CPU server returning the requested pages, and the CPU utilization at the web server is 80% or higher. In the Servlet Filter mode, we have a 32-threaded parallel server returning the pages, and the CPU utilization is around 40% at the peak throughput. Thus, especially because no dynamic page generation is being requested, our application server is able to handle the incoming requests much more comfortably (and with less queuing and probably less operating system overhead) than the web server. If we had used a weaker web server, the difference in performance would have been even greater for 100% cached pages. If we had used a (much) weaker application server, the relative performance of the Plugin mode could have been much higher, and it is easy to imagine configurations (e.g., where an application server less powerful than the web server is used) where the relative performance of the two modes for 100% cached pages is reversed.

Finally, we examine the results for the intermediate cases, where there is a mix of fully cached and uncached pages. Here, especially for the cases where the percentage of cached pages is higher (such as 90% cached pages), the peak throughput for the Servlet Filter mode is somewhat (around 10-15%) less than the throughput for the Plugin mode. This is because, at high loads, the amount of dynamic page generation being done is sufficient to use almost all of the processing power at the application server, and a substantial amount of additional work is still required for handling the large volume of requests for cached pages. Therefore, splitting the work between the application server and the web server as in the Plugin mode proves to be an advantage. For cases where the percentage of cached pages is lower (e.g., 50% fully cached), there is so much dynamic page generation work that, as in the 100% uncached case, there is relatively little advantage to adding the web server to the picture.

Response time results in Figure 19 also indicate similar slight differences between the two modes of operation for our hardware and our test case.

## **Conclusion**

In general, at high loads and with high proportions of cached pages together with at least some dynamic page generation, the performance provided in the Plugin mode was found slightly superior to the performance of the Servlet Filter mode, due to the ability to split the work between two servers in the Plugin mode. Overall the differences were not large, and, as noted before, are probably very sensitive to the relative computing capabilities of the web server layer and the application server layer.

## **3.7 Varying the processing capability of the application server**

### **Objective**

In this section, our goal is to vary the processing capability (which translates to the number of hardware threads of execution configured) at the application server and see how this affects the achievable performance for a mix of fully-cached and uncached pages. This can potentially help answer questions such as: if 80% of my pages can be cached, can I get the same performance as I did prior to installing HPD, while using less hardware? To keep the analysis simple, results for only the Servlet Filter mode are provided.

## Test Description and Results

The same tests were used as those in Section 3.2, where the number of fully-cached vs. uncached pages was varied. The application server was initially configured with its default 32 hardware threads of execution (corresponding to four threads each at eight cores – please see Sun T2000 documentation for their explanation of these terms). The Solaris “psradm” command was used to throttle down the number of threads to 16 and 8, and the tests were repeated for each setting for the number of threads. Figure 20 shows the throughput for various combinations of threads and cached pages. Since the graph is somewhat crowded, especially towards the low-throughput region, the underlying page views per second data is also provided in Table 4. To try to avoid even more data overload, response time results are not provided in this section.

## Analysis

A good way to look at these results is to ask the question: suppose my baseline environment uses 32 threads, and I do not have caching enabled. If I could cache X% of my pages using VHPD, how many threads would I need to achieve about the same peak throughput as I got without VHPD caching?

Looking at Table 4, the peak throughput with no caching (100% uncached) and 32 threads is about 86 page views/second for 32 zero-think-time users. If 80% of the pages could be cached, then with 8 threads (i.e., one-fourth the hardware), we got about 100 page views/second (see the row labeled “8 threads, 80% cached”). If only 50% of my pages were cached, then I could still get around the same peak throughput with 16 threads (see the row labeled “16 threads, 50% cached”).

(Note: The fact that the peak throughput for 100% cached pages is very similar for 32 threads and 16 threads may initially be surprising. However, this is because the network bandwidth is the bottleneck for 100% cached pages in both cases – the application server CPU utilization is only around 40% with 32 threads and around 80% with 16 threads. When the number of threads is reduced to 8, the CPU utilization becomes the bottleneck and the peak throughput for the 100% cached case is lower.)

## Conclusion

The results indicate that substantial reductions in hardware may be possible if VHPD is deployed. Of course, the results may vary with the workload.

