IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

ORACLE CORPORATION and ORACLE U.S.A. INC.,	)	
Plaintiffs/Counterclaim Defendants,	)	C.A. No. 06-414-SLR
v.	)	**JURY TRIAL DEMANDED**
EPICREALM LICENSING, LP,	)	
Defendant/Counterclaim Plaintiff.	)	

NOTICE OF SUBPOENA *DUCES TECUM AND AD TESTIFICANDUM*

PLEASE TAKE NOTICE that, pursuant to Rules 34 and 45 of the Federal Rules of Civil Procedure, a subpoena *duces tecum and ad testificandum* will be served on IBM Corporation. A true and correct copy of the subpoena is attached hereto.

POTTER ANDERSON & CORROON LLP

By: */s/ David E. Moore*
Richard L. Horwitz (#2246)
David E. Moore (#3983)
Hercules Plaza, 6th Floor
1313 N. Market Street
Wilmington, DE 19899
Tel: (302) 984-6000
rhorwitz@potteranderson.com
dmoore@potteranderson.com

Attorneys for Defendant/Counterclaim Plaintiff epicRealm Licensing, LP

OF COUNSEL:

Harry J. Roper
George S. Bosy
Aaron A. Barlow
Patrick L. Patras
David R. Bennett
Paul D. Margolis
Benjamin J. Bradford
JENNER & BLOCK
330 N. Wabash Avenue
Chicago, IL 60611-7603
Tel: (312) 923-8305

Dated: February 4, 2008
846396 / 31393 / Oracle

AO88 (Rev. 12/06) Subpoena in a Civil Case

Issued by the

UNITED STATES DISTRICT COURT

SOUTHERN DISTRICT OF NEW YORK

ORACLE CORPORATION and
ORACLE U.S.A., Inc.,
Plaintiffs

v.

EPICREALM LICENSING, LP,
Defendant
And Related Counterclaims

SUBPOENA IN A CIVIL CASE

Case Number:[1]
USDC, District of Delaware
Case No. 06-cv-414 (SLR)

TO: IBM Corporation
1 New Orchard Road
Armonk, NY 10504-1722, USA

☐ YOU ARE COMMANDED to appear in the United States District court at the place, date, and time specified below to testify in the above case.

PLACE OF TESTIMONY	COURTROOM
	DATE AND TIME

☒ YOU ARE COMMANDED to appear at the place, date, and time specified below to testify at the taking of a deposition in the above case. Pursuant to Federal Rule of Civil Procedure 30(b)(6). See Exhibit B.

PLACE OF DEPOSITION	DATE AND TIME
Jenner & Block LLP 919 3rd Avenue, 37th Fl., New York, NY 10022 (212) 891-1600	February 20, 2008 10:00 AM

☒ YOU ARE COMMANDED to produce and permit inspection and copying of the following documents or objects at the place, date, and time specified below (list documents or objects):

See Exhibit A.

PLACE	DATE AND TIME
Attn: George S. Bosy, Jenner & Block LLP 919 3rd Avenue, 37th Fl., New York, NY 10022 (212) 891-1600	February 15, 2008 10:00 AM

☐ YOU ARE COMMANDED to permit inspection of the following premises at the date and time specified below.

PREMISES	DATE AND TIME

Any organization not a party to this suit that is subpoenaed for the taking of a deposition shall designate one or more officers, directors, or managing agents, or other persons who consent to testify on its behalf, and may set forth, for each person designated, the matters on which the person will testify. Federal Rules of Civil Procedure, 30(b)(6).

ISSUING OFFICER'S SIGNATURE AND TITLE (INDICATE IF ATTORNEY FOR PLAINTIFF OR DEFENDANT)	DATE
Attorney for Defendant, EPICREALM LICENSING, LP	February 4, 2008

ISSUING OFFICER'S NAME, ADDRESS AND PHONE NUMBER

Patrick L. Patras, Jenner & Block LLP, 330 North Wabash Avenue, Chicago, IL 60611 (312) 222-9350

(See Rule 45, Federal Rules of Civil Procedure, Subdivisions (c), (d), and (e), on next page)

[1] If action is pending in district other than district of issuance, state district under case number.

