COMPAQ
Technology Brief

January 2000
11LQ-0100A-WWEN

Prepared by ECG Technology Communications Group

Compaq Computer Corporation

Contents

Compaq TaskSmart C-Series servers obtain greater availability with Foundry Networks ServerIron Switch

Abstract: As companies grow more dependent on the Internet for all types of e-transactions, it is important that enterprise information technology departments, Internet Service Providers, institutions, and small businesses supply greater levels of availability to their customers. The purpose of this paper is to describe how to create a cluster using two TaskSmart C-Series caching proxy servers in client acceleration mode and a Foundry Networks ServerIron. The Web Polygraph Benchmark will then be used to verify the high availability and performance scalability achieved.

Notice

Compaq TaskSmart C-Series servers obtain greater availability with Foundry Networks ServerIron Switch
Technology Brief prepared by ECG Technology Communications Group

First Edition (January 2000)
Document Number 11LQ-0100A-WWEN

Introduction

As companies and individuals become dependent on the Internet and e-transactions, it becomes necessary to add capacity and fault tolerance. In many situations, additional capacity becomes necessary, and the upgrade process is filled with issues like poor scalability, complex configurations, and even downtime.

This paper describes how to create a load-balancing cluster using two Compaq TaskSmart C-Series servers and a Foundry Networks ServerIron switch. The configuration will be transparent to the users, will be fault tolerant, and will obtain a greater than 95% performance scalability factor. The Web Polygraph Benchmark will be used to verify high availability and performance scalability.

Equipment Requirements

System under test

2 – Compaq TaskSmart C-2000R

1 – Foundry ServerIron (Model FBS24 Part No. FBSLB24)

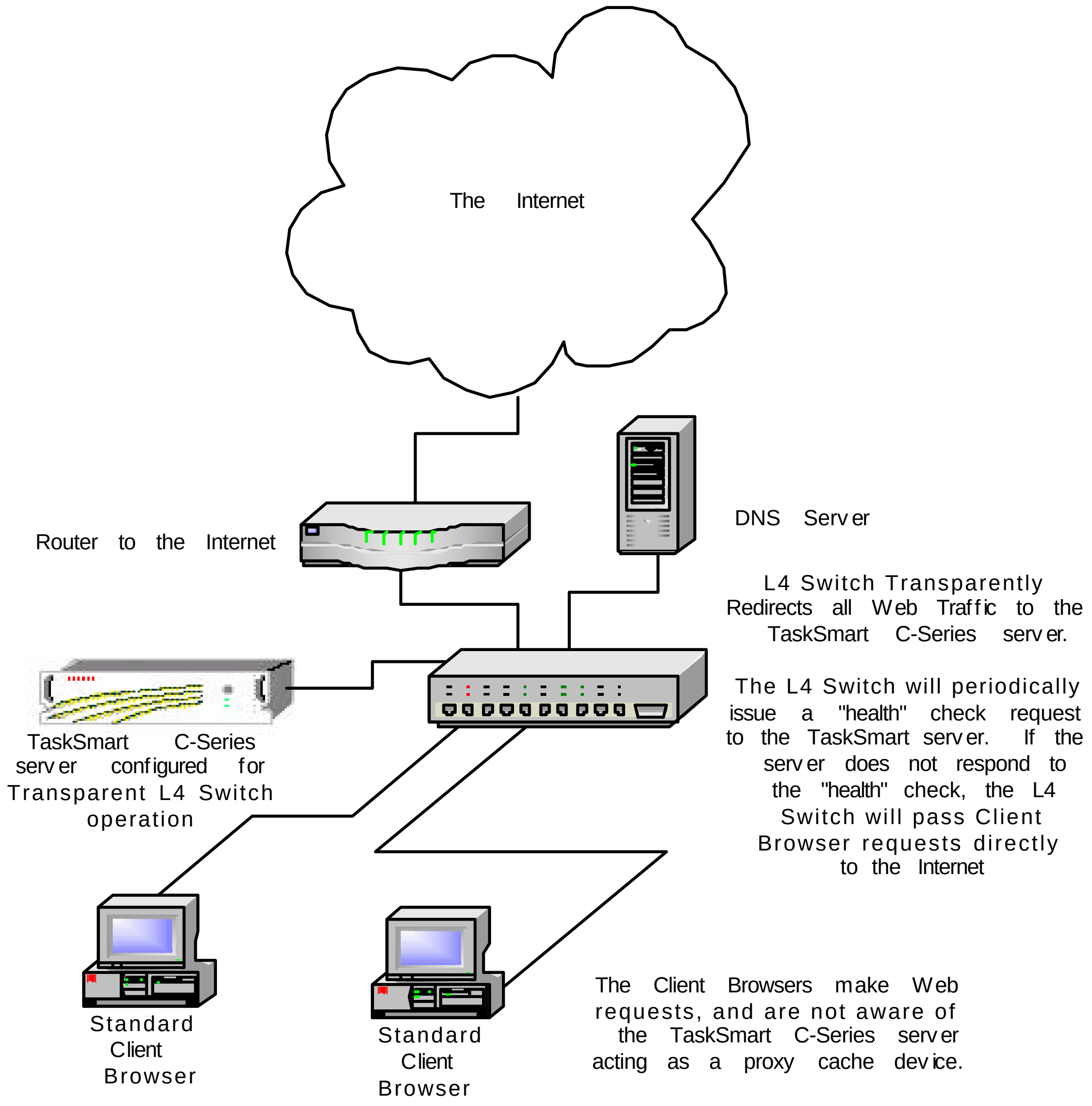
8 – Web Polygraph client/server pairs capable of 300 requests/second each

(Refer to White Paper "Deploying Web Polygraph to Benchmark and Qualify Proxy Cache Compaq TaskSmart C-Series Servers", Document No. 1176-1099A-WWEN at http://www.compaq.com/support/techpubs/whitepapers/WhitePapers_Performance.html)

1 – Web Polygraph controller with one network adapter

TaskSmart C-Series server Transparent L4 System Configuration

The TaskSmart C-Series server can be used with standard L4 (Layer 4) switches to eliminate the need for any special configuration of the client browsers. The L4 switch redirects all Web or HTTP requests to the TaskSmart C-Series server. Prior to redirecting any Web requests, the L4 switch performs a "health" check on the TaskSmart server to assure it is up and running. If the "health" check fails, the L4 switch will just forward client requests directly to the Internet, bypassing the TaskSmart completely. Other features in the L4 switches allow for load balancing between multiple TaskSmart servers and limiting total number of connections. Shown below is a simple network diagram using one TaskSmart server in Transparent Client acceleration mode using an external L4 switch:


The Internet
Router to the Internet
DNS Server
L4 Switch Transparently Redirects all Web Traffic to the TaskSmart C-Series server.
The L4 Switch will periodically issue a "health" check request to the TaskSmart server. If the server does not respond to the "health" check, the L4 Switch will pass Client Browser requests directly to the Internet
TaskSmart C-Series server configured for Transparent L4 Switch operation
Standard Client Browser
Standard Client Browser
The Client Browsers make Web requests, and are not aware of the TaskSmart C-Series server acting as a proxy cache device.

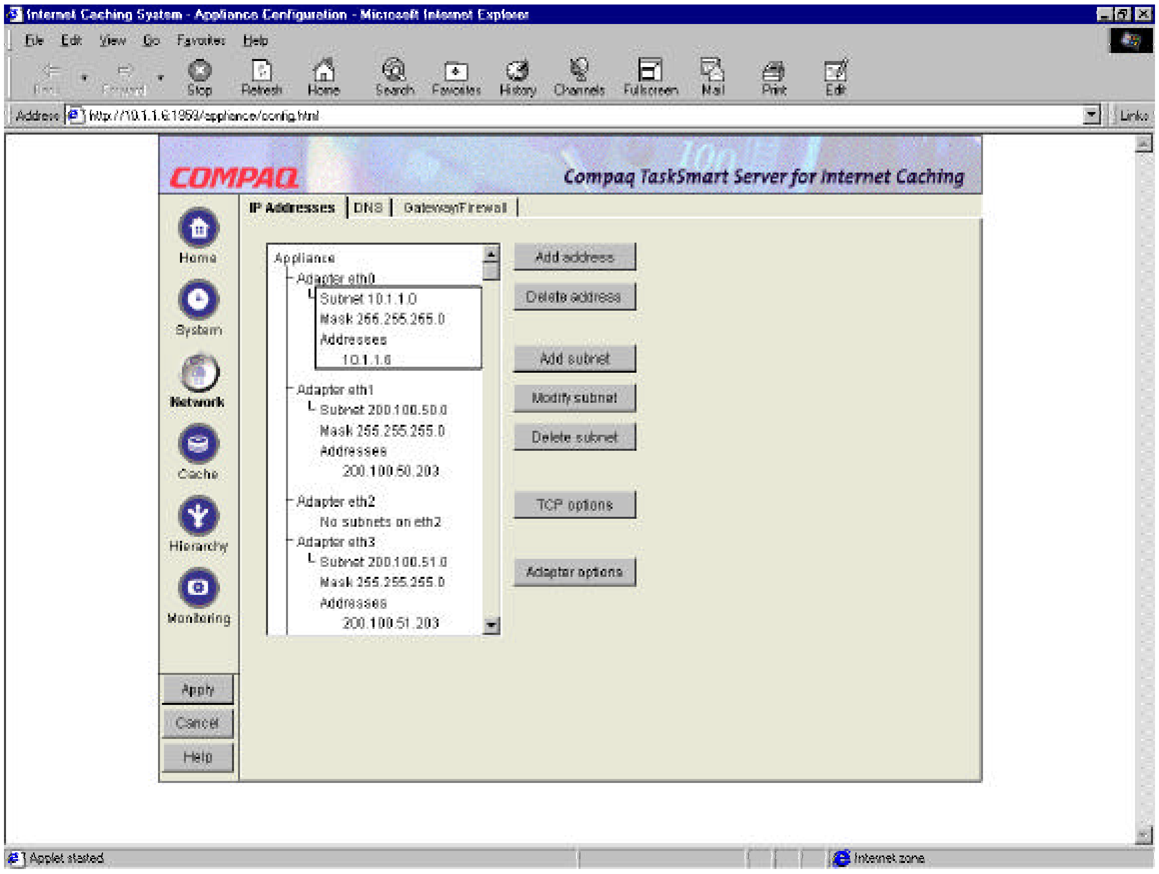
In order to configure the system for Transparent Proxy L4 operation, both the L4 switch and the TaskSmart server must be configured. Although the terminology and methods vary from one L4 switch manufacturer to another, the basic configuration method is to create a "Cache Group" and add the IP address of the actual TaskSmart server where the switch is to redirect to. Once the "Cache Group" is created, "IP Policies" or redirection filters are applied to specific interface ports on the switch. For situations where load balancing will be used the "Cache Group" will contain the IP addresses for multiple TaskSmart servers. For testing the two TaskSmart servers with two networks it was necessary to create one "Cache Group" for each network. See Appendix A for details about "Cache Groups" and the redirection policies for this configuration.

Once the L4 switch is set up to do Transparent L4 redirection, the TaskSmart server must be configured to accept redirected requests. Configuring the TaskSmart server for redirection requires three steps:

Step 1. Set up the IP address on the TaskSmart server to match the ones described in the "Cache Group" in the L4 switch and Apply the changes. Below are two screen shots showing the configuration of the IP addresses on the TaskSmart server.

