Content Server

Version: 6.3

Delivery Systems Sizing Guide

Document Revision Date: Jul. 25, 2006

FatWire SOFTWARE

Content Server 6.3 Delivery Systems Sizing Guide
Document Revision Date: Jul. 25, 2006
Product Version: 6.3

FatWire Technical Support
www.fatwire.com/Support

FatWire Headquarters
FatWire Corporation
330 Old Country Road
Suite 207
Mineola, NY 11501
www.fatwire.com

Table of

Contents

1 Abstract. 5
Scope 6
Key Findings 6
Base Conditions 7
Definitions 8
System Configurations 9
Hardware 10
Software 10

Part 1. Scaling Factors

2 Page Caching 13
Results 14
Scaling Charts 14
Scaling Factors 16
Methods 16

3 Blob Objects 19
Results 20
Scaling Charts 20
Scaling Factors 22
Methods 22

4 Application Servers 25
Results 26
Scaling Chart 26
Scaling Factors 27
Methods 27

Test Conditions 27

5 Horizontal Clustering 29
Results 30
Scaling Charts 30
Scaling Factors 31
Methods 31

Part 2. System Specifications

6 Page Design and Template Specifications 35
Page Design 36
Page Content 36
Page Rendering 36
Page Caching 37
Template Specifications 37
Template Call Structure 38
Template Descriptions 39
Wrapper Templates 39
Called Templates 40
Rendering Templates 40

7 Test Conditions 41
Setting Up 42
Pre-Loading 42
Ramp-Up 42
Monitoring 42
Operating System Monitoring 42
Application Server Monitoring 43
End-User Monitoring 43
Database Monitoring 43

8 Template Code 45
Page/ProdList1 46
Product_C/ProductList1 47
Product_C/ProductListNestedCached 49
Page/ProdList2 52
Product_C/ProductList2 53
Product_C/ProductListNestedUncached 55
Product_C/FSII Summary 57
Media_C/FSII Summary 62

Chapter 1

Abstract

The size that you determine for your system depends on many factors, such as the following, to name a few:

- Platform configuration
- Tuning parameters
- Page design
- Data model
- Page caching
- Complexity of the rendering templates
- Network connectivity of internal systems
- Internet connectivity

This guide provides examples of CS-powered delivery systems (both single node and horizontally clustered) to help you size your own environment according to the performance you require.

Note

This guide presents relative performance metrics. Absolute performance metrics and tuning parameters are available in a separate publication. Please contact your sales representative for a copy.

Scope

This document describes the results of sizing tests that FatWire completed on Content Server 6.3 delivery systems. We measured key performance indicators—hits per second and response times—and analyzed them to size Content Server 6.3 installations according to:

- Percent page caching
- Number of blob objects per page
- Type of application server
- Number of CS nodes in a horizontal cluster

We tested the systems that are diagrammed in "System Configurations," on page 9. We also created test pages. Before starting the sizing tests, we tuned the application server, JVM, database, operating system, and Content Server in order to optimize the performance of our delivery system. Page design and template specifications are given in Part 2, "System Specifications."

Key Findings

- **Percent caching**

  Performance improves significantly for CS delivery sites as users access more cached pages than uncached pages. For instance, a single-CS system delivering pages at a 1-second response time supports greater loads as percent caching increases, as shown in the table below.

Percent Cached Pages	Supported Load	Percent Improvement
50	1.9X users[a]	
75	2.3X users	21.0
100	3.5X users	52.2

a. X=100 across all tests in this guide.

Scaling charts are given in Chapter 2, "Page Caching" for the following caching scenarios and response times: 100%, 93.5%, 87.5%, 75%, 62.5%, 50%, 25%, and 0% caching at 1-, 3-, and 5-second response times.

- **Number of blob objects per page**

  Adding blob objects to a page degrades performance.

  Given a baseline page with 10 blob objects, adding blob objects to the page reduces the load that Content Server can support at 10 blob objects. Percent reduction is roughly the following at 1-second response time:

Blob Objects per Page	Supported Load
20	70% L_{max} [a]
30	50% L_{max}
40	40% L_{max}

a. L_{max} is the load supported by the system serving 10 blob objects per page in a given caching scenario.

  Scaling charts are given in Chapter 3, “Blob Objects” for different page caching scenarios at response times of 1, 3, and 5 seconds.

- **Supported application servers**

  Compared to WebLogic, Content Server running on Resin performs at least 40% better. On JBoss, Tomcat, and WebSphere, Content Server performs 15–25% better than on WebLogic.

  Scaling charts are given in Chapter 4, “Application Servers” for different page caching scenarios.

- **Number of CS nodes in a horizontal cluster**

  Adding CS nodes to a single-CS system improves the ability to handle hits per second.

  - Adding one CS node (2 nodes total) improves peak hits-per-second by a factor of 2.2 (on average).
  - Adding another CS node (3 nodes total) improves peak hits-per-second by a factor of 3.2 (on average, relative to the single-CS system).

  Scaling charts are given in Chapter 5, “Horizontal Clustering” for different page caching scenarios.

Base Conditions

We set several conditions for performance testing:

- We tested delivery from a development system, assuming it should not be different from a production system, as none of our scenarios involved publishing operations while delivery performance was being tested.
- We based our tests on the asset framework in FirstSiteII (specifically the “Product” asset family, and the “Media” asset family).
  - Using the “Product” and “Media” assets, we created a simple page design that used uncached wrapper pages to call nested cached pagelets. Our design made it simple for us to measure the extent to which page caching improves performance.

  - We also designed our own templates to make them specific to our intended measurements and to keep things simple and flexible during the testing process.

  For information about the pages and associated templates, see Chapter 6, "Page Design and Template Specifications."

- Before collecting performance data, we needed to ensure that our systems are stable and performance tests are reproducible. To this end, we optimized and tuned various parameters in the environment as well as in Content Server. Optimization and tuning helped us ensure that performance was affected only by the parameters that we changed and not by extraneous factors or unknown entities.

Definitions

Load	Number of virtual users at any given condition (percent page caching, number of blob objects per page, and so on).
L_{max}	Maximum load that can be supported. L_{max} is defined on a case-by-case basis.

System Configurations

We tested Content Server's scalability in several configurations using the software and hardware listed on page 10.

Figure 1: Single Content Server

This configuration was used to test Content Server's response time, ability to handle hits per second, and ability to support user load in various tests based on page caching. For test results, see:

- Chapter 2, "Page Caching"
- Chapter 3, "Blob Objects"
- Chapter 4, "Application Servers"

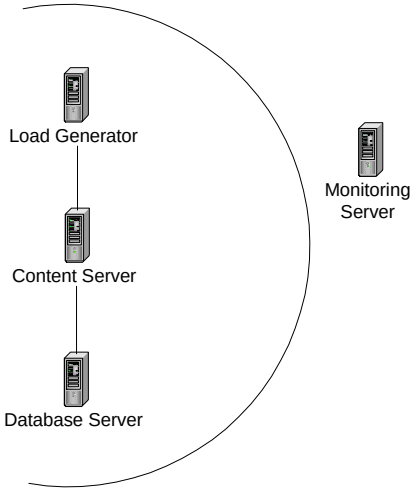
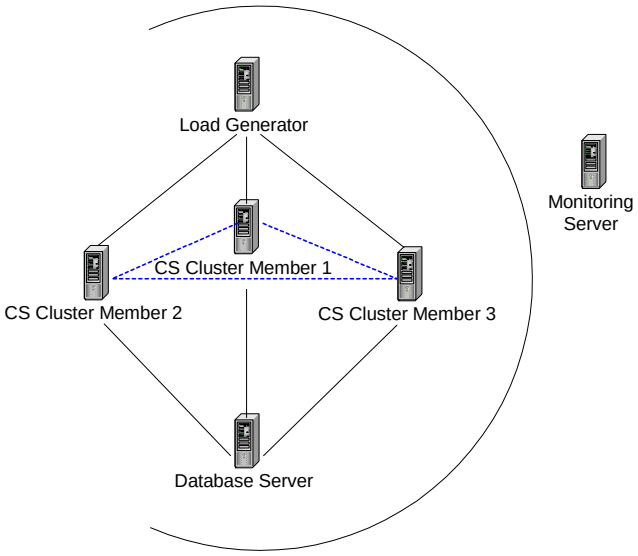


Figure 2: Horizontally Clustered Content Servers

This configuration was used to test 2-CS and 3-CS clusters. Clusters were created by adding either one or two CS nodes to the base configuration in Figure 1. (The diagram at the right shows the configuration for the maximum number of CS nodes.)

We tested each cluster's ability to handle hits per second and support user load in various page caching scenarios. For test results, see:

- Chapter 5, "Horizontal Clustering"



Hardware

The following machine configurations were used for testing:

Table 1: Hardware Specifications

Component	Model	Platform	CPU Speed	Number of CPU	Memory, GB	Drive Capacity, GB
Load Generator	HP Desktop	x86	3.0GHz	1	2	300
Content Server	Sunfire V2 0Z	x86 x 64	2.4GHz	2	8	72
Database Server	Sunfire V2 0Z	x86 x 64	2.4GHz	2	8	36
Monitoring Server	Poweredge 245	x86	733MHz	2	1	18

Software

The following software was installed on the machines.

Table 2: Software Specifications

Component	OS	Arch	App Server	Database
Load Generator	Win XP SP2	32 bit	—	—
Content Server	Linux SUSE 9.3 Pro	32 bit	WebLogic 8.1.5[a]	Oracle 10G R2
Database Server				
Monitoring Server				

a. WebLogic was used in all configurations and tests. Other supported application servers were substituted only when their performances were being tested relative to the performance of WebLogic. For information about application server performance, see Chapter 4, “Application Servers.”

Part 1

Scaling Factors

This part presents scaling charts, scaling factors, and computational methods for the Content Server systems shown in "System Configurations," on page 9.

This part contains the following chapters:

- Chapter 2, "Page Caching"
- Chapter 3, "Blob Objects"
- Chapter 4, "Application Servers"
- Chapter 5, "Horizontal Clustering"

Chapter 2

Page Caching

This chapter reports scaling factors for a single-CS system serving a mix of cached and uncached pages. Charts show how load scales with percent page caching at selected response times.