AO88 (Rev. 12/06) Subpoena in a Civil Case

PROOF OF SERVICE

DATE PLACE

SERVED

SERVED ON (PRINT NAME) MANNER OF SERVICE

SERVED BY (PRINT NAME) TITLE

DECLARATION OF SERVER

I declare under penalty of perjury under the laws of the United States of America that the foregoing information contained in the Proof of Service is true and correct.

Executed on _____
DATE

SIGNATURE OF SERVER

ADDRESS OF SERVER

Rule 45, Federal Rules of Civil Procedure, Subdivisions (c), (d), and (e), as amended on December 1, 2006:

(c) PROTECTION OF PERSONS SUBJECT TO SUBPOENAS.

(1) A party or an attorney responsible for the issuance and service of a subpoena shall take reasonable steps to avoid imposing undue burden or expense on a person subject to that subpoena. The court on behalf of which the subpoena was issued shall enforce this duty and impose upon the party or attorney in breach of this duty an appropriate sanction, which may include, but is not limited to, lost earnings and a reasonable attorney's fee.

(2) (A) A person commanded to produce and permit inspection, copying, testing, or sampling of designated electronically stored information, books, papers, documents or tangible things, or inspection of premises need not appear in person at the place of production or inspection unless commanded to appear for deposition, hearing or trial.

(B) Subject to paragraph (d)(2) of this rule, a person commanded to produce and permit inspection, copying, testing, or sampling may, within 14 days after service of the subpoena or before the time specified for compliance if such time is less than 14 days after service, serve upon the party or attorney designated in the subpoena written objection to producing any or all of the designated materials or inspection of the premises — or to producing electronically stored information in the form or forms requested. If objection is made, the party serving the subpoena shall not be entitled to inspect, copy, test, or sample the materials or inspect the premises except pursuant to an order of the court by which the subpoena was issued. If objection has been made, the party serving the subpoena may, upon notice to the person commanded to produce, move at any time for an order to compel the production, inspection, copying, testing, or sampling. Such an order to compel shall protect any person who is not a party or an officer of a party from significant expense resulting from the inspection, copying, testing, or sampling commanded.

(3) (A) On timely motion, the court by which a subpoena was issued shall quash or modify the subpoena if it

(i) fails to allow reasonable time for compliance;

(ii) requires a person who is not a party or an officer of a party to travel to a place more than 100 miles from the place where that person resides, is employed or regularly transacts business in person, except that, subject to the provisions of clause (c)(3)(B)(iii) of this rule, such a person may in order to attend trial be commanded to travel from any such place within the state in which the trial is held;

(iii) requires disclosure of privileged or other protected matter and no exception or waiver applies; or

(iv) subjects a person to undue burden.

(B) If a subpoena

(i) requires disclosure of a trade secret or other confidential research, development, or commercial information, or

(ii) requires disclosure of an unretained expert's opinion or information not describing specific events or occurrences in dispute and resulting from the expert's study made not at the request of any party, or

(iii) requires a person who is not a party or an officer of a party to incur substantial expense to travel more than 100 miles to attend trial, the court may, to protect a person subject to or affected by the subpoena, quash or modify the subpoena or, if the party in whose behalf the subpoena is issued shows a substantial need for the testimony or material that cannot be otherwise met without undue hardship and assures that the person to whom the subpoena is addressed will be reasonably compensated, the court may order appearance or production only upon specified conditions.

(d) DUTIES IN RESPONDING TO SUBPOENA.

(1) (A) A person responding to a subpoena to produce documents shall produce them as they are kept in the usual course of business or shall organize and label them to correspond with the categories in the demand.

(B) If a subpoena does not specify the form or forms for producing electronically stored information, a person responding to a subpoena must produce the information in a form or forms in which the person ordinarily maintains it or in a form or forms that are reasonably usable.

(C) A person responding to a subpoena need not produce the same electronically stored information in more than one form.

