Content Server

Version: 6.3

Editorial Systems Sizing Guide

Document Revision Date: Jul. 21, 2006

FatWire®
SOFTWARE

Content Server 6.3 Editorial Systems Sizing Guide
Document Revision Date: Jul. 21, 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

Part 2. System Specifications

Chapter 1

Abstract

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

- Platform configuration
- Tuning parameters
- Data model
- Page caching
- Network connectivity of internal systems
- Internet connectivity

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

Scope

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

- Distribution of editorial tasks
- Number of attributes for a given asset
- Application server performance
- Clustering

We tested the systems that are diagrammed in "System Configurations," on page 8. We also designed a single-level asset model, using the flex family. 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 editorial system. For asset design specifications, see Part 2, "System Specifications."

Note

Tuning parameters for the systems in this guide are available in a separate publication. Please contact your sales representative for a copy.

Key Findings

- **Number of Users**

  A single-CS system can support up to 35 simultaneous users at 2-second response time and up to 50 simultaneous users at 4-second response time under our test conditions. That is, the system must be running on WebLogic 8.1 SP5 and executing our editorial scenarios (listed below) on the flex asset model described in Chapter 6, "Asset Design."
- **Distribution of editorial tasks**

  Response times are unaffected by changes to the distribution of editorial tasks. Four scenarios were tested:
  - **Scenario 1**: 60% Inspect, 40% Search
  - **Scenario 2**: 40% Inspect, 40% Search, 20% Edit
  - **Scenario 3**: 30% Inspect, 40% Search, 20% Edit, 10% Create/Delete
  - **Scenario 4**: 30% Inspect, 30% Search, 20% Edit, 10% Create/Delete, 10% Approve
- **Number of attributes**

  Increasing the number of attributes up to 30 for a given asset does not affect performance. (We tested for a maximum of 30 attributes, none of which are inherited.)
- **Application server**

  The best performer in the editorial system is WebSphere at 127% the efficiency of WebLogic, followed by Sun JES at 118%, and then Resin at 105%. The remaining three servers JBoss, Tomcat, and WebLogic performed nearly identically. (All performance was tested relative to WebLogic.)

- **Clustering**

  Clustering an editorial system leads to a gain of 165% for a 2-member CS system and 243% for a 3-member system, regardless of how the editorial tasks are distributed.

Base Conditions

We set several conditions for performance testing:

- We tested editorial operations on a development system, assuming it should not differ from an editorial system, as none of our scenarios involved either delivery or development while editorial performance was being tested.
- We based our tests on a custom asset model, which we created in FirstSiteII. The asset model is a simple one consisting of a single level of assets that have no parents and therefore do not inherit attributes. We assigned up to 30 attributes to the assets. For information about the asset model, see Chapter 6, "Asset Design."
- 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 during any given test (distribution of editorial tasks, clustering, 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 shown below, using the software and hardware listed on page 9.

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. For test results, see:

- Chapter 2, "Editorial Test"
- Chapter 3, "Number of Attributes"
- Chapter 4, "Application Servers"