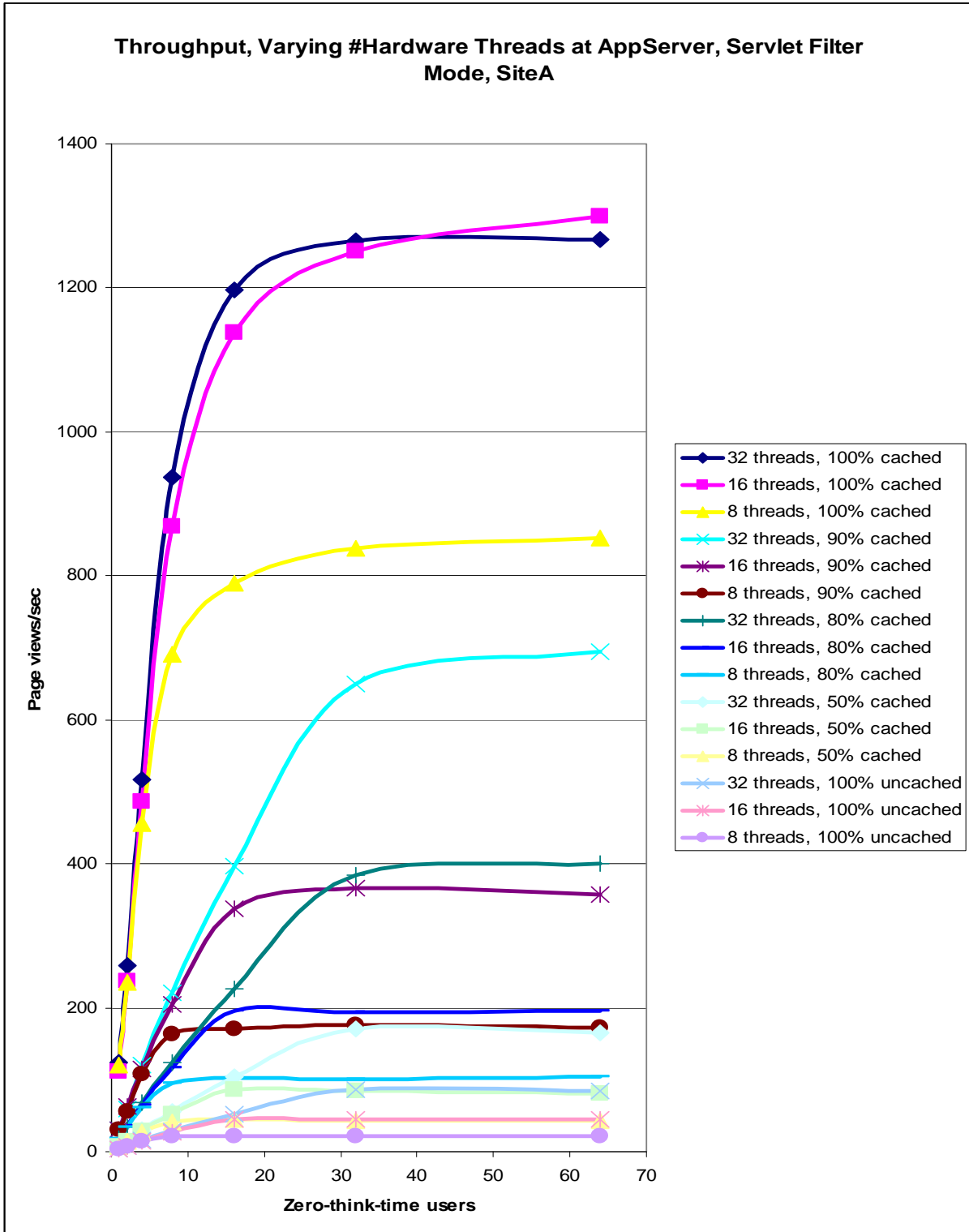


Figure 20

Servlet Filter Mode Throughput Results For Varying # HW Threads, Page Views/Sec							
Number of Zero-Think-Time Users →	1	2	4	8	16	32	64
32 threads, 100% cached	124	259	516	938	1197	1266	1267
16 threads, 100% cached	111	237	487	868	1137	1251	1299
8 threads, 100% cached	120	236	456	691	790	838	853
32 threads, 90% cached	30	60	120	221	397	649	694
16 threads, 90% cached	30	62	115	205	337	366	358
8 threads, 90% cached	31	56	108	164	170	175	173
32 threads, 80% cached	17	35	68	125	226	384	400
16 threads, 80% cached	17	36	64	116	196	194	196
8 threads, 80% cached	17	34	61	96	103	100	104
32 threads, 50% cached	8	15	30	55	104	170	166
16 threads, 50% cached	8	15	29	52	87	84	81
8 threads, 50% cached	7	15	27	42	45	43	44
32 threads, 100% uncached	4	8	15	29	51	86	84
16 threads, 100% uncached	4	8	15	27	45	45	44
8 threads, 100% uncached	4	8	14	22	22	22	21

Table 4

### 3.8 Varying page complexity

#### Objective

In all the tests described thus far, the same Portal site (SiteA) has been used, where the nominal single-user response time per uncached page is about 0.25 seconds on our hardware, much of it spent doing random number