(D) A person responding to a subpoena need not provide discovery of electronically stored information from sources that the person identifies as not reasonably accessible because of undue burden or cost. On motion to compel discovery or to quash, the person from whom discovery is sought must show that the information sought is not reasonably accessible because of undue burden or cost. If that showing is made, the court may nonetheless order discovery from such sources if the requesting party shows good cause, considering the limitations of Rule 26(b)(2)(C). The court may specify conditions for the discovery.

(2) (A) When information subject to a subpoena is withheld on a claim that it is privileged or subject to protection as trial-preparation materials, the claim shall be made expressly and shall be supported by a description of the nature of the documents, communications, or things not produced that is sufficient to enable the demanding party to contest the claim.

(B) If information is produced in response to a subpoena that is subject to a claim of privilege or of protection as trial-preparation material, the person making the claim may notify any party that received the information of the claim and the basis for it. After being notified, a party must promptly return, sequester, or destroy the specified information and any copies it has and may not use or disclose the information until the claim is resolved. A receiving party may promptly present the information to the court under seal for a determination of the claim. If the receiving party disclosed the information before being notified, it must take reasonable steps to retrieve it. The person who produced the information must preserve the information until the claim is resolved.

(e) CONTEMPT. Failure of any person without adequate excuse to obey a subpoena served upon that person may be deemed a contempt of the court from which the subpoena issued. An adequate cause for failure to obey exists when a subpoena purports to require a nonparty to attend or produce at a place not within the limits provided by clause (ii) of subparagraph (c)(3)(A).

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

CERTIFICATE OF SERVICE

I, David E. Moore, hereby certify that on February 4, 2008, the attached document was electronically filed with the Clerk of the Court using CM/ECF which will send notification to the registered attorney(s) of record that the document has been filed and is available for viewing and downloading.

I further certify that on February 4, 2008, the attached document was Electronically Mailed to the following person(s):

Mary B. Graham
Morris, Nichols, Arsht & Tunnell LLP
1201 North Market Street
P.O. Box 1347
Wilmington, DE 19899
MGraham@MNAT.com

James G. Gilliland
Theodore T. Herhold
Chad E. King
Robert J. Artuz
Eric M. Hutchins
Townsend and Townsend and Crew LLP
379 Lytton Avenue
Palo Alto, CA 94301
OracleEpicrealm@townsend.com

/s/ David E. Moore
Richard L. Horwitz
David E. Moore
Potter Anderson & Corroon LLP
Hercules Plaza – Sixth Floor
1313 North Market Street
P.O. Box 951
Wilmington, DE 19899-0951
(302) 984-6000
rhorwitz@potteranderson.com
dmoore@potteranderson.com

Exhibit A

EXHIBIT A

Pursuant to Federal Rule of Civil Procedure 45, and the definitions provided herein, epicRealm Licensing, LP, by its attorneys, requests that IBM produce by February 15, 2008 the documents described herein. The documents are to be produced at the New York offices of JENNER & BLOCK LLP, attention George S. Bosy, 919 Third Avenue, 37th Floor, New York, NY 10022-3908, or at such other place and time as may be agreed upon by the parties. In lieu of appearing at the place specified above to produce the documents described herein, the documents may be shipped in advance to the Chicago offices of epicRealm Licensing, LP's attorneys, JENNER & BLOCK LLP, attention George S. Bosy, 330 N. Wabash Ave., Chicago, Illinois, 60611.

DEFINITIONS

1. "IBM," "you," and "your" mean IBM, and all predecessors, successors, subsidiaries, divisions, parents and/or affiliates thereof, past or present, and all past or present officers, directors, affiliates, agents, employees, attorneys, consultants, representatives and any other person acting or purporting to act on behalf of IBM.

2. The term "document" or "documents" is to be interpreted in the broadest sense allowed under the Federal Rules of Civil Procedure.

3. "epicRealm" and "defendant" mean epicRealm, defendant and counterclaimant in this action, and all predecessors, successors, subsidiaries, divisions, parents and/or affiliates thereof, past or present, and all past or present officers, directors, partners, affiliates, agents, employees, attorneys, consultants, representatives and any other person acting or purporting to act on behalf of epicRealm.

4. "And" or "or" shall be construed conjunctively and disjunctively so as to acquire the broadest meaning possible.