First TaskSmart server network configuration:

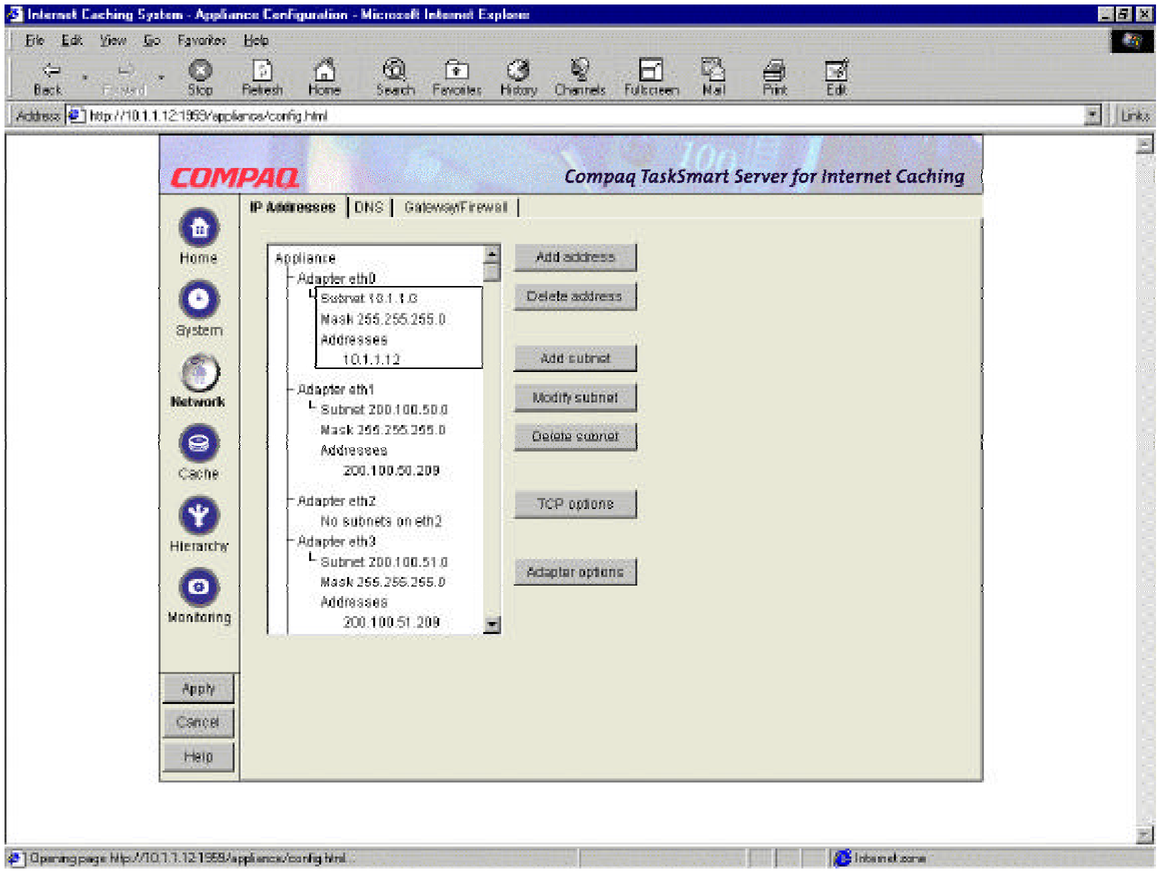
eth1 = 200.100.50.203
eth3 = 200.100.51.203


Internet Caching System - Appliance Configuration - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print Edit
Address http://10.1.1.6:1959/appliance/config.html
Links
COMPAQ
Compaq TaskSmart Server for Internet Caching
IP Addresses DNS Gateway/Firewall
Home System Network Cache Hierarchy Monitoring
Add address
Delete address
Add subnet
Modify subnet
Delete subnet
TCP options
Adapter options
Apply Cancel Help
Applet started
Internet zone

```
Appliance
 Adapter eth0
   Subnet 10.1.1.0
   Mask 255.255.255.0
   Addresses
     10.1.1.6
 Adapter eth1
   Subnet 200.100.50.0
   Mask 255.255.255.0
   Addresses
     200.100.50.203
 Adapter eth2
   No subnets on eth2
 Adapter eth3
   Subnet 200.100.51.0
   Mask 255.255.255.0
   Addresses
     200.100.51.203
```

Second TaskSmart server network configuration:

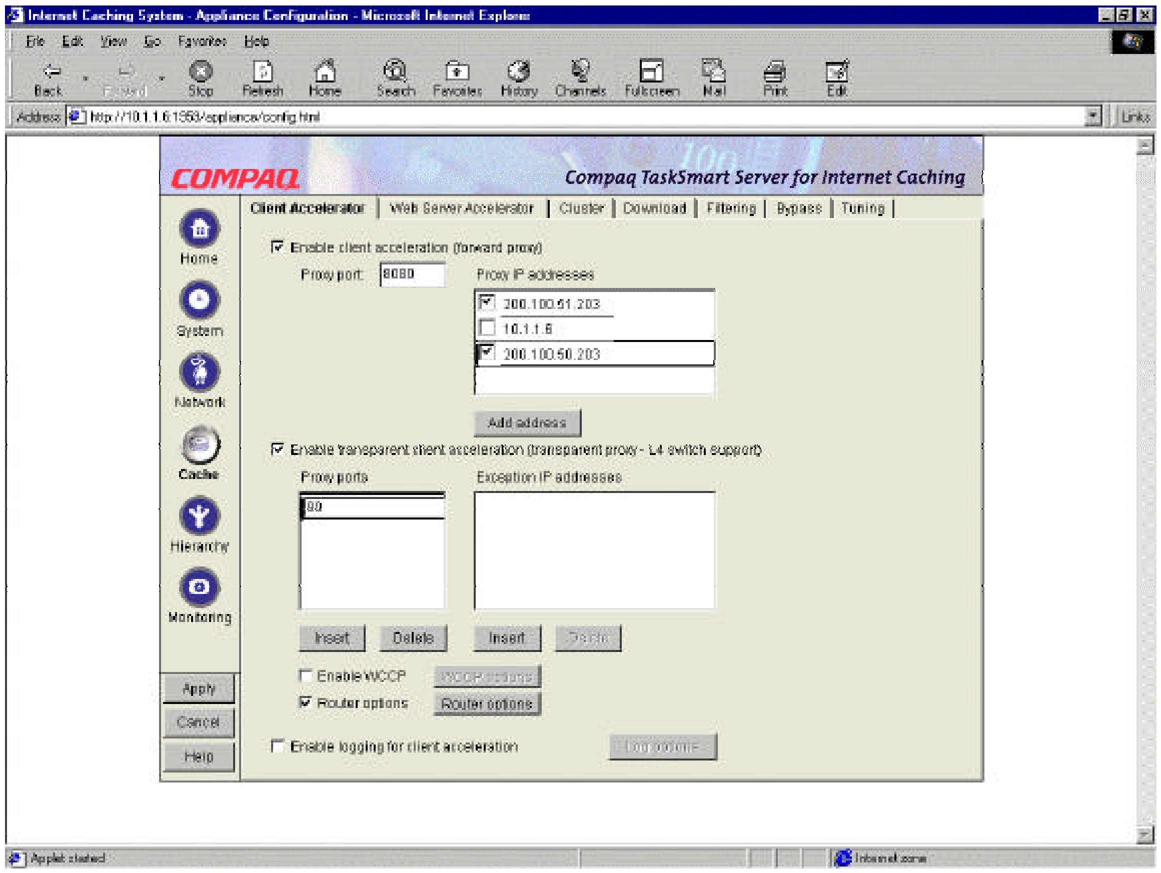
eth1 = 200.100.50.209
eth3 = 200.100.51.209


Internet Caching System - Appliance Configuration - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print Edit
Address http://10.1.1.12:1959/appliance/config.html
COMPAQ
Compaq TaskSmart Server for Internet Caching
IP Addresses | DNS | Gateway/Firewall
Home System Network Cache Hierarchy Monitoring
Apply Cancel Help
Add address Delete address Add subnet Modify subnet Delete subnet TCP options Adapter options
Opening page http://10.1.1.12:1959/appliance/config.html
Internet zone

```
Appliance
  Adapter eth0
    Subnet 10.1.1.0
    Mask 255.255.255.0
    Addresses
      10.1.1.12
  Adapter eth1
    Subnet 200.100.50.0
    Mask 255.255.255.0
    Addresses
      200.100.50.209
  Adapter eth2
    No subnets on eth2
  Adapter eth3
    Subnet 200.100.51.0
    Mask 255.255.255.0
    Addresses
      200.100.51.209
```

Step 2. Enable Transparent Client Acceleration (Transparent Proxy L4 switch support) by clicking on the checkbox. Also, enable the two network IP addresses for forward proxy use by clicking on the checkboxes. Below are two screen shots showing the configuration of the Transparent Client Accelerator and Forward Proxy addresses on the TaskSmart server. Please note the Router options checkbox is also enabled when Transparent Client Acceleration is enabled.

First TaskSmart server Transparent Client Acceleration configuration:


Internet Caching System - Appliance Configuration - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print Edit
Address
Links
COMPAQ
Compaq TaskSmart Server for Internet Caching
Client Accelerator | Web Server Accelerator | Cluster | Download | Filtering | Bypass | Tuning
Home
System
Network
Cache
Hierarchy
Monitoring
Apply
Cancel
Help
Add address
Insert
Delete
Insert
Delete
WCCP options
Router options
Log options
Applet started
Internet zone

Address: http://10.1.1.6:1953/appliance/config.html

[✓] Enable client acceleration (forward proxy)

Proxy port: 8080

Proxy IP addresses:
- [✓] 200.100.51.203
- [ ] 10.1.1.6
- [✓] 200.100.50.203

[✓] Enable transparent client acceleration (transparent proxy - L4 switch support)

Proxy ports: 80

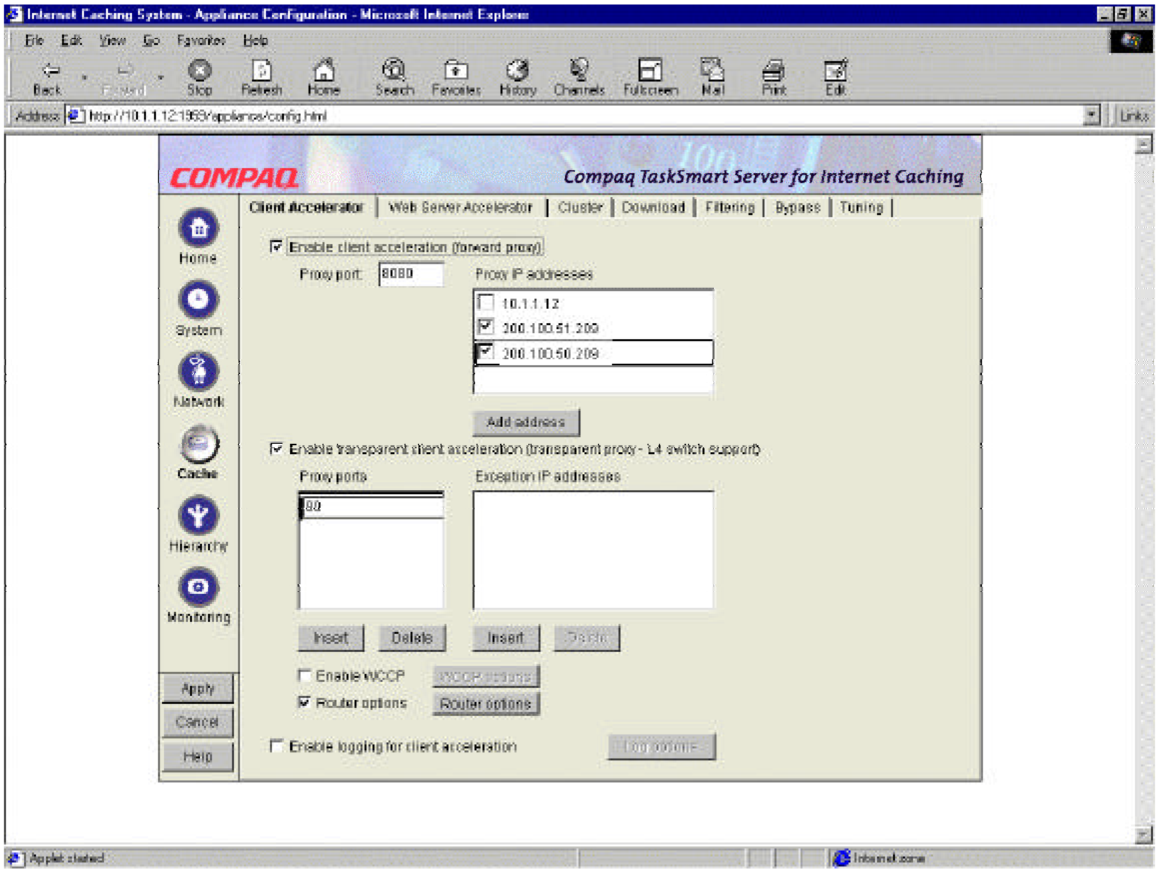
Exception IP addresses:

[ ] Enable WCCP

[✓] Router options

[ ] Enable logging for client acceleration

Second TaskSmart server Transparent Client Acceleration configuration:


Internet Caching System - Appliance Configuration - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print Edit
Links
COMPAQ
Compaq TaskSmart Server for Internet Caching
Client Accelerator | Web Server Accelerator | Cluster | Download | Filtering | Bypass | Tuning
Home
System
Network
Cache
Hierarchy
Monitoring
Apply
Cancel
Help
Add address
Insert
Delete
Insert
Delete
WCCP options
Router options
Log options
Applet started
Internet zone

Address: http://10.1.1.12:1959/appliances/config.html

☑ Enable client acceleration (forward proxy)

Proxy port: 8080

Proxy IP addresses
- ☐ 10.1.1.12
- ☑ 200.100.51.209
- ☑ 200.100.50.209

☑ Enable transparent client acceleration (transparent proxy - L4 switch support)