This chapter contains the following sections:

- Results
- Scaling Charts
- Scaling Factors
- Methods

Results

Load scales directly with percent page caching at response times of 1, 3, and 5 seconds. **All pages are served by Content Server.**

Scaling Charts

The scaling charts on page 15 were determined for:

- The single-CS configuration shown in Figure 1, on page 9.
- The baseline page described in "Page Design," on page 36.
- The following page caching scenarios:

Table 3: Page Caching Scenarios

Scenario	Description[a]
0%	No users were hitting the cached version of the page. (All users were hitting the uncached version of the page.)
25%	25% of the users were hitting the cached version of the page.
50%	50% of the users were hitting the cached version of the page.
62.5%	62.5% of the users were hitting the cached version of the page.
75%	75% of the users were hitting the cached version of the page.
87.5%	87.5% of the users were hitting the cached version of the page.
93.5%	93.5% of the users were hitting the cached version of the page.
100%	All users were hitting the cached version of the page.

a. In each scenario, we maintained the percentage of users calling cached versions of the pages at a constant value (for example, 25%). The remaining users (75% in this example) called the uncached version of the pages. As we increased load, we kept the proportion constant (25:75 in this example).

Load was scaled as explained in the text to the left of Charts 1 and 2. Details of the scaling calculations are given in "Methods," on page 16.

Chart 1.
Load scaling at 1- and 3-second response times *vs.* percent cached pages.

(Scaling factors were obtained by normalizing to the load supported at 100% caching at the given response time.)

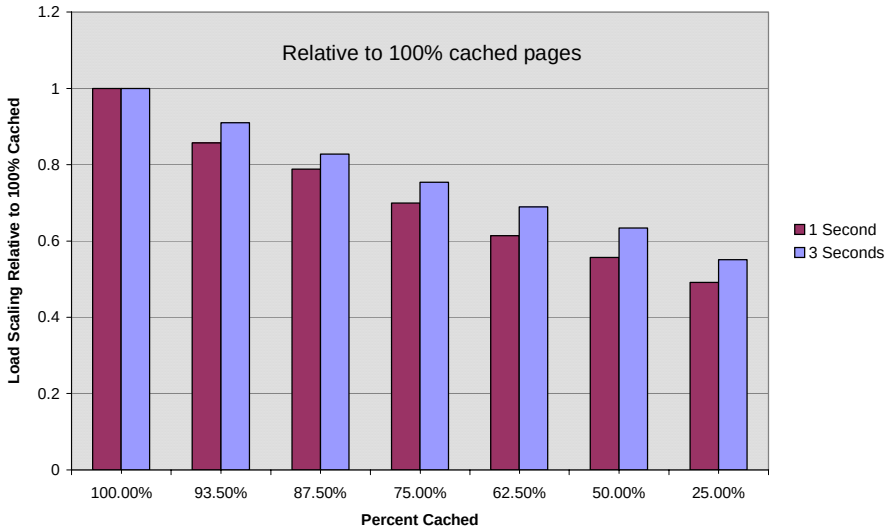
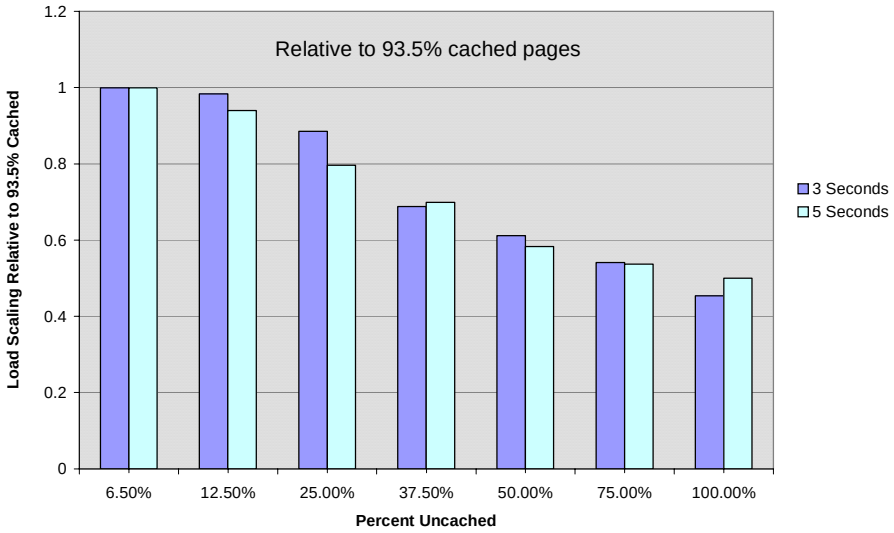


Chart 2.
Load scaling at 3- and 5-second response times *vs.* percent uncached pages.

(Scaling factors were obtained by normalizing to the load supported at 93.5% caching at the given response time.)



Scaling Factors

Note

For the scaling factors in our charts to be meaningful for your environment, you need to ensure two conditions:

- Your page design is similar to the design that is specified in this guide (see Chapter 6, "Page Design and Template Specifications").
- You are setting realistic limits on the expected performance of your system. In our scenarios, a realistic response time is not greater than 3 seconds for cached pages, and not greater than 5 seconds for uncached pages.

When determining scaling factors, we aimed to answer the following question:

> "What is the effect on load if I change the percentage of cached pages but maintain the response time?"

Based on the charts, load is directly proportional to percent page caching. Your system supports progressively greater loads as you increase the percentage of cached pages.

For example, referring to Chart 1:

- If your system is serving 25% cached pages at 1-second response time, it is handling 50%L_{max} (where L_{max} is the load supported by the system serving 100% cached pages).
- Increasing percent page caching to 50% raises system capacity to 55%L_{max}.
- Similarly, increasing percent page caching to 62.5% raises system capacity to 61%L_{max}, and so on. (Note that increasing response time to 3 seconds further raises system capacity.)
- To maximize load, you can allow for 100% page caching or a 3-second response time (which supports greater loads than does a 1-second response time).

The same type of analysis applies to Chart 2 (load scaling for uncached pages).

Methods

To simulate percent caching scenarios:

1. We used different percentages of users to call cached and uncached pages (as outlined in Table 3, on page 14).
2. In each caching scenario we varied load through the saturation region, where hits per second no longer change as load increases. Three sets of performance charts were obtained:
   - Chart A-1: Hits per second for all pages vs. ramp-up time
   - Chart A-2: Cached page response time (in seconds) vs. ramp-up time
   - Chart A-3: Uncached page response time (in seconds) vs. ramp-up time

3. From the performance charts, we calculated load scaling factors in the pre-saturation region for 1-second response times, and in the post-saturation region at 3-second and 5-second response times. The scaling factors are plotted in Charts 1 and 2 on page 15.

Chapter 3

Blob Objects

This chapter reports scaling factors for a single-CS system using Blob Server. Charts show how load scales with the number of blob objects on a page.

This chapter contains the following sections:

- Results
- Scaling Charts
- Scaling Factors
- Methods

Results

Load scales inversely with the number of blob objects (up to 40) on cached and uncached pages served at 1-, 3-, and 5-second response times. **All the blobs are served by Blob Server as pages are served by Content Server**.

Scaling Charts

The charts on page 21 were determined for:

- The single-CS configuration shown in Figure 1, on page 9.
- The baseline page described in "Page Design," on page 36. During testing we supplemented this page with 10, 20, and 30 additional blob objects (20KB for each object).
- Caching scenarios selected from those defined in Table 3, on page 14.

Scaling for any caching scenario is relative to the load supported by the system at 10 blob objects on the baseline page for the same caching scenario.

Chart 3.
Load scaling *vs.* number of blob objects on cached/ uncached pages at 1-second response time.

(In each caching scenario, scaling factors were obtained by normalizing to the load supported at 10 blob objects and 1-second response time in the same caching scenario.)

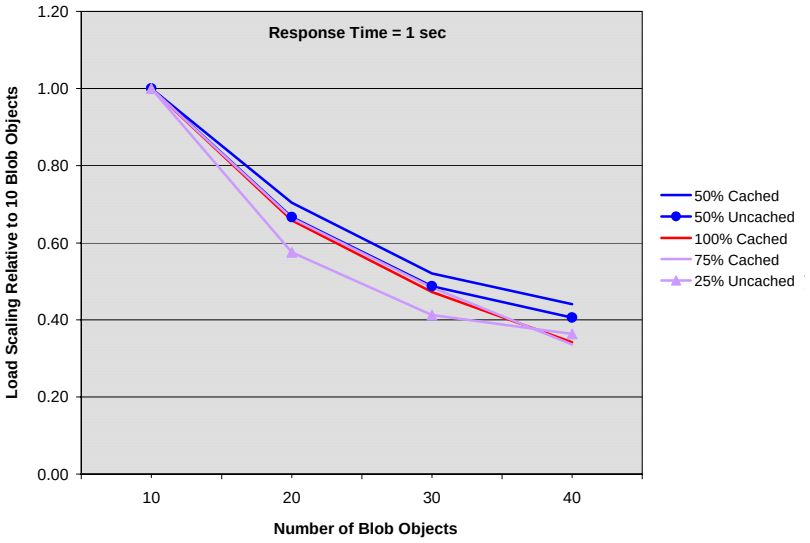


Chart 4.
Load scaling *vs.* number of blob objects on cached/ uncached pages at 3-second response time.

(In each caching scenario, scaling factors were obtained by normalizing to the load supported at 10 blob objects and 3-second response time in the same caching scenario.)