5. "Any" shall be construed to include "any and all."

6. "Including" shall be construed broadly, to mean "including but not limited to" or "including without limitation."

7. "The 1996 Atlanta Olympic Games Website" means any web site created and operated by IBM for the 1996 Atlanta Olympic Games including the web sites at the domains www.atlanta.olympic.org and results.atlanta.olympic.org. (See, *e.g.*, "Load Distribution for Scalable Web Servers: Summer Olympics 1996-A Case Study" by German Goldszmidt and Andy Stanford-Clark from the IBM Watson Research Center)

DOCUMENTS REQUESTED

1. All documents relating to the architecture, design, programming, marketing, operation, success, costs, commercialization, publicity, performance and deficiencies of the 1996 Atlanta Olympic Games Website.

Exhibit B

EXHIBIT B

PLEASE TAKE NOTICE that on February 20, 2008 at JENNER & BLOCK LLP, 919 Third Avenue, 37th Floor, New York, NY 10022-3908, or at such other time and place as agreed to by the parties, epicRealm Licensing, LP, pursuant to Federal Rules of Civil Procedure 30 and 45, shall take the deposition upon oral examination of IBM. The deposition will continue from day to day until completed and shall be taken before a duly certified court reporter and notary public or other person authorized by law to administer oaths. The deposition will be recorded by stenographic, sound and video means.

Pursuant to Federal Rule of Civil Procedure 30(b)(6), and the definitions provided herein in Exhibit A, IBM shall designate one or more officers, directors, managing agents or other representatives who consent to testify on its behalf, to testify to matters known or reasonably available to IBM regarding the topics listed herein. To the extent more than one deponent is identified, IBM shall state in advance of the deposition which portions of this notice each deponent is prepared to address.

30(B)(6) DEPOSITION TOPICS

1. The 1996 Atlanta Olympic Games Website.

2. The architecture, design, programming, marketing, operation, success, costs, commercialization, publicity, performance and deficiencies of the 1996 Atlanta Olympic Games Website.

3. IBM's resources, including financial resources, invested in the 1996 Atlanta Olympic Games Website.

4. The following documents:

    a. G. Goldszmidt and A. Stanford-Clark. "Load Distribution for Scalable Web Servers: Summer Olympics 1996 - A Case Study." IBM Watson Research Center. (Exhibit 1).

    b. D.M. Dias, W. Kish, R. Mukherjee, R. Tewari. "A Scalable and Highly Available Web Server." (Exhibit 2).

c. P. Dantzig. "Architecture and Design of High Volume Web Sites (A Brief History of IBM Sport and Event Web Sites)." (Exhibit 3).

d. A. Stanford-Clark. "Atlanta Olympics WOMplex." (Exhibit 4).

e. "Managing the Web." IBM Think Research, http://domino.watson.ibm.com/comm/ wwwr_thinkresearch.nsf/pages/wom496.html. (Exhibit 5).

f. "Load-Balancing Internet Servers." IBM International Technical Support Organization, Austin Center. (Exhibit 6).

Exhibit 1

Load Distribution for Scalable Web Servers: Summer Olympics 1996 - A Case Study

Germán Goldszmidt and Andy Stanford-Clark
IBM Watson Research Center
{gsg, andysc}@watson.ibm.com

Abstract

The 1996 Atlanta Olympic Games were the first to have an official Web site. This site had more than 60 Web server hosts, spread across 5 locations in 4 countries, and registered over 192 million hits during the 17 days of the games. This paper describes some of the challenges of building, operating and managing such a large distributed system. In particular, it outlines the use of novel TCP/IP technologies for sharing the load of HTTP requests across local and global servers. Using these technologies enabled us to build a scalable distributed Web server site that was able to comfortably support a huge demand. We also describe some of the management tools used for monitoring the system, logging HTTP requests, and providing secure access to system resources.

1 Introduction

The 1996 Atlanta Olympic Games Web site, www.atlanta.olympic.org, was created and operated by IBM. The name *WOMplex* [1] was used as the umbrella code-name to describe a combination of technologies that included content and application servers, scalable parallel processing hardware, a content authoring environment, load sharing software, and several user-level applications and supporting sub-systems. The WOMplex contains collections of reusable objects that can be dispersed over the Internet. HTML pages are dynamically assembled using these objects.