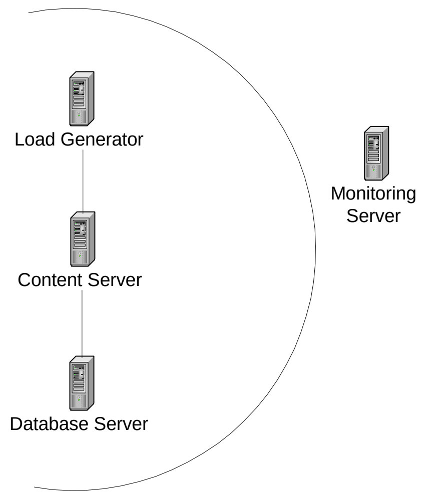
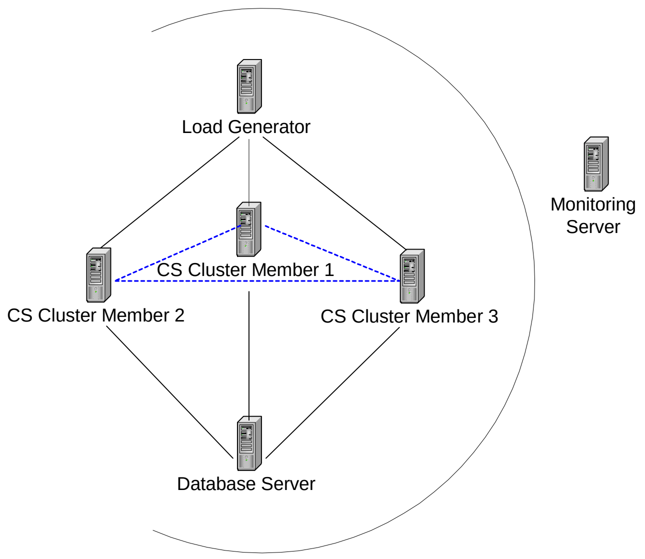


Figure 2: Horizontally Clustered Content Servers

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 in various editorial scenarios (described in Chapter 2, "Editorial Test").

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 and summarizes computation methods for the Content Server systems shown in "System Configurations," on page 8.

This part contains the following chapters:

Chapter 2

Editorial Test

This test reports system performance in response to users executing various editorial tasks (Inspect, Search, Create/Delete, and Approve). Charts show the absolute load that can be supported at response times of 2 and 4 seconds for a range of editorial tasks.

This chapter contains the following sections:

Results

An editorial task, when performed in combination with other tasks, is not affected by the other tasks. Supported load remains constant, within experimental error, at 2- and 4-second response times. For example, in Chart 1, the **Inspect** operation allows for 35 to 37 users to work concurrently at 2-second response time, regardless of how many other editorial tasks are being performed.

Editorial Scenarios

To determine how Content Server performs in response to simultaneously executed editorial tasks, we created four scenarios using a mix of tasks, performed in random order:

- **Scenario 1**: 60% Inspect, 40% Search
- **Scenario 2**: 40% Inspect, 40% Search, 20% Edit
- **Scenario 3**: 30% Inspect, 40% Search, 20% Edit, 10% Create/Delete[1]
- **Scenario 4**: 30% Inspect, 30% Search, 20% Edit, 10% Create/Delete, 10% Approve

Each scenario begins with a "Log in to Site" task. These scenarios were scripted and a load of between 0 and 50 concurrent users was executed against Content Server's interface. The response time for each transaction[2] was then monitored, and the results at 2 and 4 seconds were plotted to create the charts in this chapter.

Note

The scenarios above are used throughout this guide to test the performance of single-CS systems, horizontally clustered systems, supported application servers, and changes to a flex asset's number of attributes (none of which are inherited).

Performance Charts

Charts 1 and 2 show the absolute load that Content Server can support at 2-second and 4-second response times for a given editorial task when it is being performed simultaneously with other tasks. The charts were determined for:

- The single-CS configuration shown in Figure 1, on page 8
- The asset model described in "Assets for Editorial, App Server, and Clustering Tests," on page 32. (For detailed editorial test conditions, see page 33.)

1. The scripts were developed to ensure that an equal number of Create and Delete operations occurred, allowing the system to be restored to its original number of assets once testing was complete. Thus, every created asset was also deleted by the same user in the same execution of the script.
2. A transaction is a completed editorial operation. Examples of transactions include "login_into_site" and "perform_search," where "login_into_site" measured the time it takes to load the initial interface page presented to a user (including the applet), and "perform_search" measured the time it takes to render a search result page.

Chart 1.
Number of users *vs.* editorial task at 2-second response time.