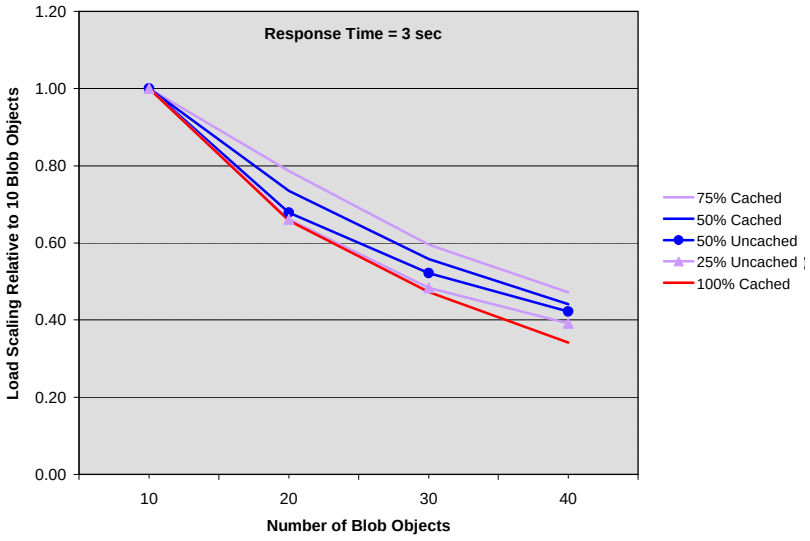
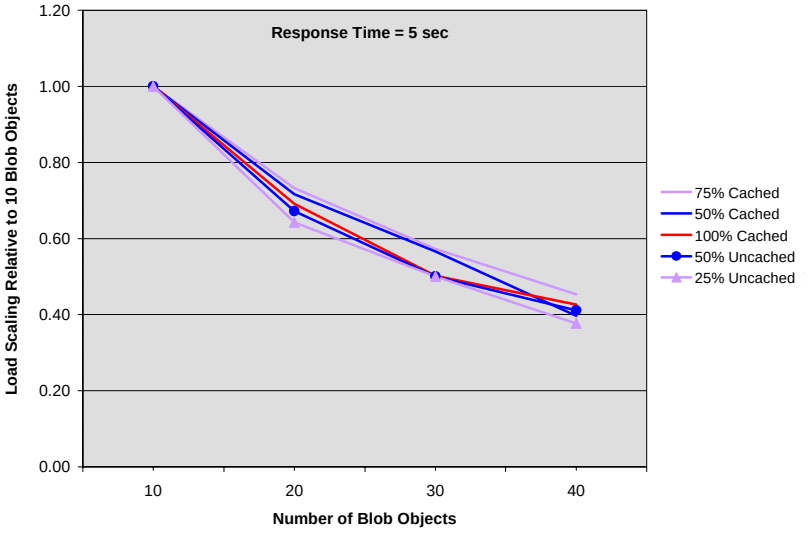


Chart 5.
Load scaling *vs.* number of blob objects on cached/ uncached pages at 5-second response time.

(In each caching scenario, scaling factors were obtained by normalizing to the load supported at 10 blob objects and 5-second response time in the same caching scenario.)



Scaling Factors

Note

For the scaling factors in our charts to be meaningful for your environment, you need to ensure two conditions:

- Your page design is similar to the design that is specified in this guide (see Chapter 6, "Page Design and Template Specifications").
- Pages are being served under a load that results in 1-second, 3-second, and 5-second response times.

When determining scaling factors, we aimed to answer the following question:

> "What is the effect on load if I change the number of blob objects on the pages of my site?"

Based on the charts, load is inversely proportional to the number of blob objects on a page. Your system supports progressively greater loads as you decrease the number of blob objects per page.

For example, referring to Chart 3, 100% caching:

- If your system currently serves 40 blob objects per page at 1-second response time, it is handling 38% L_{max} (where L_{max} is the load supported by the system serving 10 blob objects per page for the same caching scenario and response time). If you redesign your pages to display 10 blob objects, your system can now support 2.3 times as many users (at 1-second response time).
- The chart can also be read in reverse. Increasing the number of blob objects from 10 to 20 per page while maintaining a 1-second response time causes the load to drop to 69%L_{max}.

  Similarly, increasing the number of blob objects from 10 to 30, causes load to drop to 50%L_{max}.

The same type of analysis applies to Charts 4 and 5.

Methods

To determine scaling factors (i.e., how load is affected by changes in the number of blobs), we used our baseline page to create three new conditions in which we increased the number of blob objects in steps of 10 (from the default 10) to produce 20, 30, and 40 images per page, at 20KB per image. (For more information about the baseline page, see Chapter 6, "Page Design and Template Specifications.")

1. Five caching scenarios were tested: 100% cached, 75% cached, 50% cached, 50% uncached, and 25% uncached.

   In each caching scenario, scaling was first tested at a 1-second response time:

   a. The number of blobs on the baseline page was increased to 20, users were gradually ramped up (to a maximum of 5X users), response times were measured, and load was noted for a response time exceeding 1 second. The same procedure was repeated for 30 and 40 blobs.

b. For each blob count, the load at the given response time was normalized to the load at 10 blobs.

c. The normalized values were plotted as scaling factors against the number of blobs (Chart 3).

2. Step 1 was repeated for 3-second and 5-second response times. Scaling factors are plotted in Charts 4 and 5.

Chapter 4

Application Servers

This chapter compares the performance of supported application servers in the CS 6.3 Patch 1 release.

This chapter contains the following sections:

- Results
- Scaling Chart
- Scaling Factors
- Methods
- Test Conditions

Results

Application servers in order of decreasing performance are: Resin 3.0.18, Tomcat 5.0.2, JBoss 4.0.3 SP1, WebSphere 6.0.1, and either Sun JES 8.1 (2005 Q4) or WebLogic 8.1 SP5 (depending on the caching scenario).

Note

This test is the only one that uses different application servers. All other tests used only WebLogic.

Scaling Chart

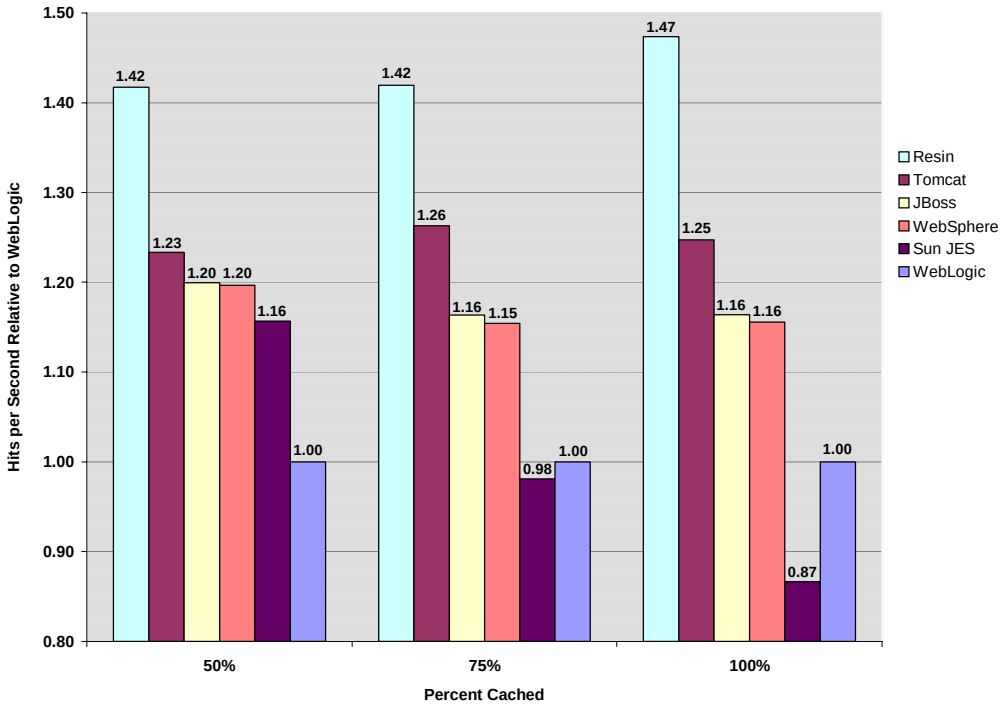
Chart 6 shows how application servers in the Content Server 6.3 Patch 1 release perform relative to WebLogic when 50%, 75%, and 100% cached **pages are served by Content Server**. The chart was determined for:

- The single-CS configuration shown in Figure 1, on page 9
- The baseline page described in "Page Design," on page 36
- Caching scenarios selected from those defined in Table 3, on page 14

Hits per second were scaled as explained in the text to the left of Chart 6. The chart's legend lists the application servers in order of performance, starting with the best performer. For version numbers of the application servers, see "Results" on this page (and "Test Conditions," on page 27).

Chart 6. Hits per second relative to WebLogic *vs.* percent pages cached.

(In each caching scenario, scaling factors were obtained by normalizing each app server's peak hits-per-second to WebLogic's peak hits per second in the same scenario.)



Scaling Factors

Note

For the scaling factors in our charts to be meaningful for your environment, your page design must be similar to the design that is specified in this guide (see Chapter 6, "Page Design and Template Specifications").

Application server performance reported in this guide is specific to our test conditions and is not a reflection of the products' general performance.

When determining scaling factors, we aimed to answer the following question:

"What is the effect on hits per second if I switch to a different application server?"

Based on Chart 6, hits per second improve with application servers in the following order: WebLogic, Sun JES, WebSphere, JBoss, Tomcat, and Resin.

- For example, if your system is now using WebLogic and serves 75% cached pages, switching to Resin will improve hits per second by a factor of 1.42.
- Whereas Resin, Tomcat, JBoss, and WebSphere are consistently the top four performers, Sun JES shows variability by underperforming WebLogic in the 100% page caching scenario, matching it at 75%, and outperforming WebLogic at 50%.

Methods

To determine how each application server's performance scales with percent pages cached, we ran three tests—100%, 75 and 50% cached—identical to those in the caching scenarios (Chapter 2, "Page Caching"). Since hits per second is a good indication of how much load a server can handle before it becomes saturated (leading to higher response times per transaction), this metric was used to compare the performances of the application servers.

The maximum hits per second for each application server was recorded for each caching scenario and normalized to the value returned by WebLogic in the same caching scenario. For example, if at 75% cached pages JBoss served 491 hits per second and WebLogic served 422 hits per second, JBoss performed 1.16 times better than WebLogic (491/422 = 1.163).