The primary WOMplex was located in Southbury, Connecticut, and hosted the majority of the Web server nodes. The rest of the nodes were located in academic centers at Cornell (New York), Karlsruhe (Germany) and Keio (Japan), and at the IBM Laboratories in Hursley (UK). Each of these sites hosted mirror replicas of most of the content of the primary site. Some of the content was only served from the primary site, including the rapidly changing "Sneak Peek" images which provided a constantly changing montage of captured images from 38 video feeds from the Games in Atlanta. The WOMplex also delivered live audio from a local Atlanta radio station, and low bit-rate video and audio clips.

Before the games, some people wondered whether the Internet would "survive" the Olympics [6]. Bob Metcalfe, for instance, asked "Can IBM's Womplex handle Internet traffic of Olympian proportions?" [5]. The WOMplex technology, which was pioneered to support large sporting events on the Internet, proved adequate to cope with the demands made by so many visitors to the site.

[1] WOM is an acronym for "Web Object Management".

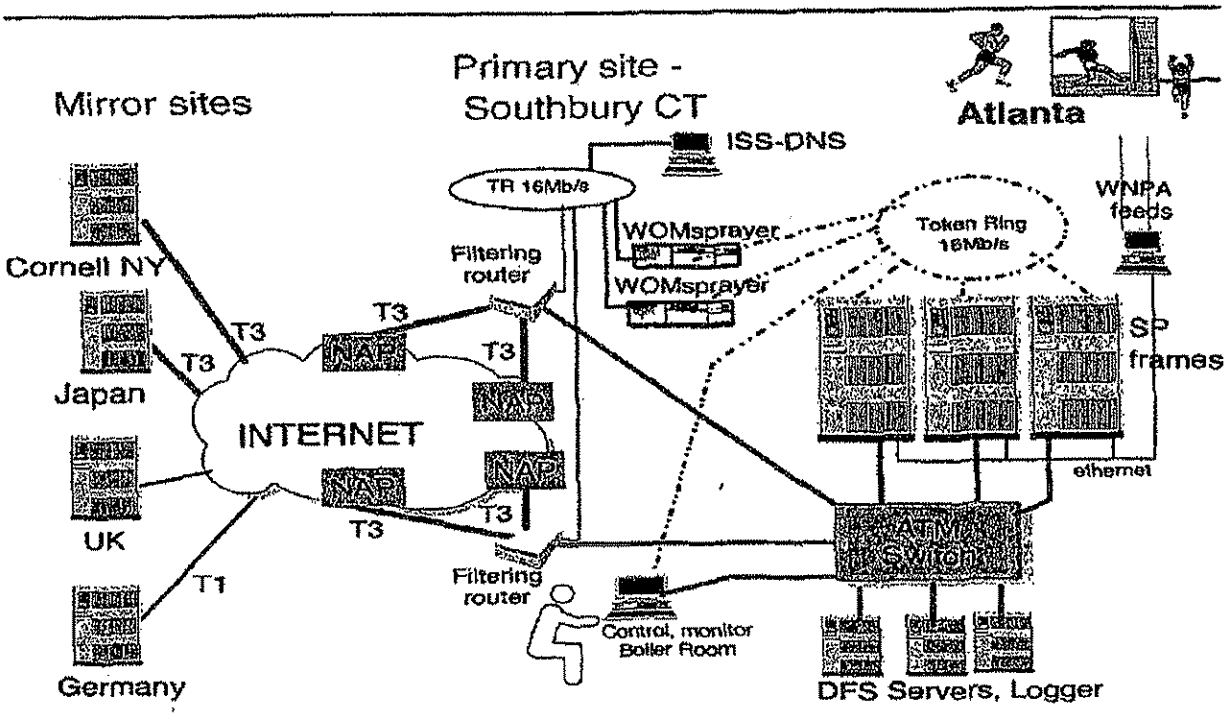


Figure 1: The Infrastructure of the WOMplex for the Atlanta 1996 Olympic Games.