Proxy ports: 80

Exception IP addresses

☐ Enable WCCP

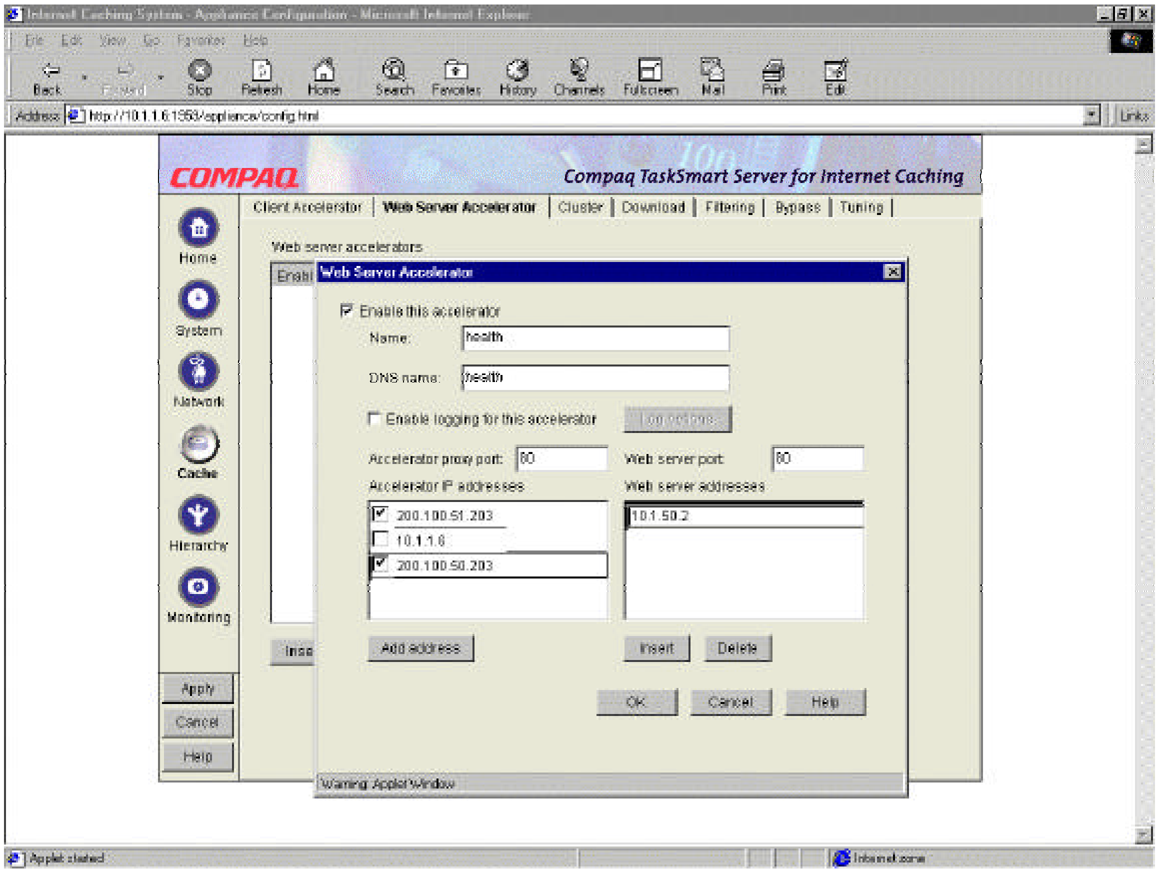
☑ Router options

☐ Enable logging for client acceleration

Step 3. Create a “fake” Web Server Accelerator on both TaskSmart servers. The purpose of the “Fake” Web server accelerator is to respond when the L4 switch issues a “health” check request. An L4 switch “health” check request is usually an HTTP GET request directly to the TaskSmart server. The purpose is to determine if the TaskSmart server is capable of accepting redirected Web traffic. Without the “fake” Web Server Accelerator the TaskSmart server cannot respond because it is expecting only redirected Web traffic. Effectively, the Web Server Accelerator tricks the L4 switch by responding with a “404 server unreachable” error.

The “fake” Web Server Accelerator should be named HEALTH with the DNS name of HEALTH, and a bogus web server address for a Web server addresses. In this example a bogus or non-existent Web server address is used. If the TaskSmart server is already being used as Web Server Accelerator, this step is not necessary. Below are two screen shots showing the configuration of the “fake” Web Server Accelerators for the TaskSmart server.

First TaskSmart server “fake” Web Server Accelerator configuration:



Address: http://10.1.1.6:1953/appliance/config.html

Web server accelerators

Web Server Accelerator

[✓] Enable this accelerator

Name: health

DNS name: health

[ ] Enable logging for this accelerator

Accelerator proxy port: 80

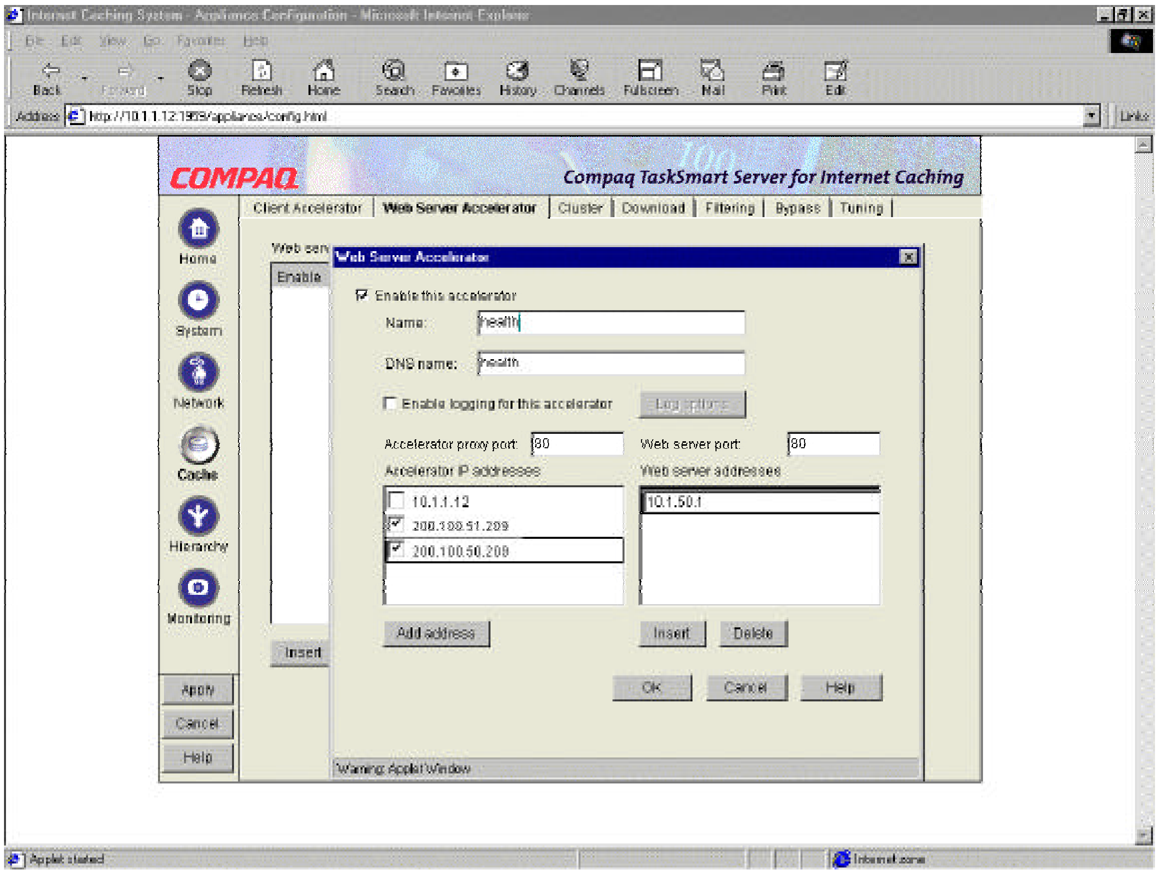
Web server port: 80

Accelerator IP addresses:
- [✓] 200.100.51.203
- [ ] 10.1.1.6
- [✓] 200.100.50.203

Web server addresses:
- 10.1.50.2

Warning: Applet Window

Second TaskSmart server "fake" Web Server Accelerator configuration:


Internet Caching System - Appliance Configuration - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print Edit
Address http://10.1.1.12:1959/appliance/config.html
Links
COMPAQ
Compaq TaskSmart Server for Internet Caching
Client Accelerator | Web Server Accelerator | Cluster | Download | Filtering | Bypass | Tuning
Home
System
Network
Cache
Hierarchy
Monitoring
Apply
Cancel
Help
Web serv
Enable
Insert
Web Server Accelerator
Log options
Add address
Insert
Delete
OK
Cancel
Help
Warning: Applet Window
Applet started
Internet zone

☑ Enable this accelerator

Name: health

DNS name: health

☐ Enable logging for this accelerator

Accelerator proxy port: 80

Web server port: 80

Accelerator IP addresses:
- ☐ 10.1.1.12
- ☑ 200.100.51.209
- ☑ 200.100.50.209

Web server addresses:
- 10.1.50.1

Test Bed Configuration

The hardware configuration for both the performance and scalability and NonStop high availability are exactly the same. The only difference is the request method used by the Polygraph software. It is important to understand that the port numbers on the 24 Port Foundry ServerIron are programmed to act in a specific way. In other words, the ports into which the TaskSmart C-Series servers plug are defined as part of a cache group. Polygraph servers representing web servers in the Internet "cloud" have specific policy descriptions programmed on them. Finally, the clients are plugged into other ports. The table below summarizes how the Foundry ServerIron ports were divided up for this dual network configuration:

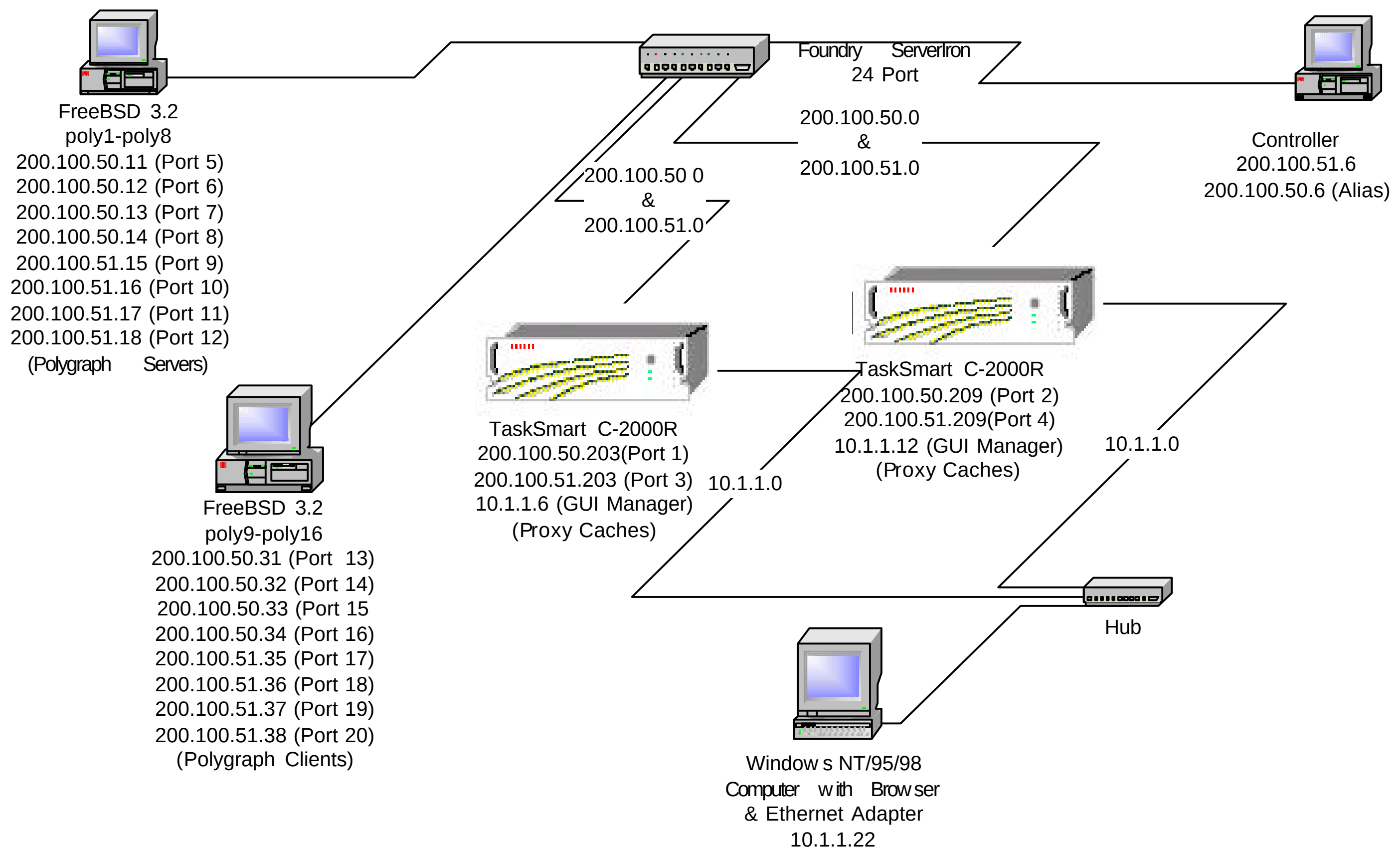
Note: Detailed instructions for programming the Foundry ServerIron are listed in Appendix A.