computations within each of the JSR168 portlets making up the page. (This nominal response time doesn't change much, whether the Plugin mode or Servlet Filter mode is used, since almost all the time is spent in the application server for uncached pages; the web server doesn't add to the response time much.) In this section, we examine how much the performance improvements provided by VHPD vary if the page complexity is changed: e.g., if instead of 0.25 seconds, it takes one second, or two seconds, or 0.1 second to generate the page. We varied the time taken by increasing the number of random number operations per portlet and/or adding "Thread.sleep()" calls (delays) to the portlets.

#### Test Description and Results

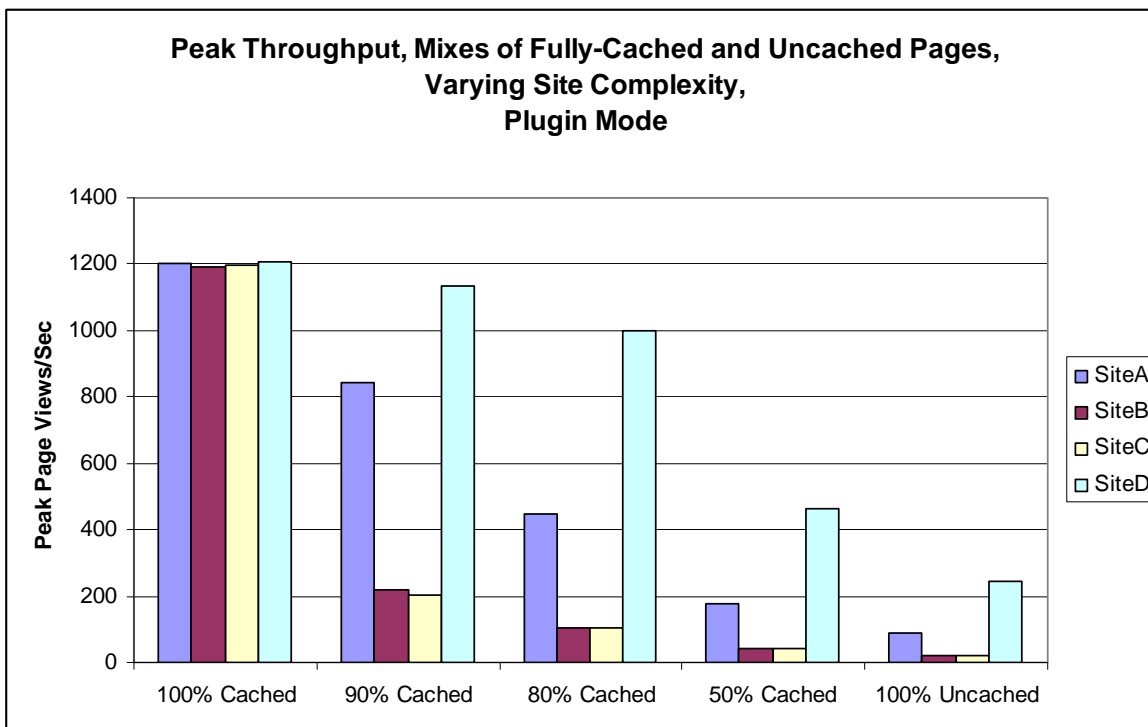
The number of fully-cached vs. uncached pages was varied, as in the tests described in Section 3.2. The site had 100 pages with 10 JSR168 portlets each, but the work done in each portlet was varied as shown in the Table 5. **Note that the total size of each page body remained the same across all sites**, so that the number of bytes sent to the requesting virtual user per requested page did not vary with the sites.