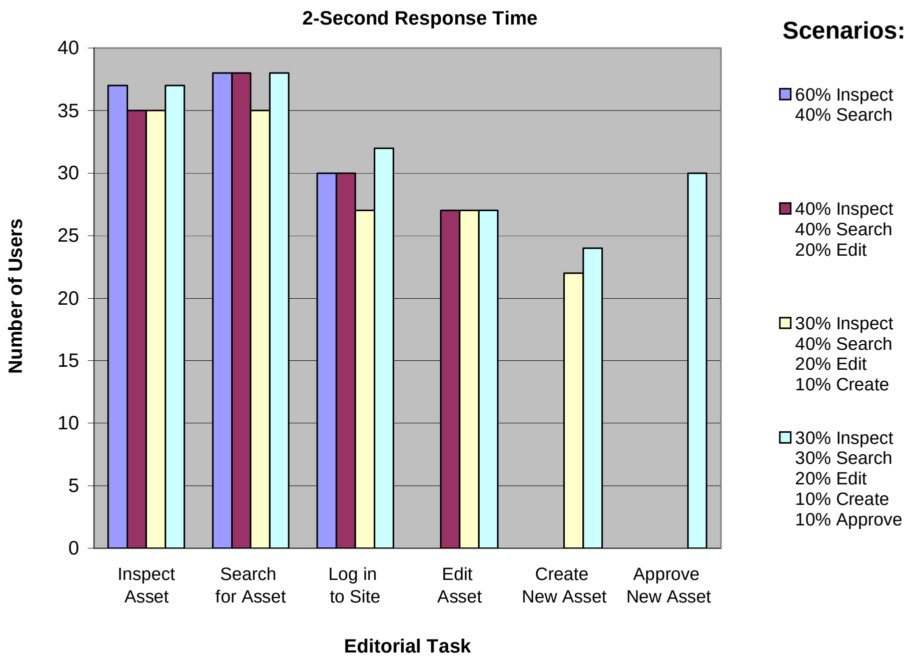


Chart 2.
Number of users *vs.* editorial task at 4-second response time.

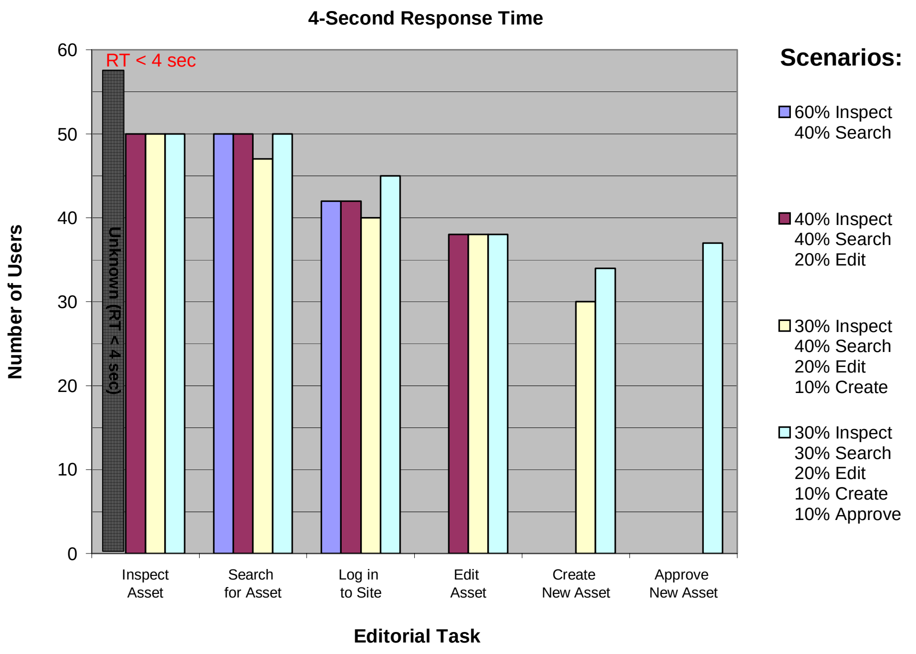


Chart 1 displays the load at which a given editorial task takes 2 seconds to complete.