Test Conditions

1. WebLogic is the baseline system to which all other systems are compared. WebLogic is configured as in all other tests.
2. All tests were carried out as for page caching and clustering on WebLogic.
3. The WebLogic JVM was used for configurations using WebLogic. The Sun JES and WebSphere JVM were used, as each is provided with its application server.
4. Performance settings may not be optimal for all application servers. A best effort was made to tune each server in the given time, but no extensive testing could be performed.

Chapter 5

Horizontal Clustering

This chapter reports scaling factors for horizontally clustered systems. Charts show how hits per second scale with the number of CS nodes.

This chapter contains the following sections:

- Results
- Scaling Charts
- Scaling Factors
- Methods

Results

Hits per second to cached and uncached pages scale directly with the number of Content Server nodes (up to three nodes) in a horizontal cluster.

For a fixed number of nodes, scaling factors agree to within 15% (within experimental error) as percent caching varies.

All pages are served by Content Server.

Scaling Charts

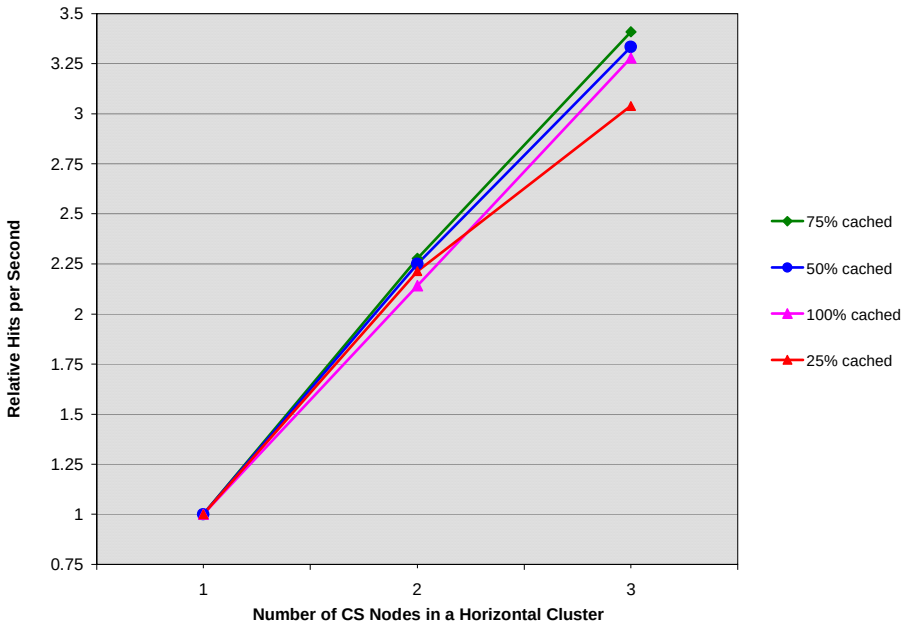
Chart 7 was determined for:

- The single-CS configuration shown in Figure 1, on page 9. During testing, we added one and two Content Server nodes to the single-CS configuration. (The three-node system is shown in Figure 2, on page 9.)
- The baseline page described in "Page Design," on page 36.
- Caching scenarios selected from those defined in Table 3, on page 14.

Load was scaled as explained in the text to the left of Chart 7.

Chart 7. Scaling of hits per second *vs.* number of CS nodes in a horizontal cluster for different percent caching.

(In each caching scenario, scaling factors were obtained by normalizing the cluster's peak hits-per-second to the peak hits-per-second supported by a single-CS system in the same caching scenario.)



Scaling Factors

Note

For the scaling factors in our charts to be meaningful for your environment, your page design must be similar to the design that is specified in this guide (see Chapter 6, "Page Design and Template Specifications").

When determining scaling factors, we aimed to answer the following questions:

- "What is the effect on hits per second if I change the number of CS nodes in a caching scenario?"

  Based on Chart 7, hits per second are directly proportional to the number of CS nodes in a cluster.

  - For one additional CS node (2 nodes total), hits per second improves by a factor of 2.2 (on average) compared to the single-node system.
  - For two additional nodes (3 nodes total), hits per second improves by a factor of 3.2 (on average) compared to the single-node system.

  For example, if a single CS node can handle 100 hits per second, then two CS nodes can handle 220 hits per second, and three CS nodes can handle 320 hits per second.

- "How does performance scale across percent caching scenarios?"

  For a fixed number of nodes in a horizontal cluster, scaling factors agree to within 15% as percent caching varies.

Methods

In order for us to test across a cluster of CS nodes, certain changes were required in how the load was generated. Because load balancers do not work correctly when a large number of users originate from a single source (1,500 in the case of three CS nodes), it was necessary to direct the traffic manually to the correct node. This was done using a random probability that each server would be used for a given iteration. The result is that, like in the real world, the load is never equally distributed across servers at a given instant. Over time, however, the distribution is balanced.

Part 2

System Specifications

This part presents specifications of the Content Server systems that we used in our tests.

This part contains the following chapters:

- Chapter 6, “Page Design and Template Specifications”
- Chapter 7, “Test Conditions”
- Chapter 8, “Template Code”

Chapter 6

Page Design and Template Specifications

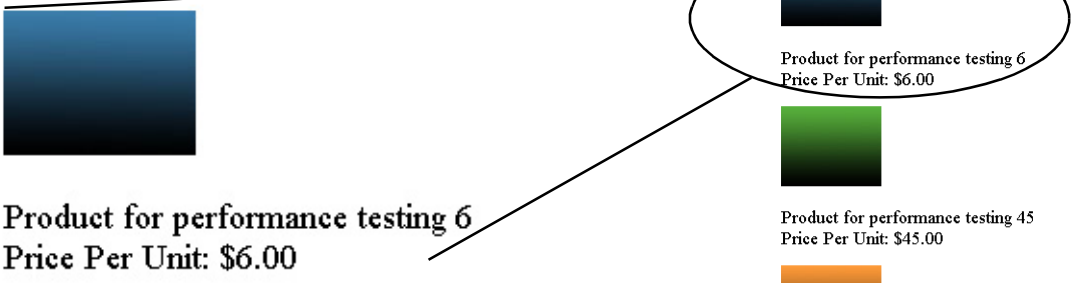
This chapter describes the pages that we designed for our tests and the templates that rendered the pages.

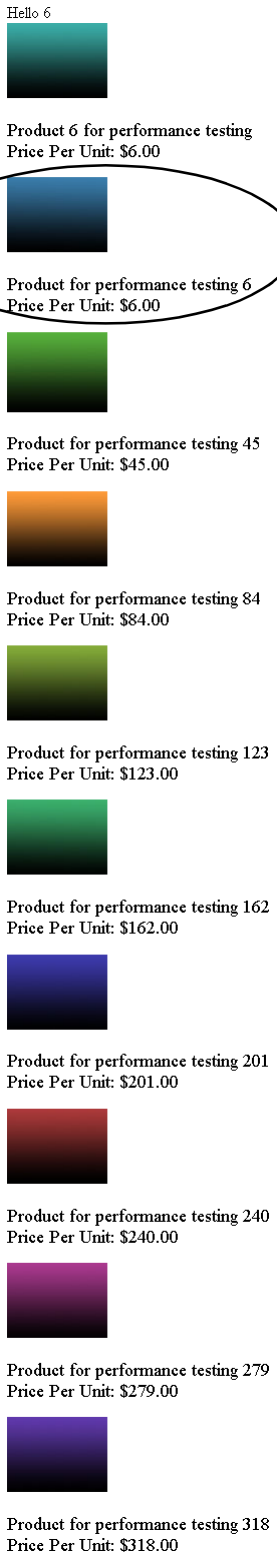
This chapter contains the following sections:

- Page Design
- Template Specifications

Page Design

We designed our system to render a maximum of 5,000 unique product assets over 500 *cached* pages. We repeated the same conditions over 500 *uncached* versions of the same pages (all dynamic). Each page looks like the one at the right—i.e., it lists 10 products. Each product consists of a unique product description and a media asset (a 20KB image).





Across the 500 pages, no two product descriptions are identical, but any one of 500 media assets (images) is used for the product image.

Page Content

We based our page design on the asset framework in FirstSiteII, specifically the “Product” and “Media” asset families (both flex families).

- The “Product” family has an attribute of type “Asset” referencing a media asset. The “Media” family has an attribute of type “Blob.”
- The “Product” family is the source of the product description. The “Media” family is the source of the 20KB image.
- To ensure that 10 unique product assets would be rendered per page, we created a `SortNum` attribute for the “Product” family.

Page Rendering

We designed two sets of templates: one set for rendering cached pages, and one set for rendering uncached pages. Each set of templates uses `Product_C/FSII Summary` to render the product description, and `Media_C/FSII Summary` to render the image. The call structure of the templates is given on page 38. Descriptions of the templates are given on page 39.

Page Caching

- To simulate percent pages cached, we varied the percentage of users that called the cached and uncached versions of a page (rendered by the template named `Product_C/ProductListNested`). For example, to simulate 100% cached pages, we had 100% of the virtual users call the cached version of `Product_C/ProductListNested` (via the `Page/ProdList1` wrapper template). To simulate 60% cached pages, we had 60% of the virtual users call the cached versions of `Product_C/ProductListNested` (also via the `Page/ProdList1` wrapper template), and 40% call the uncached version.
- We also removed "Context" from the caching criteria for templates, as it interfered with caching. (In FirstSiteII, all templates are automatically assigned the "Context" cache criterion.)
- All pages were preloaded into cache before performance tests were run.