Site	Nominal Single-User Uncached Page Response Time (Using T2000 as Application Server)	Random numbers used for computations, per portlet	Sleep() call duration per portlet
SiteA	0.25 second	125,000	None
SiteB	1.0 second	600,000	None
SiteC	2.0 seconds	600,000	100 milliseconds

SiteD	0.1 second	22,000	None
-------	------------	--------	------

**Table 5**

The peak throughput measured for each site, for each percentage of fully-cached pages in the site, is shown in Figure 21. Note that the number of zero-think-time users for which the throughput peaked was not the same for all the sites and parameters (it peaked at 16, 32 or 64 users depending on the test); therefore, comparing response times at peak throughput is not meaningful in this case. Figure 22 shows the ratio of the peak throughput measured in each scenario, to the peak throughput for 100% uncached pages for that same site. Thus, Figure 22 shows examples of how much improvement in throughput VHPD can provide for different types of sites and pages.



**Figure 21**

**Analysis**

As expected, when there were uncached pages in the mix, the peak throughput varied from site to site. The processing power at the application server was usually the bottleneck resource when the mix included uncached pages. This helps to explain the fact that even though the sleep() call added to the nominal response time for SiteC compared to SiteB, the peak throughputs for these two sites were very similar, since the amount of processing per portlet was the same for both these sites.

For 100% cached pages, the network bandwidth was the bottleneck resource, with the web server CPU utilization close behind. All the peak throughput results for 100% cached pages were therefore very close to each other.

Figure 22 shows that the relative throughput improvement achieved as a result of deploying VHPD can vary quite a bit, depending on the expense of generating the pages dynamically. For relatively lightweight pages such as in SiteD, where even without any caching the peak throughput exceeds 200 page views per second, the relative improvement in throughput is not nearly as high as when the baseline uncached pages are more expensive (Sites A, B and C). Note that one of the main reasons why SiteD does not show as much improvement as the other sites is that our test network bandwidth is limited to 1Gb/sec; had a much faster network been available, the throughput improvement for

SiteD would have been closer to that for the other sites.