Chart 2 displays the load at which a given editorial task takes 4 seconds to complete (*except for* the "Inspect" operation, which took less than 4 seconds to complete as noted in Chart 2).

Notes

- Because not every scenario in Charts 1 and 2 includes every type of editorial task, the number of bars per editorial task varies. For example, only the last scenario contains the “Approve New Asset” operation. Each chart therefore displays a single “Approve New Asset” bar, corresponding to the last scenario (30% Inspect, 30% Search, 20% Edit, 10% Create/Delete, 10% Approve).
- The number of users was varied across editorial tasks in order to keep response times at 2 and 4 seconds.

Chapter 3

Number of Attributes

This test reports scaling of system performance in response to changes in the number of non-inherited attributes that are assigned to a flex asset. Testing was done on a single-CS system.

This chapter contains the following sections:

- Results
- Scaling Chart

Results

For the editorial scenarios shown in Chart 3, changes to a flex asset's non-inherited attributes have no effect on system performance. We tested up to 30 attributes.

Scaling Chart

Chart 3 shows how hits per second scale with the number of non-inherited attributes that are assigned to flex assets. Flex assets were tested under the editorial scenarios that are listed in the legend of Chart 3.

Chart 3.
Scaled hits per second *vs.* number of attributes.

(Scaling factors were determined *within* each set of attributes by normalizing each scenario's peak hits-per-second to the peak hits-per-second in the first scenario.)

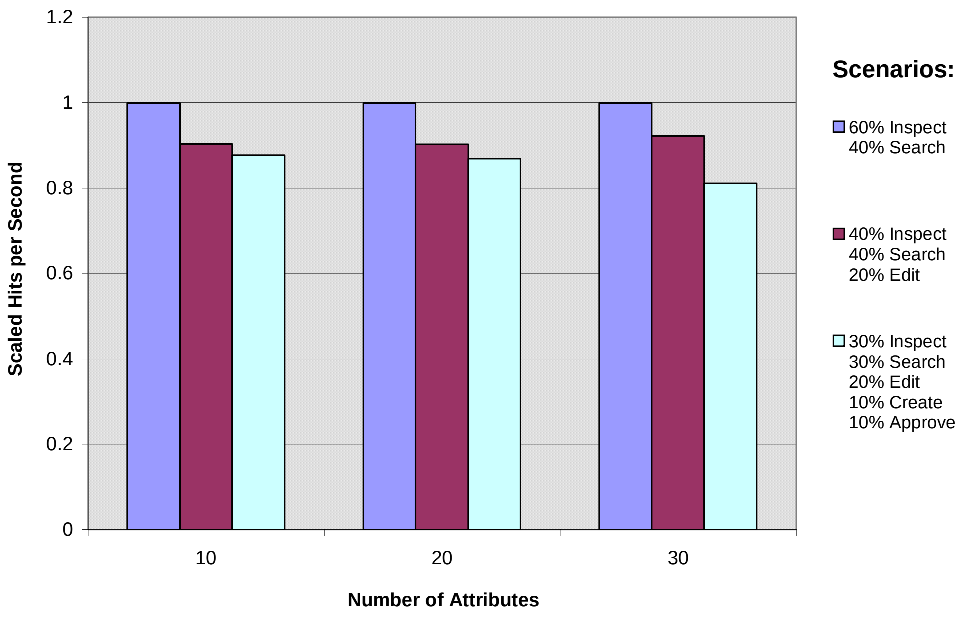


Chart 3 was determined for:

- The single-CS configuration shown in Figure 1, on page 8.
- The asset model described in "Assets for Attribute Testing," on page 32.
  To determine how performance scales with changes to the number of attributes in a flex asset, we created three identical assets with identical data, except that one asset contained 10 attributes; another contained 20 (10 repeated twice); and the last contained 30 (10 repeated three times). Three of the four base scenarios (listed on page 14) were then executed upon these assets to determine how their performances compared to each other.[1]

Hits per second were recorded and scaled as explained in the text to the left of Chart 3.