Template Specifications

This section contains the following information:

- "Template Call Structure," on page 38
- "Template Descriptions," on page 39

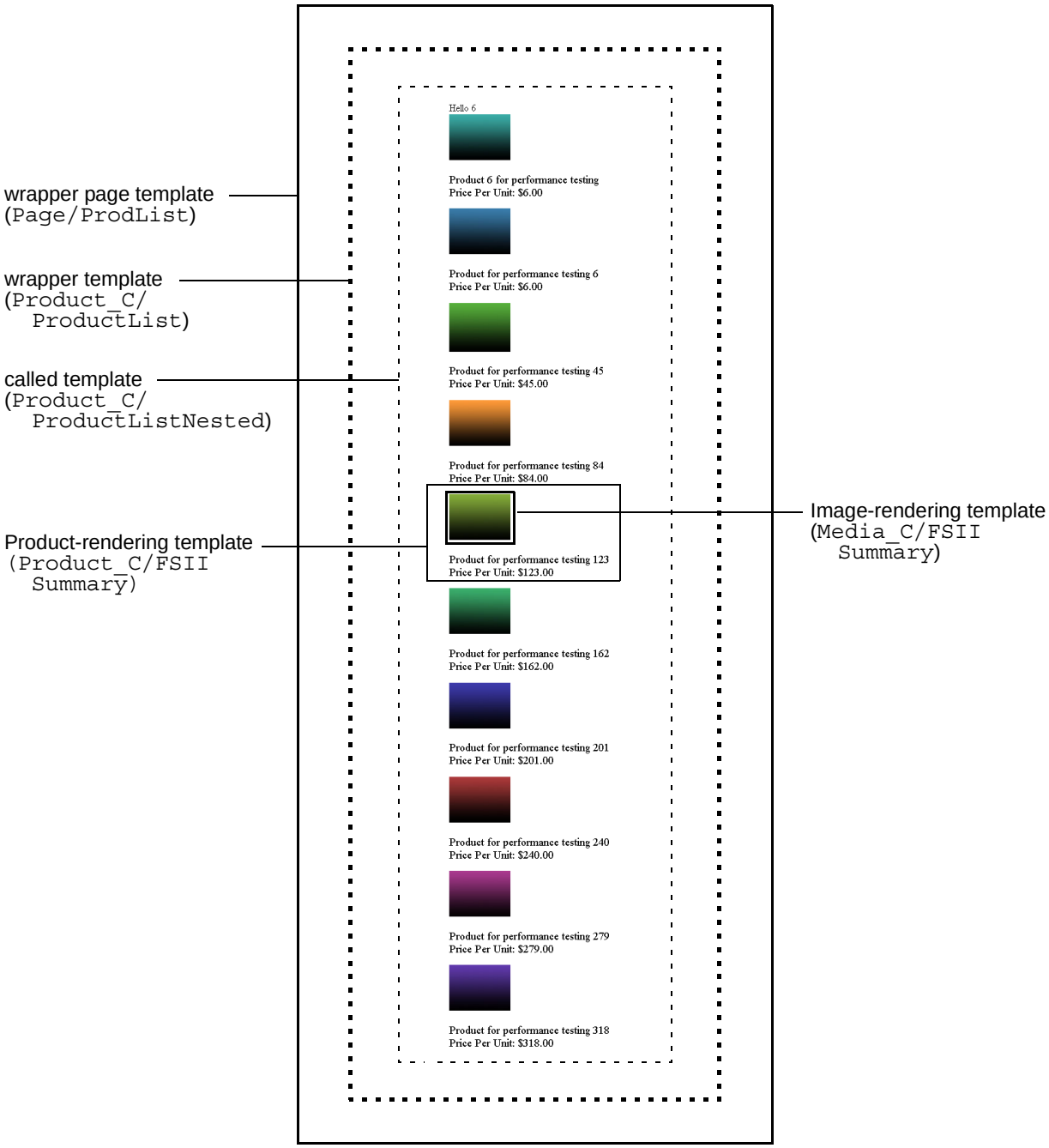
For template code, see Chapter 8.

Template Call Structure

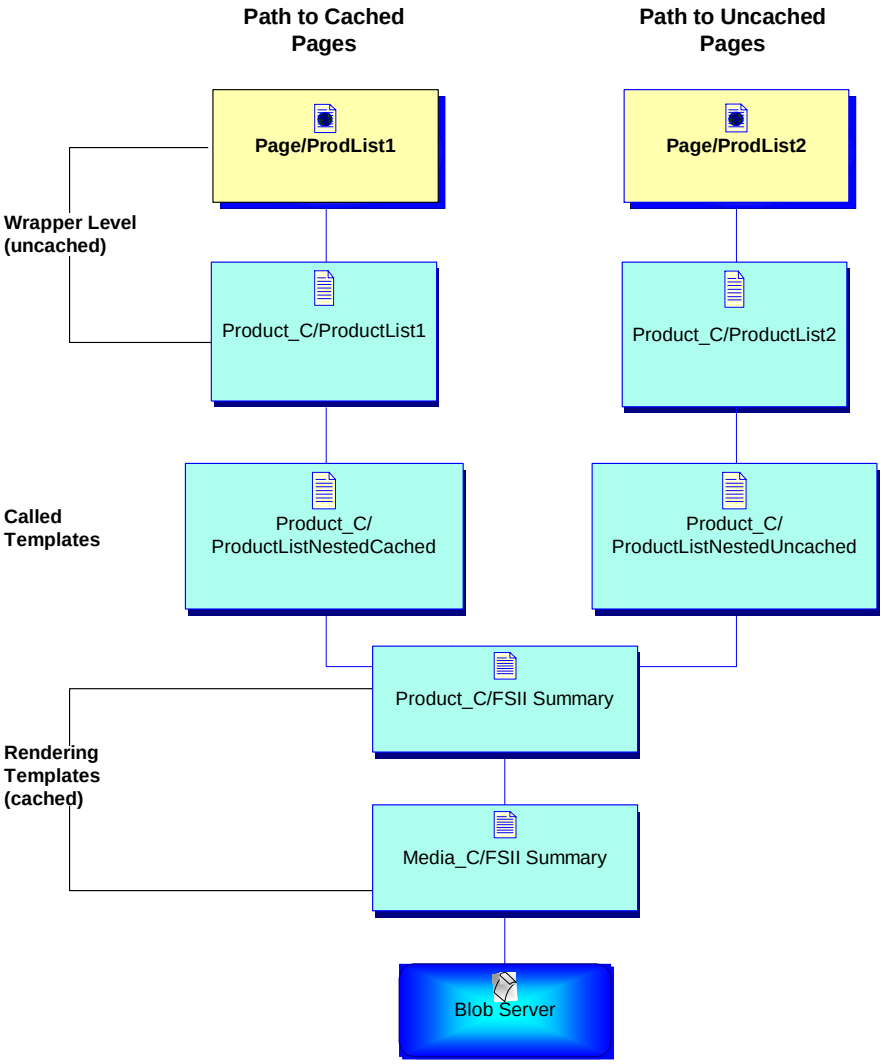
Figure 3: Template call structure for rendering cached and uncached versions of a page

Solid lines denote templates that render the objects being pointed to.

Dotted lines denote templates that provide the business logic.



Template Descriptions



Wrapper Templates

The wrapper level is uncached.

- The first tier of the wrapper level contains the following templates:
  - `Page/ProdList1` is a wrapper page template that calls the `Product_C/ProductList1` template.
  - `Page/ProdList2` is a wrapper page template that calls the `Product_C/ProductList2` template.
- The second tier of the wrapper level contains the following templates:
  - `Product_C/ProductList1` is a wrapper template that generates a random number (`num`) and uses that random number as a parameter to call a cached template (`Product_C/ProductListNestedCached`).

- `Product_C/ProductList2` is identical to `Product_C/ProductList`1, except that it calls a cached template (`Product_C/ProductList NestedCached`).

Called Templates

One of the called templates is cached; the other is not (as the template names indicate).

- `Product_C/ProductListNestedCached` uses the random number `num` to create a searchstate of the product assets and find the assets that match the value for the `SortNum` attribute (which we created). There are 10 products for each value of `SortNum`. And then goes through a loop to render the information about each of the product assets returned by the searchstate. It calls the `Product_C/FSIISummary` with the id of the product asset. This template is cached according to the `num` parameter.
- `Product_C/ProductListNestedUncached` is the uncached version of `Product_C/ProductListNestedCached`.

Rendering Templates

The rendering templates are cached.

- `Product_C/FSIISummary` renders the product description and leaves room for the 20KB image, provided by the media asset. The `Product_C/FSIISummary` template uses the id passed by the `ProductListNested` template to get information from the `Product_C` table about the product asset, and then calls the `Media_C/FSIISummary` template with the id of the `Media_C asset`. This template is cached based on the id of the product asset.
- `Media_C/FSIISummary` renders the image for the product description. This template uses the id passed by `Product_C/FSIISummary` template to get information from the `Media_C` table about the image asset, and then calls the blob server to render the image. This template is cached according to the id of the media asset.

Chapter 7

Test Conditions

This chapter summarizes performance testing conditions.

This chapter contains the following sections:

- Setting Up
- Monitoring

Setting Up

To ensure that all sizing tests were consistent for scenarios involving cached pages, we configured the systems such that no data would be dropped from cache during testing. We also ramped up virtual users in a way that ensured completeness of their transactions.

Pre-Loading

To prevent dropping of data from cache, prior to running each test we ran automated scripts to load all our pages and blobs into the cache (500 pages consisting of 5,000 product assets and 10 unique images). This loaded all the page caches and resultset caches into memory, which we pre-configured to be large enough to hold all the data.

Ramp-Up

In order to simulate an ever increasing number of users on a site and determine how the increase in users affects performance, a constant ramp-up was used. Five hundred users were ramped up over 255 minutes (4 1/4 hours). At the end of the ramp up, the 500 users were allowed to run for 45 minutes. This steady-state period was used to ensure that the system is not overloaded, that performance remains steady for this period, and that the signal-to-noise ratio is reasonably high.

The ramp-up interval is 4 users every 2 minutes, based on the concept that a user completes a transaction in less than 30 seconds (even in the worst case), and therefore has the time to complete at least 4 transactions before the load is increased. Thus, within a 2-minute interval, a steady state is also achieved.

Monitoring

Each test was monitored at the following levels: operating system, application server, and end-user level. All three results were then analyzed in order to determine how the system responded as a whole throughout the test. The database was monitored in real-time.

Operating System Monitoring

The operating system was monitored via automated scripts. The master script copied a monitoring script to each OS to be monitored then recorded data at given intervals and finally returned the gathered information to the master monitor script.