Table 1. Description of programming and connection information on the Foundry ServerIron

Port No.	**Host IP address using it**	**Host Description**	**ServerIron Programming Information**
1	200.100.50.203	TaskSmart #1 eth1	compaq1-50 (Managed via eth0 at 10.1.1.6)
2	200.100.50.209	TaskSmart #2 eth1	compaq2-50 (Managed via eth0 at 10.1.1.12)
3	200.100.51.203	TaskSmart #1 eth2	compaq1-51 (Managed via eth0 at 10.1.1.6)
4	200.100.51.209	TaskSmart #2 eth2	compaq2-51 (Managed via eth0 at 10.1.1.12)
5	200.100.50.11	Poly1 Server	ip-policy 2 (Layer 4 redirection) paired with Poly9
6	200.100.50.12	Poly2 Server	ip-policy 2 (Layer 4 redirection) paired with Poly10
7	200.100.50.13	Poly3 Server	ip-policy 2 (Layer 4 redirection) paired with Poly11
8	200.100.50.14	Poly4 Server	ip-policy 2 (Layer 4 redirection) paired with Poly12
9	200.100.51.15	Poly5 Server	ip-policy 2 (Layer 4 redirection) paired with Poly13
10	200.100.51.16	Poly6 Server	ip-policy 2 (Layer 4 redirection) paired with Poly14
11	200.100.51.17	Poly7 Server	ip-policy 2 (Layer 4 redirection) paired with Poly15
12	200.100.51.18	Poly8 Server	ip-policy 2 (Layer 4 redirection) paired with Poly16
13	200.100.50.31	Poly9 Client	cache-group 1 (load balancing) paired with Poly1
14	200.100.50.32	Poly10 Client	cache-group 1 (load balancing) paired with Poly2
15	200.100.50.33	Poly11 Client	cache-group 1 (load balancing) paired with Poly3
16	200.100.50.34	Poly12 Client	cache-group 1 (load balancing) paired with Poly4
17	200.100.51.35	Poly13 Client	cache-group 2 (load balancing) paired with Poly5
18	200.100.51.36	Poly14 Client	cache-group 2 (load balancing) paired with Poly6
19	200.100.51.37	Poly15 Client	cache-group 2 (load balancing) paired with Poly7
20	200.100.51.38	Poly16 Client	cache-group 2 (load balancing) paired with Poly8
21	200.100.50.200	Controller	no cache-group
22	200.100.51.200	Controller	no cache-group
23	No Connection	None	None
24	No Connection	None	None

The ports on the Foundry Server Iron are separated into three distinct groups. One group of ports is for the caches defining the real server addresses. The next group of ports is for connection to the simulated outside world (or Polygraph servers). Finally, the last group is for the simulated internal enterprise network (or Polygraph clients). Connections 21 or 22 can be used for the Polygraph controller.

Diagram of the Configuration



Testing Methodologies

Polygraph Simulations (Workloads)

The Web Polygraph Benchmark is a very extensible program. It uses a simulated web server called “polysrv” and a simulated client browser called “polyclt”. The server and client workload simulations are executed via Perl scripts provided with the Polygraph source code containing command line information. The Perl scripts are not well documented at this time by the Ircache team. (See Appendix C for modification details for bb.pl script) For this document, the PolyMix-1 workload will be used for performance and scalability testing. A slightly modified version of the PolyMix-1 workload will be used for the high availability test. For more information on Web Polygraph and its associated workloads, please visit http://polygraph.ircache.net.

Polygraph Workload for Performance and Scalability Testing

The Web Polygraph PolyMix-1 workload is a one hour test that is used to measure cache throughput in http requests/second. Its main characteristics are a uniform request rate, a document hit ratio of 55%, a mean request size of 13Kbytes, and 80% of the requests are capable of being cached (20% non-cacheable).

A reasonable result for a Compaq TaskSmart C-2000R is a peak of 1,200 request/second with an average response time of less than 3 seconds per request while maintaining a document hit rate of 55%. The server and client command line information is listed below:

Table 2. Description of variables used in command lines below

Variable Name	Description	Example
$proxy	IP address of the TaskSmart proxy	200.100.50.203
$origin	IP address of a corresponding Polygraph Server	200.100.50.10
$rr	Polygraph request rate (Uniform in PolyMix-1)	300
$logname	Name of the file to store Polygraph logs	my25log

Server command line:

```
./polysrv - - port 80 \
--xact_think norm:3s,1.5s \
--verb_lvl 1 \
-- goal –1:1hr:0.30
```

Client command line:

```
./polyclt  -- ports 1024:30000 \
--proxy $proxy:8080 \
--origin $origin:80 \
--verb_lvl 2 \
--robots 1 \
--rep_cachable 80p \
--dhr 55p \
--req_rate $rr/sec
--abs_urls no \
--pop_model unif \
--tmp_loc none \
--cool_phase 1min \
-- log $logname \
--goal –1:1hr:0.30
```

Polygraph Workload for High Availability Test

The workload for High Availability is similar to the PolyMix-1 test that is used to measure cache throughput in http requests/second. The main difference is the best effort request rate instead of the uniform request rate in PolyMix-1. Best effort request rates do not force the Polygraph client to make a specific number of requests per second, but instead instructs it to make as many as possible per second. To use the best effort request model with Polygraph client, just omit the *"—req_rate"* command line parameter, change the number of *"—robots"* from 1 to 300, and the Polygraph clients will attempt as many requests as possible. The reason for not specifying a particular request rate is due to the Polygraph clients' expectation that the rate be maintained and will drop off (run out of file descriptors) when it is not maintained. Best effort allows the clients to continue at a lower rate if necessary, just like real users in an enterprise network.

Server command line:

```
./polysrv - - port 80 \
--xact_think norm:3s,1.5s \
--verb_lvl 1 \
-- goal –1:1hr:0.30
```

Client command line:

```
./polyclt  -- ports 1024:30000 \
--proxy $proxy:8080 \
--origin $origin:80 \
--verb_lvl 2 \
--robots 300 \
--rep_cachable 80p \
--dhr 55p \
--abs_urls no \
--pop_model unif \
--tmp_loc none \
--cool_phase 1min \
-- log $logname \
--goal –1:1hr:0.30
```

Polygraph Performance and Scalability Test

The Polygraph performance scalability test consists of two sections. The first section is to use the client server pairs to fill the cache with objects. The second section uses the same client and server pairs to run the one-hour PolyMix-1 workload for ten different constant requests/second rates with ten minute idle periods between each. The fill procedure is required and takes less than two hours for a Compaq TaskSmart C-2000R; the PolyMix-1 workload section takes about 12 hours. A single Compaq TaskSmart C-2000R will reliably sustain over 1,200 request/second peak using PolyMix-1 workload. Using two TaskSmart C-2000R servers in a load-balanced configuration will reliably sustain 2,400 requests/second, thus achieving near linear scalability.

Polygraph Performance Test Description

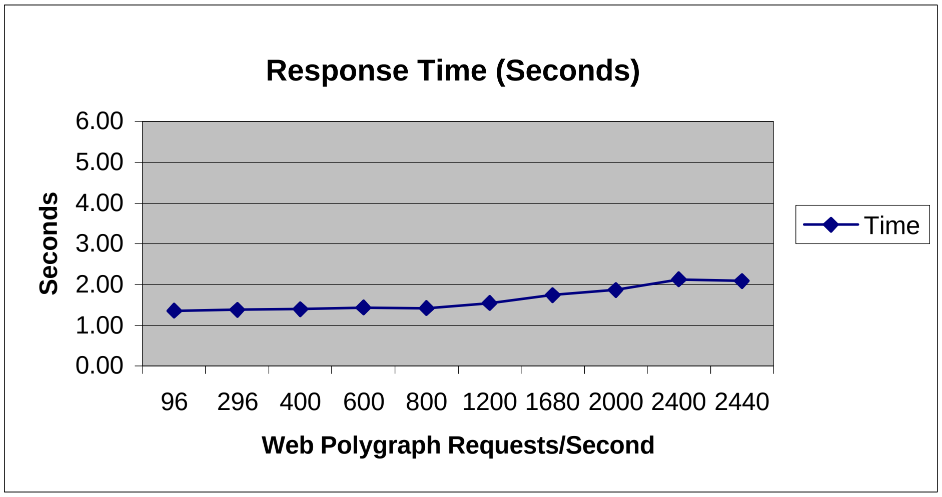
Simulated Polygraph requests for the test are generated by using four Polygraph client/server pairs connected to each network (200.100.50.0 and 200.100.51.0). Each Polygraph client is programmed through the Perl script to request 1/8 of the total request rate load. So, to generate a constant request rate load of 2,400 requests/second, each Polygraph client is programmed to make 300 requests/second each. Responses to the Polygraph clients' requests are provided by the Polygraph servers, have a mean size of 13K bytes, and are the equivalent of one web object. The workload defines the content as 80% cacheable, and the document hit rate for requests is 55% for the test. In other words, the Compaq TaskSmart C-2000R will cache the objects that are cacheable, and serve about half (55%) of them to the clients directly. All the other objects will be requested from the Polygraph servers and sent to the clients. In the load-balanced configuration, the Foundry ServerIron nearly doubles the capability by automatically splitting the requests between each of the TaskSmart C-2000R servers.

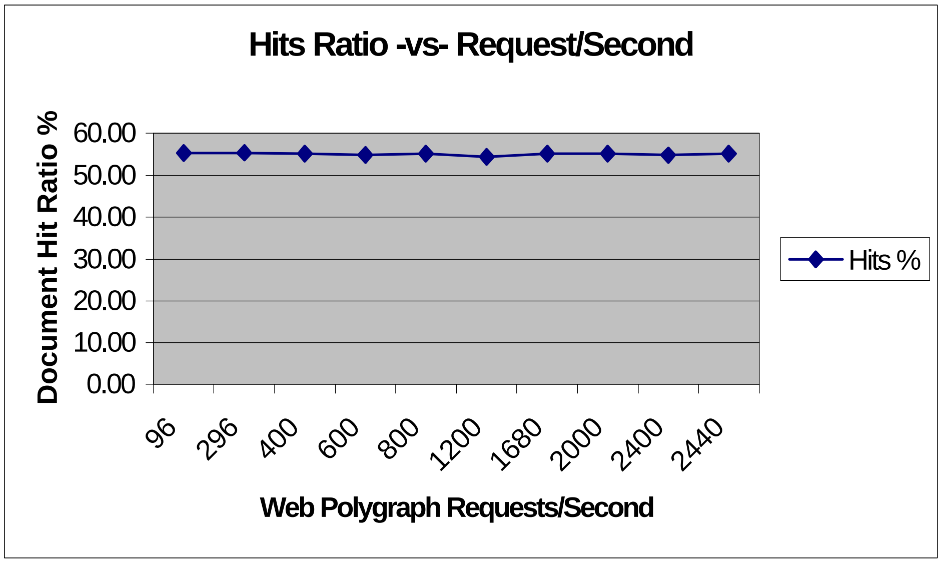
A successful performance test for the TaskSmart C-2000R is to maintain a 55% document hit ratio with a measured response time well under 3 seconds per request at each of the ten request rates used in the test. This is especially true for the peak 2,440 requests/second test period. A caching server configuration that doesn't perform as well will be unable to maintain the same peak request/second rate, have longer response times, and have a document hit ratio less than 55%.