The chart shows that:

- Adding editorial tasks to a scenario reduces the number of hits per second. (For example, at 10 attributes, the **Edit** task in the middle scenario reduces hits per second to 90% of the hits supported by the first scenario.)

1. Because our editorial test (in Chapter 2) had shown little difference in system performance across scenarios, we chose only three scenarios (at random) for sizing tests.

- The three sets of columns are identical (within experimental error). For a given scenario, changing the number of attributes leaves hits per second unaffected. (For example, at 10 attributes the middle scenario supports 90% of the hits supported by the first scenario. The same holds for 20 and 30 attributes.)

Chapter 4

Application Servers

This test compares the performance of supported application servers in a single-CS system running the CS 6.3 Patch 1 release.

This chapter contains the following sections:

- Results
- Scaling Chart
- Hits per second were recorded and scaled as explained in the text to the left of Chart 4.

Results

All application servers can handle the CS editorial interface, but certain application servers perform better than others. We tested six application servers in a single-CS configuration. Starting with the best performer, they are:

- WebSphere 6.0.1, handling 127% of WebLogic's capacity[1]
- Sun JES Q4 2005, handling 118.5%
- Resin 3.0.19, handling 105%
- Tomcat and JBoss, which performed almost identically to WebLogic

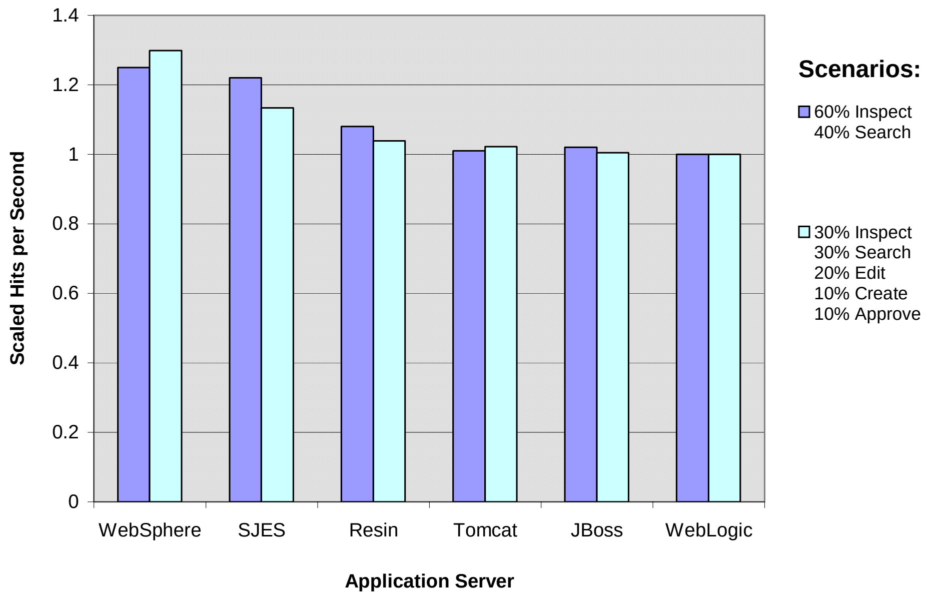
Note that we tested the same six application servers that we tested in delivery and found the results to differ. For information about delivery system tests, see the *Content Server 6.3 Delivery System Sizing Guide*.

Scaling Chart

Chart 4 shows how application servers in the Content Server 6.3 Patch 1 release perform relative to WebLogic in two of our editorial scenarios (listed on page 14).[2]

Chart 4.
Scaled hits per second (relative to WebLogic) *vs.* application server.

(Scaling factors were determined by normalizing each scenario's peak hits-per-second to WebLogic's peak hits-per-second.)



The chart was determined for:

- All application servers running in the single-CS configuration shown in Figure 1, on page 8.