Collected data consists of:

- Open sockets
- System statistics (including but not limited to memory utilization, CPU utilization, CPU queue lengths, context switching, and network statistics)
- Socket information
- Per process statistics (including, but not limited to memory utilization, CPU utilization, and faults)

Application Server Monitoring

WebLogic was monitored using SNMP requests sent to the individual WebLogic servers at given intervals by the master monitoring script.These statistics included garbage collections, thread counts, queue lengths, and memory utilization.

End-User Monitoring

The load generation tool captured statistics including response times, hits per second, throughput, and number of failures.

Database Monitoring

The database (Oracle 10GR2) was monitored using a real-time health monitor which allows the database's health (and to some extent, performance) to be monitored. Statistics that were viewed include:

- Network bandwidth utilized
- SQL queries per second
- Shared pool utilization
- Redo logs
- SQL queries executed (including total counts and time spent on average)
- Number of total connections to the database

Chapter 8

Template Code

- Page/ProdList1
- Product_C/ProductList1
- Product_C/ProductListNestedCached
- Page/ProdList2
- Product_C/ProductList2
- Product_C/ProductListNestedUncached
- Product_C/FSII Summary
- Media_C/FSII Summary

Note

For descriptions of the templates and their function in page generation, see Chapter 6, "Page Design and Template Specifications."

Page/ProdList1

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ taglib prefix="render" uri="futuretense_cs/render.tld"
%><%@ taglib prefix="string" uri="futuretense_cs/string.tld"
%><%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"
%><%@ page import="java.util.Random"
%><cs:ftcs>
<ics:if
 condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logde
 p cid='<%=ics.GetVar("tid")%>'   c="Template"/></ics:then></
 ics:if>
<div id="ProductDetailView">

   <div id="ProdListArea">

      <render:calltemplate tname='ProductList'
            site='<%=ics.GetVar("site")%>'
            tid='<%=ics.GetVar("tid")%>'
            slotname="ProductList"
            c='Product_C'
            cid='<%=ics.GetVar("p")%>'
                                                ttype="Template">
      <render:argument name="p" value='<%=ics.GetVar("p")%>'/>
      </render:calltemplate>
   </div>

</div>
</cs:ftcs>
```

Product_C/ProductList1

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"
%><%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"
%><%@ taglib prefix="commercecontext" uri="futuretense_cs/
 commercecontext.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ taglib prefix="listobject" uri="futuretense_cs/
 listobject.tld"
%><%@ taglib prefix="render" uri="futuretense_cs/render.tld"
%><%@ taglib prefix="siteplan" uri="futuretense_cs/siteplan.tld"
%><%@ taglib prefix="searchstate" uri="futuretense_cs/
 searchstate.tld"
%><%@ page import="COM.FutureTense.Interfaces.*,
                   COM.FutureTense.Util.ftMessage,
                   COM.FutureTense.Util.ftErrors"
%><%@page import="java.util.Random"
%><cs:ftcs>

<%-- Product_C/ProductList

INPUT

OUTPUT

--%>

<%-- Record dependencies for the Template --%>

    <ics:if
     condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logde
     p cid='<%=ics.GetVar("tid")%>' c="Template"/></ics:then></
     ics:if>

       <%
       String num;
       Random r = new Random();
       int i = r.nextInt( 100 );
       num = "" + (i + 6);
       %>

    <render:satellitepage pagename='FirstSiteII/Product_C/
     ProductListNested'>
        <render:argument name="p" value='<%=ics.GetVar("p")%>'/>
```

```
          <render:argument name="sitepfx"
        value='<%=ics.GetVar("sitepfx")%>'/>
          <render:argument name="site" value='<%=ics.GetVar("site")%>'/>
          <render:argument name="num" value='<%=num%>'/>
<%--      <render:argument name="num" value='20'/> --%>

     </render:satellitepage>

     </cs:ftcs>
```

Product_C/ProductListNestedCached

Cached on Content Server

```
cache criteria = c,cid,num,p,rendermode,site,sitepfx,ft_ss


<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="commercecontext" uri="futuretense_cs/
  commercecontext.tld"
%><%@ taglib prefix="siteplan" uri="futuretense_cs/siteplan.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ taglib prefix="render" uri="futuretense_cs/render.tld"
%><%@ taglib prefix="string" uri="futuretense_cs/string.tld"
%><%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"
%><%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"
%><%@ taglib prefix="searchstate" uri="futuretense_cs/
  searchstate.tld"
%><%@ taglib prefix="listobject" uri="futuretense_cs/
  listobject.tld"
%><%@ page import="COM.FutureTense.Interfaces.*,
                   COM.FutureTense.Util.ftMessage,
                   COM.FutureTense.Util.ftErrors"
%>
<%-- Product_C/ProductListNestedUncached --%>

<cs:ftcs>
<ics:if
  condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logde
  p cid='<%=ics.GetVar("tid")%>'   c="Template"/></ics:then></
  ics:if>
<div id="ProductListSummaryView">

Hello

<ics:getvar name="num" />

<%-- use search state based on attribute num. --%>

<render:lookup key="SummaryTname" varname="SummaryTname"
  site='<%=ics.GetVar("site")%>' tid='<%=ics.GetVar("tid")%>'/>
<!-- create list for search state -->
<listobject:create name="NumList" columns="SortNum"/>
<listobject:addrow name="NumList">
<listobject:argument name="SortNum"
  value='<%=ics.GetVar("num")%>'/>
</listobject:addrow>
```

```
     <listobject:tolist name="NumList" listvarname="NumListQ"/>

<%-- First, populate the searchstate --%>

          <searchstate:create name="Search" op="or"/>
          <searchstate:addlikeconstraint name="Search"
       typename="Product_A" attribute="SortNum" list="NumListQ"
       caseinsensitive="true"/>

<%-- Now do the lookup and display the results --%>

     <assetset:setsearchedassets name="ProductSet" constraint="Search"
       assettypes='<%=ics.GetVar("ProductType")%>'/>
     <assetset:getassetlist name="ProductSet"
       listvarname="ProductList"/>
     <ics:if condition='<%= ics.GetList("ProductList") != null &&
       ics.GetList("ProductList").hasData() %>'>
     <ics:then>
          <div id="SearchResultsList">


<%--                 <% int i =0; %> --%>

               <ics:listloop listname="ProductList">

<%--                  <% i++;%>  --%>

                    <ics:listget listname="ProductList"
       fieldname="assettype" output="assettype"/>
                    <ics:listget listname="ProductList"
       fieldname="assetid" output="assetid"/>
                    <div class="SearchResult">
                       <render:calltemplate
       tname='<%=ics.GetVar("SummaryTname")%>'
                         site='<%=ics.GetVar("site")%>'
                         tid='<%=ics.GetVar("tid")%>'
                         slotname="SearchResultEntry"
                         c='<%=ics.GetVar("assettype")%>'
                         cid='<%=ics.GetVar("assetid")%>'
                         ttype="Template">
                         <render:argument name="p"
       value='<%=ics.GetVar("p")%>'/>

<%-- <render:argument name="imgnum" value='<%=Integer.toString(i)%>'/>
     --%>
```

```
                </render:calltemplate>

            </div>
          </ics:listloop>
        </div>
    </ics:then>
    </ics:if>
    </div>
    </cs:ftcs>
```

Page/ProdList2

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ taglib prefix="render" uri="futuretense_cs/render.tld"
%><%@ taglib prefix="string" uri="futuretense_cs/string.tld"
%><%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"
%><%@ page import="java.util.Random"
%><cs:ftcs>
<ics:if
 condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logde
 p cid='<%=ics.GetVar("tid")%>'   c="Template"/></ics:then></
 ics:if>
<div id="ProductDetailView">

   <div id="ProdListArea">

      <render:calltemplate tname='ProductListUncached'
            site='<%=ics.GetVar("site")%>'
            tid='<%=ics.GetVar("tid")%>'
            slotname="ProductList"
            c='Product_C'
            cid='<%=ics.GetVar("p")%>'
                                                ttype="Template">
      <render:argument name="p" value='<%=ics.GetVar("p")%>'/>
      </render:calltemplate>
   </div>

</div>
</cs:ftcs>
```

Product_C/ProductList2

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"
%><%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"
%><%@ taglib prefix="commercecontext" uri="futuretense_cs/
 commercecontext.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ taglib prefix="listobject" uri="futuretense_cs/
 listobject.tld"
%><%@ taglib prefix="render" uri="futuretense_cs/render.tld"
%><%@ taglib prefix="siteplan" uri="futuretense_cs/siteplan.tld"
%><%@ taglib prefix="searchstate" uri="futuretense_cs/
 searchstate.tld"
%><%@ page import="COM.FutureTense.Interfaces.*,
                   COM.FutureTense.Util.ftMessage,
                   COM.FutureTense.Util.ftErrors"
%><%@page import="java.util.Random"
%><cs:ftcs>

<%-- Product_C/ProductList

     INPUT

     OUTPUT

--%>
<%-- Record dependencies for the Template --%>

     <ics:if
      condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logde
      p cid='<%=ics.GetVar("tid")%>' c="Template"/></ics:then></
      ics:if>

        <%
        String num;
        Random r = new Random();
        int i = r.nextInt( 100 );
        num = "" + (i + 6);
        %>

     <render:satellitepage pagename='FirstSiteII/Product_C/
      ProductListNestedUncached'>
         <render:argument name="p" value='<%=ics.GetVar("p")%>'/>
         <render:argument name="sitepfx"
      value='<%=ics.GetVar("sitepfx")%>'/>
         <render:argument name="site" value='<%=ics.GetVar("site")%>'/>
```

```
        <render:argument name="num" value='<%=num%>'/>
<%--    <render:argument name="num" value='1'/>  --%>

    </render:satellitepage>

    </cs:ftcs>
```