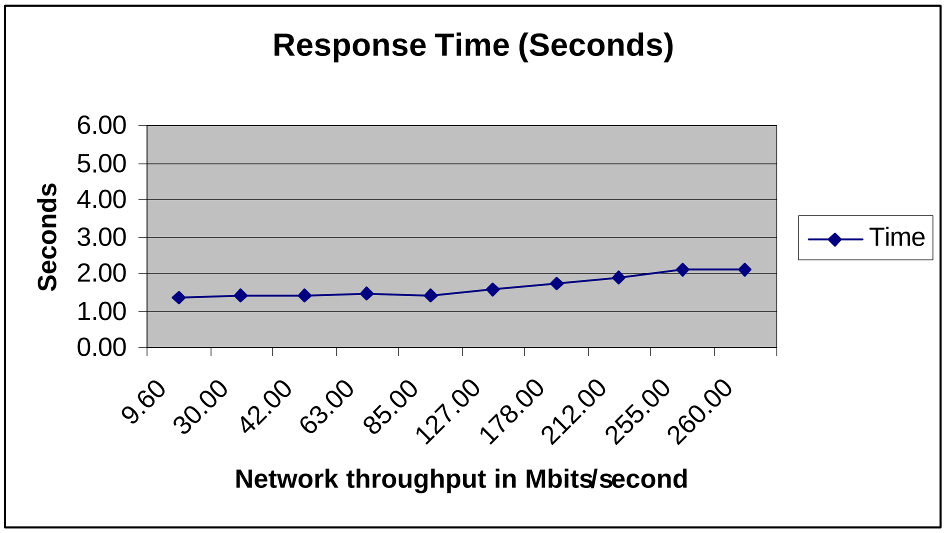
The table below shows a summary of the results of the test from the Polygraph client logs:

Table 3. Polygraph PolyMix-1 Results

Request Rate (Per Second)	Mean Response Time (Seconds)	Hit Ratio (Percent)	Throughput (Mbits/Second)	Run order
96	1.35973	55.30	9.60	10
296	1.38166	55.36	30.00	6
400	1.39517	55.26	42.00	8
600	1.43482	54.91	63.00	4
800	1.42218	55.17	85.00	7
1200	1.54743	54.42	127.00	3
1680	1.74610	55.11	178.00	5
2000	1.87042	55.11	212.00	9
2400	2.12753	54.79	255.00	2
2440	2.08604	55.12	260.00	1





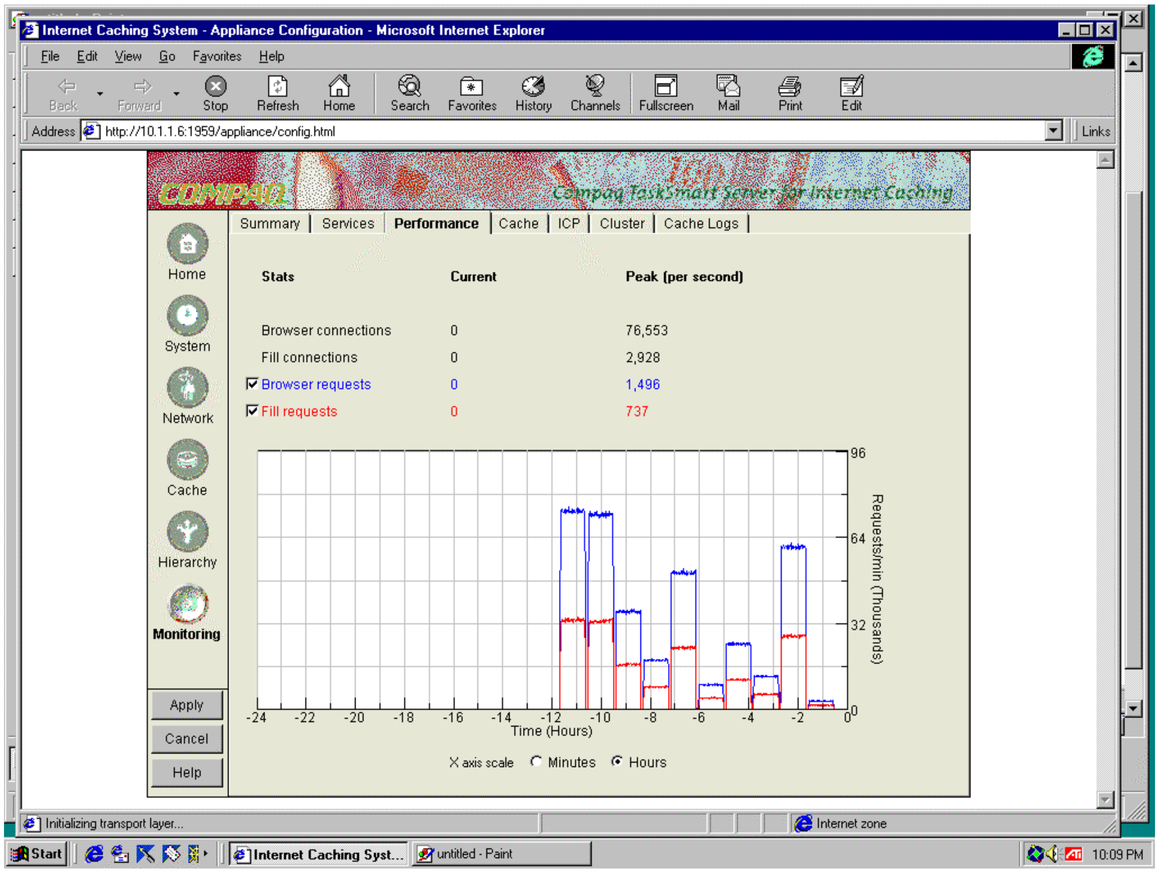


Note: As shown in the diagram, the Foundry ServerIron was not dedicated to doing only load balancing. Dedicating the Foundry ServerIron for this dual cache test would have utilized less than 30% of its total load-balancing capacity. So, adding additional networking infrastructure and TaskSmart C-2000R servers will allow the configuration to be expanded with 95% scalability. The only limiting factors are the number of ports and the actual “wire speed” (layer 2 and layer 3) of the network switches and hubs being used in the network infrastructure.

Monitoring the Test

The Compaq TaskSmart C-Series GUI management monitoring screen’s “performance” tab will display a graph. The graph will display a blue (upper) line representing total browser (Polygraph Client) requests, and a red (lower) line representing total requests that had to be obtained from the origin server (Polygraph server) to be fulfilled. Another name for the red line is “fill” requests. The difference between the two lines is the number of responses sent directly from the TaskSmart C-2000R caches to the Polygraph clients. To get the total requests serviced, the requests serviced by both the TaskSmart C-2000R servers must be added together.

Cache #1 Screen


Internet Caching System - Appliance Configuration - Microsoft Internet Explorer

Address http://10.1.1.6:1959/appliance/config.html

Compaq TaskSmart Server for Internet Caching

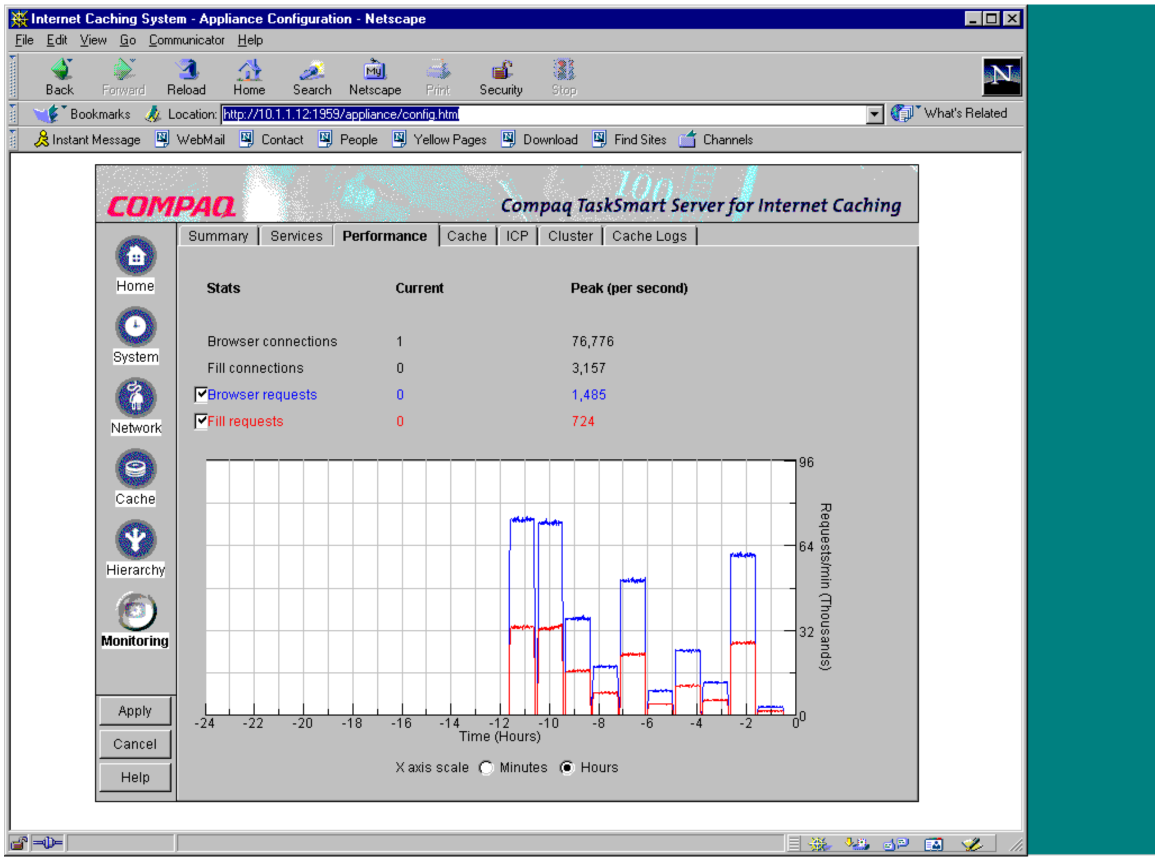
Summary | Services | Performance | Cache | ICP | Cluster | Cache Logs

Stats	Current	Peak (per second)
Browser connections	0	76,553
Fill connections	0	2,928
☑ Browser requests	0	1,496
☑ Fill requests	0	737

X axis scale ( ) Minutes (● Hours)

Initializing transport layer...

Cache #2 Screen


Internet Caching System - Appliance Configuration - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://10.1.1.12:1959/appliance/config.html
What's Related
Instant Message WebMail Contact People Yellow Pages Download Find Sites Channels
COMPAQ
Compaq TaskSmart Server for Internet Caching
Summary Services Performance Cache ICP Cluster Cache Logs
Home
System
Network
Cache
Hierarchy
Monitoring
Apply
Cancel
Help
96
64
32
0
Requests/min (Thousands)
-24 -22 -20 -18 -16 -14 -12 -10 -8 -6 -4 -2 0
Time (Hours)

Stats	Current	Peak (per second)
Browser connections	1	76,776
Fill connections	0	3,157
✓ Browser requests	0	1,485
✓ Fill requests	0	724

X axis scale ○ Minutes [✓ ● Hours]

Note the difference between the upper and lower lines represents the bandwidth savings. The upper line is the total responses, and the lower lines are “fill” requests from the “internet cloud.” In other words, over half of the total responses to client requests are served “locally” from cache, so the connection to the Internet is used only half as much. Higher hit ratios would show a larger gap; lower hit ratios would have a smaller gap between the lines. The larger the gap, the more bandwidth is being saved.

Polygraph High Availability (NonStop) Test

The Polygraph High Availability test consists of two tests. The first test simulates a single network failure, and the second test simulates a cache failure. Both failure simulations are tested by removing one or two network cables and observing the Polygraph clients’ and Compaq TaskSmart C-2000R servers’ behavior. The Polygraph clients were programmed using the Perl scripts to attempt to make as many object requests as possible using the 80% cacheable and 55% document hit rate rule used in the previous performance test (See Appendix C for modification details for bb.pl script). Programming the Polygraph clients to attempt as many requests as possible is called a “best effort” request rate. In the previous performance test, the request rate was fixed. There are two network connections per cache box, 4 total. Each connection carries about 25% of the load. Therefore, removing one cable will degrade request rates 25%; moving two cables will reduce request rates by 50%.

Polygraph High Availability Test Description

The purpose of the High Availability test is to demonstrate the fault tolerance of the configuration by putting as much request "pressure" on the system as possible and to simulate network and cache failures.

With all eight Polygraph client/server pairs running, the Compaq TaskSmart C-2000R load balanced system serviced more than 2,500 "best effort" requests/second. Twenty minutes into the test, one network cable from one of the TaskSmart C-2000R was disconnected from the Foundry ServerIron. There are two connections per cache, four total. Each connection carries about 25% of the total load. As expected, the "best effort" request rate dropped about 25% or to 1,850 requests/second. Restoring the cable connection resumed a 2,500 "best effort" request/second rate. Later in the hour, one cache was removed from the load-balanced system by disconnecting both network cables from the Foundry ServerIron. The request rate dropped to about half or 1,200 "best effort" requests/second. Restoring the connections caused the request rate to return to the 2,500 "best effort" requests/second rate.