1. WebLogic was again used as the baseline for the comparison, because it is our benchmark application server, used in all other tests.
2. Because our editorial test (in Chapter 2) had shown little difference in system performance across scenarios, we chose only two scenarios for sizing the different application servers. Scenarios were chosen to represent both the simplest and most complex of the scripted operations.

- The asset model described in "Assets for Editorial, App Server, and Clustering Tests," on page 32.

Hits per second were recorded and scaled as explained in the text to the left of Chart 4.

Chapter 5

Horizontal Clustering

This test reports the scaling of system performance with the number of nodes in a horizontally clustered system during editorial testing.

This chapter contains the following sections:

- Results
- Scaling Chart

Results

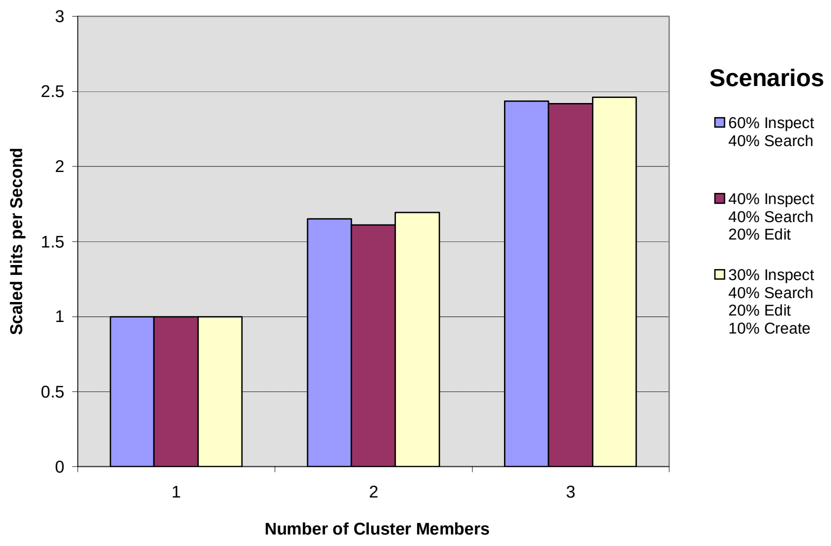
Clustering Content Server results in a viable improvement in the maximum concurrency that can be handled by the CS interface. The gains are significant: 165% (65% increase) for a 2-CS cluster, and 243% (78% increase) for a 3-CS cluster. The gains across clustered configurations are, again, independent of the editorial tasks performed by the users.

Scaling Chart

Chart 5 shows how hits per second scale with the number of CS members in a horizontal cluster. Flex assets were tested under the editorial scenarios shown in the legend of Chart 5.

Chart 5.
Scaled number of hits per second *vs.* number of cluster members.

(Scaling factors were determined by normalizing each scenario's peak hits-per-second to the peak hits-per-second for a single-CS system running the same scenario.)



To determine how Content Server performance scales with the addition of cluster members, we repeated three of the editorial scenarios (listed on page 14)[1] for clusters of 2 and 3 members.[2]

The chart was determined for:

- The configuration shown in Figure 2, on page 8
- The asset model described in "Assets for Editorial, App Server, and Clustering Tests," on page 32

Hits per second were recorded and scaled as explained in the text to the left of Chart 5.

To keep conditions as similar as possible to those of a single-member system, user ramp-up was kept at the rate previously used. However, the number of users was multiplied by the number of cluster members, so that the total number of users at any point was equivalent to the number of users in a single-member test multiplied by the number of

1. The fourth scenario was omitted only because our editorial test had shown little difference in system performance across scenarios.
2. A cluster of 1 member is the same as those previously run. Thus the existing results were used to indicate a single-member cluster.

cluster members.[1] Users were distributed among cluster members in a random manner to simulate real world operations while dealing within the limitations imposed by generating many users from a single source.

1. Since all ramp-ups for the single-member test were in increments of 1, this means that increments of 2 were used for the 2 member test, and increments of 3 were used for the 3 member test.