Product_C/ProductListNestedUncached

```
<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="commercecontext" uri="futuretense_cs/
 commercecontext.tld"
%><%@ taglib prefix="siteplan" uri="futuretense_cs/siteplan.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ taglib prefix="render" uri="futuretense_cs/render.tld"
%><%@ taglib prefix="string" uri="futuretense_cs/string.tld"
%><%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"
%><%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"
%><%@ taglib prefix="searchstate" uri="futuretense_cs/
 searchstate.tld"
%><%@ taglib prefix="listobject" uri="futuretense_cs/
 listobject.tld"
%><%@ page import="COM.FutureTense.Interfaces.*,
                   COM.FutureTense.Util.ftMessage,
                   COM.FutureTense.Util.ftErrors"
%>

<%-- Product_C/ProductListNestedUncached--%>

<cs:ftcs>
<ics:if
 condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logde
 p cid='<%=ics.GetVar("tid")%>'   c="Template"/></ics:then></
 ics:if>
<div id="ProductListSummaryView">

Hello

<ics:getvar name="num" />

<%-- use search state based on attribute num. --%>

<render:lookup key="SummaryTname" varname="SummaryTname"
 site='<%=ics.GetVar("site")%>' tid='<%=ics.GetVar("tid")%>'/>
<!-- create list for search state -->
<listobject:create name="NumList" columns="SortNum"/>
<listobject:addrow name="NumList">
<listobject:argument name="SortNum"
 value='<%=ics.GetVar("num")%>'/>
</listobject:addrow>
<listobject:tolist name="NumList" listvarname="NumListQ"/>
```

```
<%-- First, populate the searchstate --%>

        <searchstate:create name="Search" op="or"/>
        <searchstate:addlikeconstraint name="Search"
      typename="Product_A" attribute="SortNum" list="NumListQ"
      caseinsensitive="true"/>

<%-- Now do the lookup and display the results --%>

     <assetset:setsearchedassets name="ProductSet" constraint="Search"
      assettypes='<%=ics.GetVar("ProductType")%>'/>
     <assetset:getassetlist name="ProductSet"
      listvarname="ProductList"/>
     <ics:if condition='<%= ics.GetList("ProductList") != null &&
      ics.GetList("ProductList").hasData() %>'>
     <ics:then>
        <div id="SearchResultsList">
           <ics:listloop listname="ProductList">
              <ics:listget listname="ProductList" fieldname="assettype"
      output="assettype"/>
              <ics:listget listname="ProductList" fieldname="assetid"
      output="assetid"/>
              <div class="SearchResult">
                 <render:calltemplate
      tname='<%=ics.GetVar("SummaryTname")%>'
                    site='<%=ics.GetVar("site")%>'
                    tid='<%=ics.GetVar("tid")%>'
                    slotname="SearchResultEntry"
                    c='<%=ics.GetVar("assettype")%>'
                    cid='<%=ics.GetVar("assetid")%>'
                    ttype="Template">
                    <render:argument name="p"
      value='<%=ics.GetVar("p")%>'/>
                 </render:calltemplate>
              </div>
           </ics:listloop>
        </div>
     </ics:then>
     </ics:if>
     </div>
     </cs:ftcs>
```

Product_C/FSII Summary

```
      cache criteria = c,cid,p,rendermode,site,sitepfx,ft_ss

      <%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
      %><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
      %><%@ taglib prefix="render" uri="futuretense_cs/render.tld"
      %><%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"
      %><%@ taglib prefix="string" uri="futuretense_cs/string.tld"
      %><%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"
      %><%@ taglib prefix="insite" uri="futuretense_cs/insite.tld"
      %><%@ taglib prefix="currency" uri="futuretense_cs/currency.tld"
      %><%@ taglib prefix="commercecontext" uri="futuretense_cs/
        commercecontext.tld"%>

<%-- The cs:ftcs tag creates an ICS object which provides access to
     Content Server functionality.  The tag also buffers output for
     caching. It is required in all Content Server JSPs. --%>

     <cs:ftcs>
     <div class="ProductSummary">

<%-- The template that contains this JSP file needs to record itself
     as existing on this pagelet when the pagelet is rendered.  The
     render:logdep tag exists for this purpose.

     Dependency tracking is done so that cached pages can be identified
     as invalid when content on them changes. Upon publish of a given
     asset, all pages that have been rendered that use the given asset
     will be flushed and then subsequently regenerated. --%>

     <ics:if
       condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logde
       p cid='<%=ics.GetVar("tid")%>' c="Template"/></ics:then></
       ics:if>

<%-- Several variables are passed into every template.  There is
     usually a variable "c" which is the asset type of the asset being
     rendered. "cid" is the id of the asset being rendered. "site" is
     the name of the current site and "sitepfx" is a short prefix name
     for the given site.

     All of these variables are required to render assets.
     First, create an asset set that contains only the single asset we
     want to render. --%>

     <assetset:setasset name="ProductSet" type='<%=ics.GetVar("c")%>'
       id='<%=ics.GetVar("cid")%>' />
```

```
<%-- Now that we've created a set (of one asset in this case) that we
     want to render, we can now retrieve the attributes belonging to
     that set.

      All attributes can be retrieved using
      assetset:getattributevalue(s), but if we know in advance that we
      are going to retrieve more than one attribute then we should
      instead use the assetset:getmultiplevalues tag, because it is
      much more efficient than making multiple calls to
      assetset:getattributevalue(s). In particular, there is a lot
      less database access with a single assetset:getmultiplevalues
      call. Text and blob attributes, however, cannot be retrieved
      using the assetset:getmultiplevalues tag, and they must be
      retrieved individually.

      Attribute names can change when a site is replicated, so we have
      to look up the actual attribute name in a database table
      maintained by the template asset.  We assigned an arbitrary key
      to the name of the attribute (that co-incidentally correpsonds
      to the name of the attribute in the original site). The
      render:lookup tag is used to look up asset references.


      First, look up the attribute type name --%>

     <render:lookup key="ProductAttrType" varname="ProductAttrType"
      site='<%=ics.GetVar("site")%>' tid='<%=ics.GetVar("tid")%>'/>

<%-- Next, look up each of the attributes we're going to render --%>

     <render:lookup key="NameAttrName" match=":x"
      varname="NameAttrName" site='<%=ics.GetVar("site")%>'
      tid='<%=ics.GetVar("tid")%>'/>
     <render:lookup key="ShortDescription" match=":x"
      varname="ShortDescription" site='<%=ics.GetVar("site")%>'
      tid='<%=ics.GetVar("tid")%>'/>
     <render:lookup key="PriceAttrName" match=":x"
      varname="PriceAttrName" site='<%=ics.GetVar("site")%>'
      tid='<%=ics.GetVar("tid")%>'/>
     <render:lookup key="ImageAttrName" match=":x"
      varname="ImageAttrName" site='<%=ics.GetVar("site")%>'
      tid='<%=ics.GetVar("tid")%>'/>

<%-- Retrieve them. --%>

     <assetset:getmultiplevalues name="ProductSet" prefix="product"
      immediateonly="true" byasset="false">
```

```
        <assetset:sortlistentry
      attributetypename='<%=ics.GetVar("ProductAttrType")%>'
      attributename='SortNum'/>
        <assetset:sortlistentry
      attributetypename='<%=ics.GetVar("ProductAttrType")%>'
      attributename='<%=ics.GetVar("ShortDescription")%>'/>
        <assetset:sortlistentry
      attributetypename='<%=ics.GetVar("ProductAttrType")%>'
      attributename='<%=ics.GetVar("PriceAttrName")%>'/>
        <assetset:sortlistentry
      attributetypename='<%=ics.GetVar("ProductAttrType")%>'
      attributename='<%=ics.GetVar("ImageAttrName")%>'/>
    </assetset:getmultiplevalues>

<%-- Now render the content.

    Be sure to check to see if optional fields actually have data
    specified. Required fields must have data and the UI enforces it,
    so it is not necessary to check. --%>

    <ics:if
      condition='<%=ics.GetList("product:"+ics.GetVar("ImageAttrName")
      ) != null &&
      ics.GetList("product:"+ics.GetVar("ImageAttrName")).hasData()%>'
      >
    <ics:then>
        <ics:listget
      listname='<%="product:"+ics.GetVar("ImageAttrName")%>'
      fieldname="value" output="ImageID"/>
        <render:lookup key="ImageType" varname="ImageType"
      site='<%=ics.GetVar("site")%>' tid='<%=ics.GetVar("tid")%>'/>
        <render:lookup key="ImageSummary" varname="ImageSummary"
      site='<%=ics.GetVar("site")%>' tid='<%=ics.GetVar("tid")%>'/>
        <render:calltemplate tname='<%=ics.GetVar("ImageSummary")%>'
              site='<%=ics.GetVar("site")%>'
              tid='<%=ics.GetVar("tid")%>'
              slotname="ProductSummaryImage"
              c='<%=ics.GetVar("ImageType")%>'
              cid='<%=ics.GetVar("ImageID")%>'
              ttype="Template">
            <render:argument name="p" value='<%=ics.GetVar("p")%>'/>


<%--        <render:argument name="imgnum"
    value='<%=ics.GetVar("imgnum")%>'/> --%>

        </render:calltemplate>
    </ics:then>
```

```
    </ics:if>

<%-- We want to render a link to the item.

    The headline will serve as the link text, so we just need to come
    up with the URL. URLs are generated using the
    render:gettemplateurl tag. The tag needs a variety of parameters
    but basically it needs to know the layout page name and the
    uncached wrapper's page name, plus information about the asset to
    be rendered by the link.

    An alternative to creating the link inline here would be to call
    the Link template.

    Tradeoffs to consider:  Using the link template reuses code.
    Using an inline link simlifies the control flow. --%>

    <render:lookup site='<%=ics.GetVar("site")%>' varname="LayoutVar"
     key="Layout" tid='<%=ics.GetVar("tid")%>'/>
    <render:lookup site='<%=ics.GetVar("site")%>' varname="WrapperVar"
     key="Wrapper" tid='<%=ics.GetVar("tid")%>' match=":x"/>
    <render:gettemplateurl
            outstr="aUrl"
            tid='<%=ics.GetVar("tid")%>'
            slotname="Product_CLinkOnSummaryPage"
            site='<%=ics.GetVar("site")%>'
            c='<%=ics.GetVar("c")%>'
            cid='<%=ics.GetVar("cid")%>'
            tname='<%=ics.GetVar("LayoutVar")%>'
            wrapperpage='<%=ics.GetVar("WrapperVar")%>' >
       <render:argument name="p" value='<%=ics.GetVar("p")%>'/>
    </render:gettemplateurl>
    <h4><a href='<string:stream variable="aUrl"/>'><string:stream
     list='<%="product:"+ics.GetVar("NameAttrName")%>' column="value"
     /></a></h4>

    <ics:if
     condition='<%=ics.GetList("product:"+ics.GetVar("ShortDescriptio
     n")) != null &&
     ics.GetList("product:"+ics.GetVar("ShortDescription")).hasData()
     %>'>
    <ics:then>
       <p class="abstract"><string:stream
     list='<%="product:"+ics.GetVar("ShortDescription")%>'
     column="value"/></p>
    </ics:then>
    </ics:if>
```

```
<%-- return the price and any discounts or promotions --%>

    <ics:if
     condition='<%=ics.GetList("product:"+ics.GetVar("PriceAttrName")
     ) != null%>'>
    <ics:then>
       <ics:listget
     listname='<%="product:"+ics.GetVar("PriceAttrName")%>'
     fieldname="value" output="Price"/>
       <div class="ProductPrice">
          <currency:create name="currencyobj"/>
          <currency:getcurrency name="currencyobj"
     value='<%=ics.GetVar("Price")%>' varname="currencyvalue"/>
          <p>Price Per Unit: $<string:stream variable="currencyvalue"/
     ></p>
       </div>
    </ics:then>
    </ics:if>
    </div>

    <%-- ProductSummary --%>

    </cs:ftcs>
```