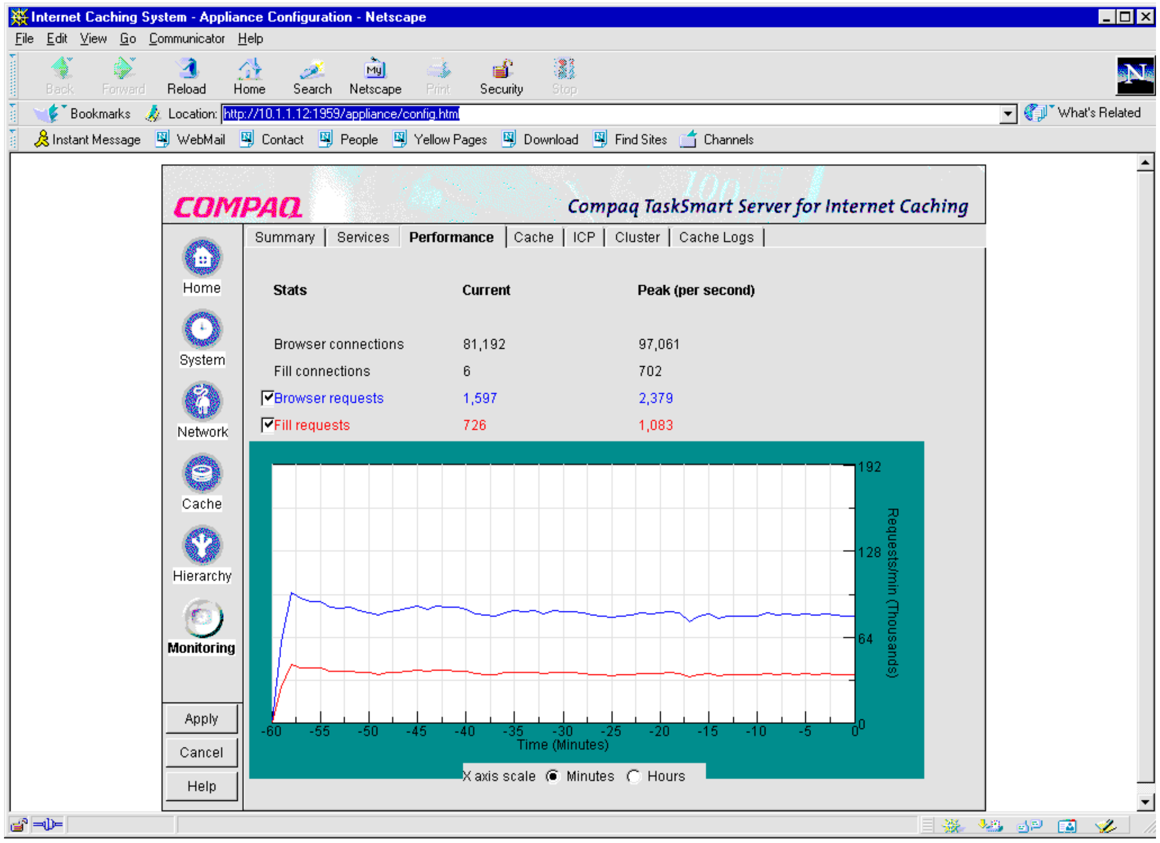
Note: Adding an additional Foundry ServerIron and enabling the ServerIron Hot Standby Recovery feature will eliminate any single points of failure. More details about Hot Standby Recovery and other Transparent Caching features are available in the "Foundry ServerIron Installation and Configuration Guide".

Monitoring the High Availability Test

The Compaq TaskSmart C-Series GUI management monitoring screen's "performance" tab will display a graph. The graph will display a blue (upper) line representing total browser (Polygraph Client) requests and a red (lower) line representing total requests that had to be obtained from the origin server (Polygraph server) to be fulfilled. Another name for the red line is "fill" requests. The difference between the two lines is the number of responses sent directly from the TaskSmart C-2000R caches to the Polygraph clients and those "filled" from Polygraph servers (cache hit ratio).

The graphs below demonstrate what happens when one network cable is disconnected from Cache #2 at –40 minutes into the test and reconnected several minutes later at –35 minutes. Later in the test at –15 minutes, both network cables were disconnected from Cache #2 and reconnected at –10. Also, note minimal impact on Cache #1 in the cluster. It continued to respond at total capacity even when Cache #2 was degraded. The cluster of two TaskSmart C-2000R servers was running at full capacity accepting about 2,500 requests/second (150K requests/minute) when the cables were disconnected.

Cache #1 Screen

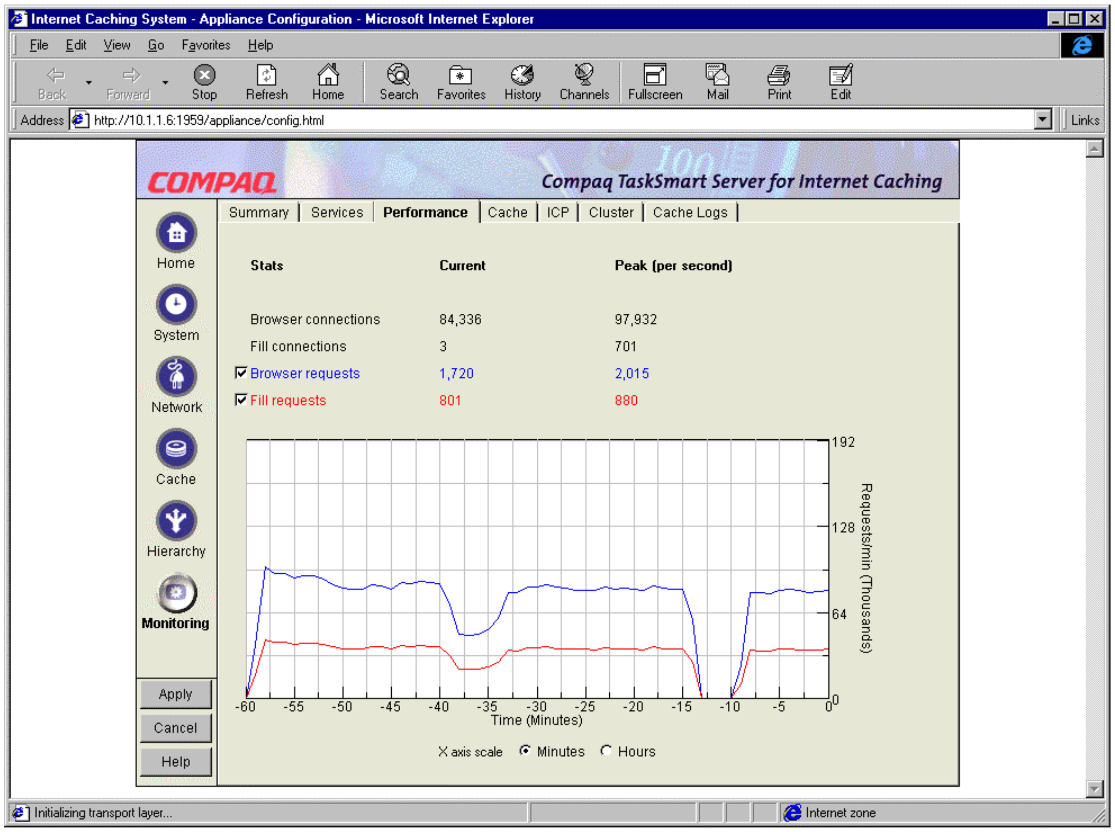

Internet Caching System - Appliance Configuration - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://10.1.1.12:1959/appliance/config.html What's Related
Instant Message WebMail Contact People Yellow Pages Download Find Sites Channels
COMPAQ Compaq TaskSmart Server for Internet Caching
Summary Services Performance Cache ICP Cluster Cache Logs
Home System Network Cache Hierarchy Monitoring
Apply Cancel Help

Stats	Current	Peak (per second)
Browser connections	81,192	97,061
Fill connections	6	702
✓ Browser requests	1,597	2,379
✓ Fill requests	726	1,083

192
128
64
0
Requests/min (Thousands)
-60 -55 -50 -45 -40 -35 -30 -25 -20 -15 -10 -5 0
Time (Minutes)

X axis scale [✓ Minutes] Hours

Cache #2 Screen


Internet Caching System - Appliance Configuration - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print Edit
Address http://10.1.1.6:1959/appliance/config.html
Links
COMPAQ
Compaq TaskSmart Server for Internet Caching
Summary | Services | Performance | Cache | ICP | Cluster | Cache Logs
Home
System
Network
Cache
Hierarchy
Monitoring
Apply
Cancel
Help
192
128
64
0
Requests/min (Thousands)
-60 -55 -50 -45 -40 -35 -30 -25 -20 -15 -10 -5 0
Time (Minutes)
Initializing transport layer...
Internet zone

Stats	Current	Peak (per second)
Browser connections	84,336	97,932
Fill connections	3	701
☑ Browser requests	1,720	2,015
☑ Fill requests	801	880

X axis scale [✓ Minutes] Hours

Note the difference between the upper and lower lines track at the designated 55% hit rate during and after both failure conditions.

Performance Summary

The TaskSmart C-Series server in both testing scenarios achieved extraordinary performance, scalability, and fault tolerance. Compared to other more traditional upgrade methods such as adding processors or round-robin DNS, the TaskSmart C-Series server and Foundry ServerIron demonstrated near 100% scalability with the NonStop fault tolerance expected from Compaq. The load balanced configuration maintained the specified document hit rate (bandwidth savings) even in failure conditions and recovered to full capacity when the fault was repaired. For applications requiring more capability, additional networking infrastructure and TaskSmart C-Series servers can be added to the configuration with the same outstanding scalability and fault tolerance.

Appendix A

Foundry Programming Procedure

In order to program the Foundry ServerIron to match the Table 2 description of programming and connection information on the Foundry ServerIron , a connection must be established via the serial management port or telnet. Foundry also provided a very user friendly web interface, but because all the procedures cannot be done using it, only the command line will be documented. The web interface can be used for all of the procedures except for setting the Hash-Mask value for the load balancing function and the ServerIron management (web/telnet access) address.

In this configuration, a Windows95 Hyperterm client was connected via serial cable to the ServerIron. Be sure to set Hyperterm for 9600 Baud, 8 Data Bits, No Parity, 1 Stop bit, and NO flow control. Below are the steps to program the ServerIron for Layer 4 and Load Balancing.

Step 1. Clear the Foundry ServerIron and reload (reboot) it by typing the following commands:

```
ServerIron> erase startup no-config <enter>
Erase flash Done.

ServerIron# reload <enter>
Enter 'b' to go to boot monitor ...
BOOT INFO: load from primary copy
BOOT INFO: code decompression completed
BOOT INFO: branch to 04001500
INFO: empty config data in the primary area, try to read from backup
INFO: empty config data in the backup area also
 SW: Version 05.0.03T12 Copyright (c) 1996-1999 Foundry Networks, Inc.
     Compiled on Jun 29 1999 at 10:56:44 labeled as SLB05003
 HW: ServerIron Switch, serial number 0333f8
 240 MHz Power PC processor 603 (revision 7) with 32756K bytes of DRAM
  24 100BaseT interfaces with Level 1 Transceiver LXT975
   2 GIGA uplink interfaces, SX
 256 KB PRAM and 8*2048 CAM entries for DMA 0, version 0807
 256 KB PRAM and 8*2048 CAM entries for DMA 1, version 0807
 256 KB PRAM and 8*2048 CAM entries for DMA 2, version 0807
128 KB boot flash memory
4096 KB code flash memory
2048 KB BRAM, BM version 10
 128 KB QRAM
512 KB SRAM
Octal System, Maximum Code Image Size Supported: 1965568 (0x001dfe00)
```

Step 2. Enable super user (switch administrator or privileged EXEC) by typing the following command:

```
ServerIron> enable <enter>
No password has been assigned yet...
ServerIron# config terminal <enter>
ServerIron(config)#
```

Step 3. Set up the ip address for administrative access to the ServerIron by typing the information below:

```
ServerIron(config)# ip address 200.100.50.199 255.255.255.0 <enter>
```

Note: The ip address above can now be used to access the ServerIron via telnet and the Web interface.

Step 4. Write the new settings to flash memory by typing the following command:

```
ServerIron(config)#  write memory <enter>
```

Step 5. Write Global Settings and disable spanning tree by entering the following commands:

```
ServerIron(config)#  Server reassign-threshold 200 <enter>
ServerIron(config)#  Server tcp-age 2 <enter>
ServerIron(config)#  Server source-nat <enter>
ServerIron(config)#  no global-st <enter>
```

Step 6. Create IP Policy 2 for Layer 4 Transparent http cache operation.

```
ServerIron(config)#  ip policy 2 cache tcp http local  <enter>
```

Step 7. Set the ServerIron Ports 1 through 4 to no-cache group. These ports will be used by the TaskSmart C-series servers.

```
ServerIron(config)#  interface e 1  <enter>
ServerIron(config-if-1)#  no cache-group <enter>
ServerIron(config-if-1)#  exit <enter>
ServerIron(config)#  interface e 2 <enter>
ServerIron(config-if-2)#  no cache-group <enter>
ServerIron(config-if-2)#  exit <enter>
ServerIron(config)#  interface e 3  <enter>
ServerIron(config-if-3)#  no cache-group <enter>
ServerIron(config-if-3)#  exit <enter>
ServerIron(config)#  interface e 4 <enter>
ServerIron(config-if-4)#  no cache-group <enter>
ServerIron(config-if-4)#  exit <enter>
```

Step 8. Set the ServerIron Ports 5 through 12 to no-cache group and apply ip policy 2. These ports will be used by the Polygraph servers to simulate the outside world or Internet cloud.

```
ServerIron(config)#  interface e 5  <enter>
ServerIron(config-if-5)#  no cache-group <enter>
ServerIron(config-if-5)#  ip-policy 2 <enter>
ServerIron(config-if-5)#  exit <enter>
ServerIron(config)#  interface e 6 <enter>
ServerIron(config-if-6)#  no cache-group <enter>
ServerIron(config-if-6)#  ip-policy 2 <enter>
ServerIron(config-if-6)#  exit <enter>
ServerIron(config)#  interface e 7  <enter>
ServerIron(config-if-7)#  no cache-group <enter>
ServerIron(config-if-7)#  ip-policy 2 <enter>
ServerIron(config-if-7)#  exit <enter>
ServerIron(config)#  interface e 8 <enter>
ServerIron(config-if-8)#  no cache-group <enter>
ServerIron(config-if-8)# ip-policy 2 <enter>
ServerIron(config-if-8)# exit <enter>
ServerIron(config)# interface e 9  <enter>
ServerIron(config-if-9)# no cache-group <enter>
ServerIron(config-if-9)# ip-policy 2 <enter>
ServerIron(config-if-9)# exit <enter>
ServerIron(config)# interface e 10 <enter>
ServerIron(config-if-10)# no cache-group <enter>
ServerIron(config-if-10)# ip-policy 2 <enter>
ServerIron(config-if-10)# exit <enter>
ServerIron(config)# interface e 11  <enter>
ServerIron(config-if-11)# no cache-group <enter>
ServerIron(config-if-11)# ip-policy 2 <enter>
ServerIron(config-if-11)# exit <enter>
ServerIron(config)# interface e 12 <enter>
ServerIron(config-if-12)# no cache-group <enter>
ServerIron(config-if-12)# ip-policy 2 <enter>
ServerIron(config-if-12)# exit <enter>
```