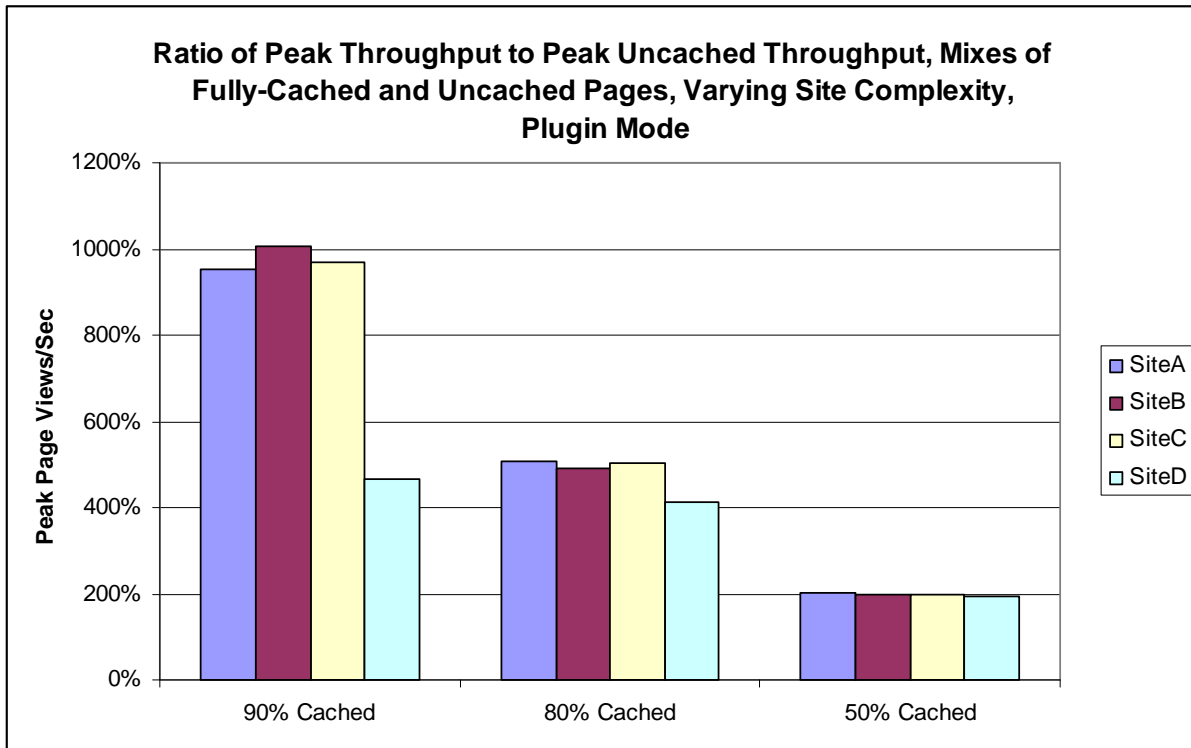


Figure 22

## Conclusion

The relative improvement that VHPD can provide will vary, based on the nature of the workload and the baseline uncached site. For lightweight pages (i.e., pages that do not take very long to generate dynamically), deploying VHPD may not result in as much benefit as for heavyweight pages.

## 4.0 Caveats

The results presented here should be interpreted with the following caveats in mind:

1. All the cached pages and portlets were served from the in-memory (Level 1) cache, and were never invalidated during the testing. This represents the best-case cache performance that VHPD can provide. In practice, a number of factors may reduce the performance, for example, (a) not all the data may fit in the in-memory cache, so some may have to be retrieved from the on-disk (Level 2) cache (b) updates may cause some (or even all) data in the cache to be invalidated from time to time, thus requiring pages to be regenerated and the cache to be repopulated.

2. Since we used zero think times in our tests, we were able to utilize much of the computing power available with a relatively small number of concurrent virtual users. For each concurrently-running virtual user, an object called a "session" is maintained by the system, which results in a certain minimum amount of memory consumption per concurrent user. Thus, as the number of users increases, the amount of memory needed within the Java Virtual Machine (JVM) running the Portal also increases. In real-world environments, the number of users interacting with a given Portal configuration is likely to be much higher than 64 (and these higher numbers of users can be supported because the think times between requests are typically several tens of seconds). Thus, more session

objects are typically likely to reside in the JVM at a given time than in our tests, which can lead to longer garbage collection operations and thus reduce overall performance somewhat.

3. For simplicity we configured the T2000 as a single Solaris zone, where all 8 cores (up to 32 threads of parallel execution) and all 32 GB of memory were available for use by a single Java Virtual Machine (JVM) running Tomcat, Portal 7.4 and VHPD. This means that while the T2000 had 32 GB of memory available, only 4GB (the limit accessible by any 32-bit JVM) was available for use by the Portal and VHPD; the remaining 28 GB was unused during our tests. In practice, customers would probably partition the T2000 into four or eight Solaris zones, so that a different JVM could run in each zone as a node of a Portal cluster and thereby make better use of the 32GB memory. For example, if eight zones were used, each of eight JVMs could use 4GB of memory. Since our tests consisted of read-only operations and had relatively few concurrent users, and therefore session-related memory usage was not a huge factor (see caveat 2 above), the performance difference between using clustering with multiple zones in this fashion and our single-zone configuration may not be very large. However, without actually configuring and measuring the clustered, multi-zoned configuration, it is hard to predict the performance that would have been observed in such more realistic environments.

4. In most tests, our portlets were highly CPU-intensive. In customer environments, other factors such as database access latencies and/or network latencies to other web sites (for portlets fetching content from such remote web sites) may play important roles in determining system performance, not just the processing power. Our attempt to insert `sleep()` calls into the portlets for SiteC is intended to emulate such non-CPU-intensive operations. However, the mix of resource limitations in real-world environments may alter the performance trends we saw.

5. Lastly, it is very important to emphasize that the trends we have reported have only been measured on the specific set of hardware that we have used, and for the specific set of tests and sites described above. We have not tested VHPD with similar parameter variations on other hardware or software platforms, and it is quite possible that different trends would emerge on different platforms.