Media_C/FSII Summary

```
cache criteria= c,cid,p,rendermode,site,sitepfx,ft_ss

<%@ taglib prefix="cs" uri="futuretense_cs/ftcs1_0.tld"
%><%@ taglib prefix="ics" uri="futuretense_cs/ics.tld"
%><%@ taglib prefix="render" uri="futuretense_cs/render.tld"
%><%@ taglib prefix="asset" uri="futuretense_cs/asset.tld"
%><%@ taglib prefix="assetset" uri="futuretense_cs/assetset.tld"
%><%@ taglib prefix="insite" uri="futuretense_cs/insite.tld"
%><%@ taglib prefix="satellite" uri="futuretense_cs/satellite.tld"
%><%@ taglib prefix="string" uri="futuretense_cs/string.tld"
%><%@ taglib prefix="blobservice" uri="futuretense_cs/
 blobservice.tld"
%><%@ taglib prefix="commercecontext" uri="futuretense_cs/
 commercecontext.tld"%>

<%-- The cs:ftcs tag creates an ICS object which provides access to
     Content Server functionality.  The tag also buffers output for
     caching. It is required in all Content Server JSPs. --%>

     <cs:ftcs>

<%-- The template that contains this JSP file needs to record itself
     as existing on this pagelet when the pagelet is rendered.  The
     render:logdep tag exists for this purpose.

     Dependency tracking is done so that cached pages can be identified
     as invalid when content on them changes.  Upon publish of a given
     asset, all pages that have been rendered that use the given asset
     will be flushed and then subsequently regenerated. --%>

     <ics:if
      condition='<%=ics.GetVar("tid")!=null%>'><ics:then><render:logde
      p cid='<%=ics.GetVar("tid")%>' c="Template"/></ics:then></
      ics:if>

<%-- Several variables are passed into every template.  There is
     usually a variable "c" which is the asset type of the asset being
     rendered. "cid" is the id of the asset being rendered.  "site" is
     the name of the current site and "sitepfx" is a short prefix name
     for the given site.

     All of these variables are required to render assets.

     First, create an asset set that contains only the single asset we
     want to render. --%>
```

```
      <assetset:setasset name="MediaSet" type='<%=ics.GetVar("c")%>'
        id='<%=ics.GetVar("cid")%>' />

<%-- Now that we've created a set (of one asset in this case) that we
      want to render, we can now retrieve the attributes belonging to
      that set.

      All attributes can be retrieved using
      assetset:getattributevalue(s), but if we know in advance that we
      are going to retrieve more than one attribute then we should
      instead use the assetset:getmultiplevalues tag, because it is much
      more efficient than making multiple calls to
      assetset:getattributevalue(s). In particular, there is a lot less
      database access with a single assetset:getmultiplevalues call.

      Text and blob attributes, however, cannot be retrieved using the
      assetset:getmultiplevalues tag, and they must be retrieved
      individually.

      Attribute names can change when a site is replicated, so we have
      to look up the actual attribute name in a database table
      maintained by the template asset. We assigned an arbitrary key to
      the name of the attribute (that co-incidentally correpsonds to the
      name of the attribute in the original site). The render:lookup tag
      is used to look up asset references.

      First, look up the attribute type name --%>

      <render:lookup key="MediaAttrType" varname="MediaAttrType"
        site='<%=ics.GetVar("site")%>' tid='<%=ics.GetVar("tid")%>'/>

<%-- Next, look up each of the attributes we're going to render --%>

      <render:lookup key="ThumbnailFile" match=":x"
        varname="ThumbnailFile" site='<%=ics.GetVar("site")%>'
        tid='<%=ics.GetVar("tid")%>'/>
      <render:lookup key="ImageMimeTypeAttrName" match=":x"
        varname="ImageMimeTypeAttrName" site='<%=ics.GetVar("site")%>'
        tid='<%=ics.GetVar("tid")%>'/>
      <render:lookup key="ImageWidthAttrName" match=":x"
        varname="ImageWidthAttrName" site='<%=ics.GetVar("site")%>'
        tid='<%=ics.GetVar("tid")%>'/>
      <render:lookup key="ImageHeightAttrName" match=":x"
        varname="ImageHeightAttrName" site='<%=ics.GetVar("site")%>'
        tid='<%=ics.GetVar("tid")%>'/>
      <render:lookup key="AltTextAttrName" match=":x"
        varname="AltTextAttrName" site='<%=ics.GetVar("site")%>'
        tid='<%=ics.GetVar("tid")%>'/>
```

```
<%-- Now get the values from the repository --%>

    <assetset:getmultiplevalues name="MediaSet" prefix="media"
     immediateonly="false" byasset="false">
       <assetset:sortlistentry
     attributetypename='<%=ics.GetVar("MediaAttrType")%>'
     attributename='<%=ics.GetVar("ImageMimeTypeAttrName")%>'/>
       <assetset:sortlistentry
     attributetypename='<%=ics.GetVar("MediaAttrType")%>'
     attributename='<%=ics.GetVar("ImageWidthAttrName")%>'/>
       <assetset:sortlistentry
     attributetypename='<%=ics.GetVar("MediaAttrType")%>'
     attributename='<%=ics.GetVar("ImageHeightAttrName")%>'/>
       <assetset:sortlistentry
     attributetypename='<%=ics.GetVar("MediaAttrType")%>'
     attributename='<%=ics.GetVar("AltTextAttrName")%>'/>
    </assetset:getmultiplevalues>
    <assetset:getattributevalues name="MediaSet"
     attribute='<%=ics.GetVar("ThumbnailFile")%>' listvarname="File"
     typename='<%=ics.GetVar("MediaAttrType")%>' />

<%-- Now render the content.

    Be sure to check to see if optional fields actually have data
    specified. Required fields must have data and the UI enforces it,
    so it is not necessary to check --%>

    <blobservice:gettablename varname="imgTable"/>
    <blobservice:getidcolumn varname="imgCol"/>
    <blobservice:geturlcolumn varname="imgUrlCol"/>

<%-- Note that we set cachecontrol to never, which means never expire.
    We do not want Satellite Server to have the blobs time out; the
    cache will be actively managed by CacheManager.  --%>

    <ics:if condition='<%=ics.GetList("File") != null &&
     ics.GetList("File").hasData()%>'>
    <ics:then>


<%-- <img src='<%="jsp/FS2/FS2_prodcuts_page_files/
    BlobServer_00"+ics.GetVar("imgnum")+".gif"%>' class="ImageSummary"
    alt="Content Server Image">  --%>

       <satellite:blob
                blobtable='<%=ics.GetVar("imgTable")%>'
                blobkey='<%=ics.GetVar("imgCol")%>'
```

```
                blobwhere='<%=
        ics.GetList("File").getValue("value")%>'
                blobcol='<%=ics.GetVar("imgUrlCol")%>'
                cachecontrol="never"
                outstring="imgurl">
            <ics:if
        condition='<%=ics.GetList("media"+ics.GetVar("ImageMimeTypeAttrN
        ame")) != null%>'><ics:then>
                <satellite:argument name='blobheader' value='<%=
        ics.GetList("media"+ics.GetVar("ImageMimeTypeAttrName")).getValu
        e("value")%>' />
            </ics:then></ics:if>
          </satellite:blob>

<%-- --%>

<%-- Stream the image tag.  String:stream is used instead of
     ics:getvar to handle proper escaping for xml-compliance. --%>


     <img src="<string:stream variable="imgurl"/>" class="ImageSummary"

 <%-- Some attributes may not be specified. Add them to the tag if
     they have been --%>

            <ics:if
        condition='<%=ics.GetList("media"+ics.GetVar("ImageWidthAttrName
        ")) != null%>'><ics:then>
                width="<string:stream
        list='<%="media"+ics.GetVar("ImageWidthAttrName")%>'
        column="value"/>"
            </ics:then></ics:if>
            <ics:if
        condition='<%=ics.GetList("media"+ics.GetVar("ImageHeightAttrNam
        e")) != null%>'><ics:then>
                height="<string:stream
        list='<%="media"+ics.GetVar("ImageHeightAttrName")%>'
        column="value"/>"
            </ics:then></ics:if>
            <ics:if
        condition='<%=ics.GetList("media"+ics.GetVar("AltTextAttrName"))
        != null%>'><ics:then>
                alt="<string:stream
        list='<%="media"+ics.GetVar("AltTextAttrName")%>'
        column="value"/>"
            </ics:then><ics:else>
<%-- --%>
<%-- alt is a required attribute - it should be specified --%>
```

```
              alt="Content Server Image"
           </ics:else></ics:if>
        />
<%-- --%>
<%-- yes, we finally get to close the img tag! Phew! --%>

     </ics:then>
     </ics:if>
     </cs:ftcs>
```