Step 9. Set ServerIron Ports 13 through 16 as cache-group 1. These ports will be used as client browser simulators, and represent the internal enterprise network. Note: When you type the "Show Config" command to verify everything is working, interfaces (ports) 13 through 16 will not display a "cache-group 1" because it is considered to be the default.

```
ServerIron(config)# interface e 13 <enter>
ServerIron(config-if-13)# cache-group 1 <enter>
ServerIron(config-if-13)# exit <enter>
ServerIron(config)# interface e 14 <enter>
ServerIron(config-if-14)# cache-group 1 <enter>
ServerIron(config-if-14)# exit <enter>
ServerIron(config)# interface e 15 <enter>
ServerIron(config-if-15)# cache-group 1 <enter>
ServerIron(config-if-15)# exit <enter>
ServerIron(config)# interface e 16 <enter>
ServerIron(config-if-16)# cache-group 1 <enter>
ServerIron(config-if-16)# exit <enter>
```

Step 10. Set ServerIron Ports 17 through 20 as cache-group 2. These ports will be used as client browser simulators, and represent the internal enterprise network. Note: When you type the "Show Config" command to verify everything is working, interfaces (ports) 17 through 20 will display a "cache-group 2".

```
ServerIron(config)# interface e 17 <enter>
ServerIron(config-if-17)# cache-group 2 <enter>
ServerIron(config-if-17)# exit <enter>
ServerIron(config)# interface e 18 <enter>
ServerIron(config-if-18)# cache-group 2 <enter>
ServerIron(config-if-18)# exit <enter>
ServerIron(config)# interface e 19 <enter>
ServerIron(config-if-19)# cache-group 2 <enter>
ServerIron(config-if-19)# exit <enter>
ServerIron(config)# interface e 20 <enter>
ServerIron(config-if-20)# cache-group 2 <enter>
ServerIron(config-if-20)# exit <enter>
```

Step 11. Create a named description for each network on the cache servers. The named descriptions include a name for each logical cache on a particular network. In this configuration, there are two cache servers with two networks each yielding four named entries.

```
ServerIron(config)# server cache-name compaq1-50 200.100.50.201<enter>
ServerIron(config-rs-compaq1-50)# asymmetric <enter>
ServerIron(config-rs-compaq1-50)# port http <enter>
ServerIron(config-rs-compaq1-50)# exit <enter>
ServerIron(config)# server cache-name compaq1-51 200.100.51.201<enter>
ServerIron(config-rs-compaq1-51)# asymmetric <enter>
ServerIron(config-rs-compaq1-51)# port http <enter>
ServerIron(config-rs-compaq1-51)# exit <enter>
ServerIron(config)# server cache-name compaq2-50 200.100.50.202<enter>
ServerIron(config-rs-compaq2-50)# asymmetric <enter>
ServerIron(config-rs-compaq2-50)# port http <enter>
ServerIron(config-rs-compaq2-50)# exit <enter>
ServerIron(config)# server cache-name compaq2-51 200.100.51.202<enter>
ServerIron(config-rs-compaq2-51)# asymmetric <enter>
ServerIron(config-rs-compaq2-51)# port http <enter>
ServerIron(config-rs-compaq2-51)# exit <enter>
ServerIron(config)#
```

Step 12. Create two cache groups, one for each network being used. Set the global cache to non-stateful and the "hash-mask" source and destination to all ones and all zeros (255.255.255.255 0.0.0.0).

```
ServerIron(config)# no server cache-stateful <enter>
ServerIron(config)# server cache-group 1 <enter>
ServerIron(config-tc-1)# hash-mask 255.255.255.255  0.0.0.0 <enter>
ServerIron(config-tc-1)# cache-name compaq1-50 <enter>
ServerIron(config-tc-1)# cache-name compaq2-50 <enter>
ServerIron(config-tc-1)# exit <enter>
ServerIron(config)# server cache-group 2 <enter>
ServerIron(config-tc-1)# hash-mask 255.255.255.255  0.0.0.0 <enter>
ServerIron(config-tc-1)# cache-name compaq1-51 <enter>
ServerIron(config-tc-1)# cache-name compaq2-51 <enter>
ServerIron(config-tc-1)# exit <enter>
ServerIron(config)#
```

To assure that the ServerIron is configured properly, type the command below. Please remember that interfaces e 13, e14, e15, and e16 will not display cache-group 1.

```
ServerIron(config)#show config <enter>
Startup configuration:
!
ver 05.0.00T12
no global-stp
!
!
server reassign-threshold 200
server tcp-age 2
server source-nat
server source-ip 200.100.51.199 255.255.255.0 0.0.0.0
!
```

```
server cache-name compaq1-50 200.100.50.201
asymmetric
port http
port http url "HEAD /"
!
server cache-name compaq1-51 200.100.51.201
asymmetric
port http
port http url "HEAD /"
!
server cache-name compaq2-50 200.100.50.202
asymmetric
port http
port http url "HEAD /"
!
server cache-name compaq2-51 200.100.51.202
asymmetric
port http
port http url "HEAD /"
!
!
no server cache-stateful
server cache-group 1
hash-mask 255.255.255.255 0.0.0.0
cache-name compaq1-50
cache-name compaq2-50
server cache-group 2
hash-mask
255.255.255.255 0.0.0.0
cache-name compaq1-51
cache-name compaq2-51
!
!
boot sys fl sec
ip address 200.100.50.199 255.255.255.0
ip default-gateway 200.100.50.1
ip policy 2 cache tcp http local
snmp-server community ..... rw
no span
!
interface e 1
no cache-group
!
interface e 2
no cache-group
!
interface e 3
no cache-group
!
interface e 4
no cache-group
!
interface e 5
no cache-group
ip-policy 2
!
interface e 6
no cache-group
ip-policy 2
!
interface e 7
no cache-group
ip-policy 2
```

```
!
interface e 8
no cache-group
ip-policy 2
!
interface e 9
no cache-group
ip-policy 2
!interface e 10
no cache-group
ip-policy 2
!
interface e 11
no cache-group
ip-policy 2
!
interface e 12
no cache-group
ip-policy 2
!
interface e 13
!
interface e 14
!
interface e 15
!
interface e 16
!
interface e 17
cache-group 2
!
interface e 18
cache-group 2
!
interface e 19
cache-group 2
!
interface e 20
cache-group 2
!
interface e 21
!
interface e 22
!
interface e 23
!
Interface e 24
!
end
ServerIron(config)#
```

Appendix B

Secure Shell

Secure Shell is a program to log into another computer over a network, to execute commands in a remote machine, and to move files from one machine to another. It provides strong authentication and secure communications over insecure channels. It is intended as a replacement for rlogin, rsh, and rcp. Secure Shell come in two distinctive parts: the ssh client and the sshd server.

Download the latest version of ssh from the following site and copy it to a DOS floppy.

ftp://ftp.pdc.kth.se/pub/ossh/ `ssh-1.2.12.tar.gz  or ossh-1.5.3.tar.gz`

Create one FreeBSD machine with two network cards similar to the way the Polygraph simulators are created. Be sure to answer yes to the “Create Ports” directories when asked. The ssh programs must be installed on all of the Polygraph clients and servers and the Polygraph controller. Attach the controller network adapter to the appropriate network.

To install, use the following steps:

Mount the DOS floppy as a Unix file system:
```
polyN# mount -t msdos /dev/fd0 /mnt  <enter>
```

Use the tar utility to unpack the ssh archive into the /usr/ports/distfiles directory:
```
polyN# tar -xvzf /mnt/ssh-1.2.12.tar.gz /usr/ports/distfiles <enter>
```

Change to the newly created directory under distfiles:
```
polyN#  cd /usr/ports/distfiles/ssh.1.2.12 <enter>
```

Once in the distfiles directory, type the following commands to create and install the ssh client package:
```
polyN# ./configure <enter>
polyN# make <enter>
polyN# make install <enter>
```

Once compiled, change to the /etc directory and add the following line to your /etc/rc.local file which will start up the secure shell server at boot time in quite mode:

```
PolyN# ee /etc/rc.local <enter>

/usr/local/sbin/sshd -q  <esc><enter>
```

and save changes.

Finally, reboot the machine to load the ssh server.

```
polyN# reboot
```

To assure ssh is working properly, install the ssh package on all the machines in the network and reboot them. Login to the Polygraph controller and type the following command which should attempt to Login to the machine at ip address 200.100.50.52, and answer "yes" to connect:

```
controller# ssh 200.100.50.52 <enter>
poly10# exit
controller#
```

Type 'exit' to return to the controller# prompt.

Repeat the step above to assure the controller is capable of logging into to all of the Polygraph machines on both networks.

Appendix C

The Polygraph Controller

The Polygraph controller is a single FreeBSD 3.2 machine configured similar to the Polygraph client/server pairs with Secure Shell (SSH), Perl 5.x, and Polygraph control Perl scripts. The unit can be any unit capable of running SSH and invoke Perl scripts. For this test, one of the Polygraph client machines was adapted to act as a controller because the FreeBSD 3.2 operating system, Polygraph software, and SSH were already installed on the unit (See Appendix B for SSH installation details).

Operating System & Network Changes

In order for the controller to work properly with both networks, an alias IP network address will be created in the /etc/rc.conf directory along with the existing IP network address. The alias address is for the 200.100.51.0 network. Once the alias is added, the controller can start Polygraph clients and servers on both the 200.100.50.0 and the 200.100.51.0 networks using the SSH program. Use the ee full screen editor program to add the ***ifconfig_fxp0*** line below to the /etc/rc.conf file:

#Existing network IP for 200.100.50.0 Network

ifconfig_fxp0 = “inet 200.100.50.6 netmask 255.255.255.0”

network_interfaces =”fxp0 lo0”

#Add alias for 200.100.51.0 Network

ifconfig_fxp0 200.100.51.6 alias

Modifying Controller Perl Script Files:

The Perl Script files are stored in the /usr/poly107/polygraph/tools/BB directory. The files are called bb.pl, Command.pm, Blob.pm, Executer.pm, Tools.pm, and Logger.pm. All filenames are case sensitive and the contents are in plain text format. The files should be copied to the /usr/poly107 directory before modifications are made. The bb.pl is the main Perl script and must be modified to work with multiple TaskSmart C-Series servers in a clustered configuration. Use the ee full screen editor program to modify the bb.pl file to look like the one below. All additions are shown in **bold** type:

```
#!/usr/local/bin/perl -w
use strict;
use lib '/usr/people/bakeoff/BB';
require Command;
require Blob;
require Executer;
require Tools;
use Logger;
# Client -> Server Names
my @Hosts = (
#       &importHostNames(),  # use shell environment variables
#'10.11.11.11' => '10.11.11.12', # or hard coded pairs
        '200.100.50.31' => '200.100.50.11',
        '200.100.50.32' => '200.100.50.12',
        '200.100.50.33' => '200.100.50.13',
        '200.100.50.34' => '200.100.50.14',
        '200.100.51.35' => '200.100.51.15',
        '200.100.51.36' => '200.100.51.16',
        '200.100.51.37' => '200.100.51.17',
        '200.100.51.38' => '200.100.51.18',
);
my $Net = '200.100';              # 1st two octets of IP
my $Host = '203:8080';            # 4 octet of IP (host)
my $Proxy = '1.1.1.1';            # one for all
# my $Proxy = '200.100.51.202:8080'; # one for all
my $Master = '200.100.51.06';      # monitoring machine
my $Take = 001;
# my $Take = time()-929000000;
my %HostPairs = @Hosts;

my @HeadCmds = (
        Cmd('date'),
        Pairs(
                'ssh $clt killall polyclt',
                'ssh $srv killall polysrv',
        ),
        All('ssh $host mkdir /usr/home/logs'),
# No time used in this test
#       All('ssh $host /usr/sbin/ntpdate -b $Master'),
        Cmd('sleep 600'),
);

my @TailCmds = (
        Pairs(
                'ssh $clt killall -INT polyclt',
                'ssh $srv killall -INT polysrv',
        ),
        Pairs(
                'scp -p $clt:/usr/home/logs/clt.\* /usr/home/logs/',
                'scp -p $srv:/usr/home/logs/srv.\* /usr/home/logs/',
        ),
        All('ssh $host mv /usr/home/logs /usr/home/logs.local/$name'),
        Cmd('perl exprep.pl /usr/home/logs/clt.$exp_mask.*.log'),
        Cmd('date'),
);
my @BlobCfgs  = (
        {
         #Best Effort Routine ‘best effort’ –
         #Workload used for the Polygraph High Availability (NonStop) Test

                name => 'best-effort.rr$rr.take$Take',
                exp_mask => 'best-effort.rr:*:.take$Take',

                body => [ Pairs(
                        'ssh $clt /usr/home/poly107/polyclt
                                --ports 1024:30000
                                --verb_lvl 2
                                --proxy $srv:80 --origin $srv:80
                                --rep_cachable 80p
                                --robots $rr
                                --dhr 55p
                                --abs_urls no
                                --pop_model unif
                                --tmp_loc none
                                --cool_phase 1min
                                --goal -1:1hr:0.03
                                --log /usr/home/logs/clt.$name.$clt.log
                                --console /usr/home/logs/clt.$name.$clt.con
                                ',
                        'ssh $srv /usr/home/poly107/polysrv
                                --port 80
                                --verb_lvl 4
                                --idle_tout 600sec
                                --goal -1:1hr:0.03
                                --log /usr/home/logs/srv.$name.$srv.log
                                --console /usr/home/logs/srv.$name.$srv.con
                                ',
                        ),
                ],

                runs => [ Runs('rr', 300) ],
        },
        {
         #Load Balancing Cluster Routine
         #Workload for the Polygraph Performance & Scalability Test

                name => 'clust.rr$rr.take$Take',
                exp_mask => 'clust.rr:*:.take$Take',

                body => [ Pairs(
                        'ssh $clt sleep 10/;
                        /usr/home/poly107/polyclt
                                --ports 1024:30000
                                --verb_lvl 2
                                --proxy $srv:80 --origin $srv:80
                                --rep_cachable 80p
                                --robots 1
                                --req_rate $rr/sec
                                --dhr 55p
                                --abs_urls no
                                --pop_model unif
                                --tmp_loc none
                                --cool_phase 1min
                                --goal -1:1hr:0.03
                                --log /usr/home/logs/clt.$name.$clt.log
                                --console /usr/home/logs/clt.$name.$clt.con
                                ',
                        'ssh $srv /usr/home/poly107/polysrv
                                --port 80
                                --verb_lvl 4
                                --idle_tout 145sec
                                --xact_think norm:3s,1.5s
                                --goal -1:1hr:0.03
                                --log /usr/home/logs/srv.$name.$srv.log
                                --console /usr/home/logs/srv.$name.$srv.con
                                ',
                        ),
                ],
                runs => [ Runs('rr', 305, 300, 150, 75, 210, 37, 100, 50, 250,
12)],
        },
        {
                name => 'no-proxy.rr$rr.take$Take',
                exp_mask => 'no-proxy.rr:*:.take$Take',

                body => [ Pairs(
                        'ssh $clt sleep 10\;
                        /usr/home/poly132/polyclt
                                --ports 1024:30000
                                --verb_lvl 4
                                --origin $srv:80
                                --rng_seed $cmd_id
                                --rep_cachable 80p
                                --pconn_use_lmt zipf:64
                                --nagle off
                                --robots 1
                                --req_rate $rr/sec
                                --dhr 55p
                                --pop_model unif
                                --tmp_loc none
                                --cool_phase 1min
                                --goal -1:1hr:0.30
                                --log_size 15MB
                                --log /usr/home/logs/clt.$name.$clt.log
                                --console /usr/home/logs/clt.$name.$clt.con
                                --notify $Master:18256
                                --label np$rr
                                ',
                        'ssh $srv /usr/home/poly132/polysrv
                                --port 80
                                --verb_lvl 4
                                --rng_seed $cmd_id
                                --idle_tout 135sec
                                --pconn_use_lmt zipf:16
                                --nagle off
                                --xact_think norm:3s,1.5s
                                --obj_bday const:-1year
                                --obj_with_lmt 100p
                                --obj_life_cycle const:2year
                                --obj_expire 100p=lmt+const:1
                                --goal 4.1hr
                                --log_size 15MB
                                --log /usr/home/logs/srv.$name.$srv.log
                                --console /usr/home/logs/srv.$name.$srv.con
                                --notify $Master:18256
                                --label np$rr
                                ',
                        ),
                ],

                runs => [ Runs('rr', 500) ],
        },
        {
         #Fill Procedure Routine 'fill'
         #Workload for the Polygraph cache fill procedure

                name => 'fill.cl$cl.take$Take',
                exp_mask => 'fill.cl:*:.take$Take',

                body => [ Pairs(
                        'ssh $clt /usr/home/poly132/polyclt
                                --ports 1024:30000
                                --verb_lvl 4
                                --proxy $Proxy --origin $srv:80
                                --rng_seed $cmd_id
                                --unique_urls 1
                                --rep_cachable 100p
                                --pconn_use_lmt const:1000
                                --nagle off
```

```
                        --robots $cl
                        --cool_phase 1min
                        --goal -1:1.25hr:.30
                        --log_size 35MB
                        --log /usr/home/logs/clt.$name.$clt.log
                        --console /usr/home/logs/clt.$name.$clt.con
                        --notify $Master:18256
                        --label fl$cl
                        ',
                'ssh $srv /usr/home/poly132/polysrv
                        --port 80
                        --verb_lvl 4
                        --rng_seed $cmd_id
                        --pconn_use_lmt const:1000
                        --nagle off
                        --obj_bday const:-1year
                        --obj_with_lmt 100p
                        --obj_life_cycle const:2year
                        --obj_expire 100p=lmt+const:1
                        --idle_tout 5min
                        --log_size 35MB
                        --log /usr/home/logs/srv.$name.$srv.log
                        --console /usr/home/logs/srv.$name.$srv.con
                        --notify $Master:18256
                        --label fl$cl
                        ',
                ),
        ],

        runs => [ Runs('cl', 10) ],
},

{
        # warning: this workload specs comes from the first
        # bake-off tests and may need adjustments with later
        # version of Polygraph
        name => 'pmix.rr$rr.take$Take',
        exp_mask => 'pmix.rr:*:.take$Take',

        body => [ Pairs(
                'ssh $clt /usr/home/poly132/polyclt
                        --ports 1024:30000
                        --verb_lvl 4
                        --proxy $Proxy --origin $srv:80
                        --rng_seed $cmd_id
                        --rep_cachable 80p
                        --robots 1
                        --req_rate $rr/sec
                        --dhr 55p
                        --pop_model unif
                        --tmp_loc none
                        --cool_phase 1min
                        --goal -1:1hr:0.30
                        --log /usr/home/logs/clt.$name.$clt.log
                        --console /usr/home/logs/clt.$name.$clt.con
                        ',
                'ssh $srv /usr/home/poly132/polysrv
                        --port 80
                        --verb_lvl 4
                        --rng_seed $cmd_id
                        --idle_tout 135sec
                        --xact_think norm:3s,1.5s
                        --goal 1.1hr
```

```
                                --log /usr/home/logs/srv.$name.$srv.log
                                --console /usr/home/logs/srv.$name.$srv.con
                                ',
                        ),
                ],

                runs => [ Runs('rr', 312, 300, 150, 75, 210, 37, 100, 50, 250,
12)],
        },
);


my @BlobPlan = ();
my @ActiveBlobNames = @ARGV;
my $FilterNames = scalar @ActiveBlobNames;
my @PossibleBlobNames = ();

foreach my $blob_cfg (@BlobCfgs) {
        my $blob = new BB::Blob($blob_cfg->{name});
        die() unless $blob_cfg->{body};
        die() unless $blob_cfg->{runs};
        $blob->exp_mask($blob_cfg->{exp_mask});
        $blob->head($blob_cfg->{head} || [@HeadCmds]);
        $blob->body($blob_cfg->{body});
        $blob->tail($blob_cfg->{tail} || [@TailCmds]);
        foreach (@{$blob_cfg->{runs}}) {
                my $b = $blob->clone($_);
                push @PossibleBlobNames, $b->name();
                # filter out extras
                if ($FilterNames) {
                        next if ! grep { $b->name() =~ m/$_/; } @ActiveBlobNames;
                }
                push @BlobPlan, $b;
        }
}

if (!@BlobPlan) {
        die("No blobs!\n") unless $FilterNames;
        die(sprintf("No matching blobs among: %s\n",
                join(' ', @PossibleBlobNames)));
}

&printRoutes(@Hosts);
&Log("starting...");
&Log("plan: ", join(' ', map { $_->name() } @BlobPlan));
&startNext(); # start the sequence
while (defined &BB::Executer::Step(60)) { ; }
&Log("terminating...");
exit 0;

#Subroutines

sub startNext {
        my $blob = shift @BlobPlan;
        return &Log("no more blobs") unless $blob;

        $blob->run(\&blobDone);
}

sub blobDone {
        &startNext();
}
```

```
sub Cmd {
      my ($tmpl, $label) = @_;
      return new BB::Command($tmpl, $label || 'local');
}

sub All {
      my $tmpl = shift;
      my $cmd = Cmd($tmpl, '$host');

      my @cfgs = Cfgs('host', @Hosts);
      return BB::Tools::Breed($cmd, @cfgs);
}

sub Pairs {
      my ($tmpl_clt, $tmpl_srv) = @_;

      my $cmd_clt = Cmd($tmpl_clt, '$clt');
      my $cmd_srv = Cmd($tmpl_srv, '$srv');

      # these are actually the same
      my @cfgs_clt = Cfgs2(\%HostPairs);
      my @cfgs_srv = Cfgs2(\%HostPairs);

      # note the order!
      return
            BB::Tools::Breed($cmd_srv, @cfgs_srv),
            BB::Tools::Breed($cmd_clt, @cfgs_clt);
}

sub Runs {
      my $var_name = shift;

      return Cfgs($var_name, @_);
}

sub Cfgs {
      my $var_name = shift;

      my @cfgs = ();

      foreach (@_) {
            push @cfgs, {
                  $var_name => $_,
                  Proxy => $Proxy,
                  Take => $Take,
                  Master => $Master,
            };
      }
      return @cfgs;
}

sub Cfgs2 {
      my $hosts = shift;
      my @cfgs = ();
      my ($clt, $srv);

# Modified by DLB – Compaq Computer Corp.
      while (($clt, $srv) = each %{$hosts}) {
            if ($Proxy ne '1.1.1.1') {
                  push @cfgs, { clt => $clt, srv => $srv,
                        Proxy => $Proxy,
                        Take => $Take,
                        Master => $Master,
                        };
                }
                else {
                    my(@str) = split(/\./,$clt);
                    (my $Proxy = join('.',$Net,$str[2],$Host));
                    push @cfgs, { clt => $clt, srv => $srv,
                        Proxy => $Proxy,
                        Take => $Take,
                        Master => $Master,
                        };
                }
        }

        return @cfgs;
}

# read well-known environment variables for clt/srv addresses
sub importHostNames {
        my $err = sprintf("bad \$clients or \$servers env vars!\nclt: %s\nsrv:
%s;\nstopped",
                $ENV{'clients'} || '<none>',
                $ENV{'servers'} || '<none>');
        die($err) if !$ENV{'clients'} || !$ENV{'servers'};
        my @clts = split(/\s+/, $ENV{'clients'});
        my @srvs = split(/\s+/, $ENV{'servers'});
        die($err) if !@clts || !@srvs;
        die($err) if (scalar @clts) != (scalar @srvs);
        die($err) if grep { !&hostAddr('[13579]$', $_) } @clts;
        die($err) if grep { !&hostAddr('[02468]$', $_) } @srvs;
        my @res = ();
        while (@clts && @srvs) {
                push @res, pop @clts, pop @srvs;
        }
        return @res;
}

sub printRoutes {
        my @routes = @_;

        my $table = '';
        for (my $i = 0; $i <= $#routes; $i += 2) {
                $table .= sprintf("%s => %s\n", $routes[$i], $routes[$i+1]);
        }
        &Log("routing table:\n", ButifyBuf($table));
}

sub hostAddr {
        my $pattern = shift;
        for (shift) {
                return undef unless $_;
                my @comps = split /\./;
                return undef if !@comps;
                return undef if $comps[$#comps] !~ /$pattern/;
        }
        return 1;
}
```

Running the bb.pl Controller Script:

To invoke the bb.pl Perl script, simply change to the /usr/poly107 directory and type the following command to fill the caches:

```
controller# ./run bb.pl fill <enter>
```

Once fill procedure is complete, type the following command to run the ten point PolyMix-1 test:

```
controller# ./run bb.pl clust <enter>
```

To execute the High-Availability NonStop testing script, type the following command:

```
controller# ./run bb.pl best-effort <enter>
```

Reporting Perl Script File:

Finally, there is an exprep.pl script file that processes the contents of multiple log files, and summarizes the results of each run. The exprep.pl can be found in the /usr/poly107/polygraph/src directory and was copied to the appropriate area for processing log files (/usr/poly107).