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

Asset Design

This chapter describes the assets that we designed for our tests.

This chapter contains the following sections:

Assets for Editorial, App Server, and Clustering Tests

In three of our tests (reported in Chapters 2, 4, and 5), we used a single-level data model consisting of flex assets whose attributes are specified in the asset definition, listed below.

• Site:	FirstSite II
• Flex Family:	Product
• Total Attributes:	(10). String00, String01,String02, String03, String04, Int1, Int2, SortNum(Int), Blob1, Asset1
• Parent Definitions:	None
• Parents:	None
• Asset Type:	Product
• Asset Definition:	EditSize1ProdDef This asset definition uses the 10 attributes specified above. To test the edit and other operations, 5,000 assets were preloaded into CS prior to running the test.
• Attribute inheritance:	None

Assets for Attribute Testing

In the attribute test (Chapter 3), we used a single-level data model consisting of flex assets whose attributes are specified in several asset definitions, listed below.

• Site:	FirstSite II
• Flex Family:	Product
• Total Attributes:	(30). String 00, String01, String02, String03, String04, String05, String 06, String07, String08, string09, String10, String11, String12, String13, String14, String15, Int1, Int2, Int3, Int4, Int5, Int6, SortNum(Int), Blob1, Blob2, Blob3, Blob4, Blob5, Asset1, Asset2, Asset3
• Parent Definitions:	None
• Parents:	None
• Asset Type:	Product
• Asset Definitions:	
- EditSize1ProdDef	Specifies 10 attributes: String 00, String01, String02, String03, String04, Int1, Int2, SortNum (Int), Blob1, Asset1
- EditSize2ProdDef	Specifies the following 20 attributes: String 00, String01, String02, String03, String04, String05, String 06, String07, String08, string09, Int1, Int2, Int3, Int4, SortNum, Blob1, Blob2, Blob3, Asset1, Asset2
- EditSize3ProdDef	Specifies the total number of attributes: String 00, String01, String02, String03, String04, String05, String 06, String07, String08, string09, String10, String11, String12, String13,

String14, String15, Int1, Int2, Int3, Int4, Int5, Int6, SortNum, Blob1, Blob2, Blob3, Blob4, Asset1, Asset2, Asset3

To test the edit and other operations, 5000 assets were preloaded into CS for each of the 3 asset types prior to running the test.

- Attribute inheritance: None

Assets in Test Runs

For each asset type, we ensured several conditions:

- Before the start of testing:
  - We created 5,000 assets on the CS editorial system.
  - We also created 100 virtual users. Each user had 50 assets assigned to its Active List. The Active List made assets easily available for Edit, Inspect, and Approve operations.
- During testing:
  - The assets described in this chapter were invoked through editorial operations, performed in random order and in various combinations by the virtual users. The operations are: Create/Delete, Edit, Inspect, Search, and Approve. Their combinations are specified as editorial scenarios on page 14 (and listed in the legends of the scaling charts in this guide). Each scenario begins with a “Log in to Site” operation.
  - Search operations were performed from the **Search** button.
  - Create operations were performed from the **New** button.
  - Every new asset that was created during the test was also deleted at the end of the test by use of the **Delete** link on the asset’s “Inspect” screen. The number of “Create” operations was maintained equal to the number of “Delete” operations. This was done to ensure that at the end of each test run, Content Server had the same number of assets as at the beginning of the run.

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 as follows, to ensure completeness of their transactions:

> In order to simulate an ever increasing number of users on a site and determine how the increases users affect performance, a constant ramp-up was used. Fifty users were ramped up over 150 minutes (2 1/2 hours). At the end of the ramp up, the 50 users were allowed to run for 35 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 2 users every 6 minutes, based on the requirement that a user completes a transaction in less than 120 seconds (even in the worst case), and therefore has the time to complete at least 3 transactions before the load is increased. Thus, within a 6-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