This paper describes some of the challenges of building, operating and managing such a large distributed system. In particular, we describe the techniques and tools used to distribute the incoming Internet workload across a collection of servers, both locally at each location and globally across locations. At each site, incoming traffic directed towards a single IP address, is distributed across multiple back-end WOMserver machines. Between sites, a global load balancing scheme allows logical Web sites to be "moved" between physical server locations. The paper also outlines the WOMplex use of network and system management technologies for monitoring the state of the WOMplex, logging requests and securing access to its resources.

The rest of this paper is organized as follows. Section 2 describes the networking, hardware, and software infrastructure of the Atlanta 1996 WOMplex, Section 3 describes the WOM-DNS and WOMbat tools used to distribute requests across locations. Section 4 describes the WOMsprayer tool used to support distribution of the HTTP requests among the servers at each location. Section 5 outlines some of the approaches taken to deal with the problems of configuration, access security, logging requests, and monitoring in this complex distributed system, and Section 6 concludes the paper.

2 The WOMplex Infrastructure

The infrastructure of the WOMplex is shown in Figure 1. The Southbury site was connected to four major US Internet Network Access Points (NAPs) by DS3 circuits (T3 connections at 45Mb/s). These multiple circuits provided redundancy protection against failure of any of the links. Such failures occurred from time to time at various NAPs. When a link failed, the IP packets were either automatically routed over the

other links, or they could be manually routed round the trouble spots. The primary site was also connected to Atlanta by 56Kb/s leased lines which provided the feed of official games results by the *World News Press Agency (WNPA)* to the WOMplex databases.

Incoming HTTP requests from the Internet were forwarded via IP filtering routers to a WOMsprayer (Section 4) which then routed them to one of the servers. DNS requests were forwarded to a modified name server, WOM-DNS, (Section 3). The filtering IP routers, WOMsprayer and WOM-DNS were attached to an *"external"* 16Mb/s Token Ring.

The network bandwidth requirements for a Web site are highly asymmetrical - one incoming IP packet containing a single "GET" HTTP request can provoke the transfer of potentially many megabytes of outgoing data. We took advantage of this asymmetry by using two separate LANs for incoming and outgoing traffic. Incoming HTTP requests were forwarded by the WOMsprayer to the back-end servers via a second Token-Ring. Outbound traffic (HTTP responses) from the WOMplex servers were forwarded over a 100Mb/s ATM switch directly to the IP routers, and from there via the NAPs to the corresponding remote clients.

The infrastructure of the WOMplex was built around IBM RS/6000 Scalable Parallel systems (referred to hereafter as SP) at each of the mirror sites. Control access to the SP frames at the remote sites was arranged using secure Internet connections. Every processor in the WOMplex could be controlled - right down to power cycling - from Southbury. In the main site, all processors were connected to a Token Ring, an Ethernet, and an ATM network. Other sites used combinations of Ethernet, FDDI, Token Ring, and the SP Switch. Transarc's Distributed File System (DFS) was used to insure that all the servers at all the sites had access to the same data. A big advantage of DFS was the client-side caching: during the Olympics, most clients achieved a 98 % cache hit rate.

WOMplex separates logical services from physical servers, allowing a Web site to be hosted from any location in the WOMplex. IP aliasing is used within each location to separate logical servers from physical nodes. All externally visible IP addresses are "virtual", that is, they are aliases of several hosts. Each host also has its own private IP address, but these addresses are never visible to the outside world. In the event of a machine failure, the IP addresses that it supported can be readily transferred to another machine. This is done by adding IP aliases to the new machine, and flushing the ARP cache on the nearest IP router so that it is forced to re-establish the new IP to MAC address mappings. This operation can be automated using SNMP-based commands to the IP router.

3 Global load balancing

Two mechanisms were implemented to allow Web traffic to be effectively spread across the multiple sites comprising the WOMplex: WOM-DNS, and the WOMbat.

3.1 WOM-DNS

WOM-DNS is a load balancing DNS method that allows the DNS administrator to configure a table of Internet domain names to several IP address mappings. A WOMagent runs on each of the servers designated to serve any given name and collects metrics from each host. Incoming resolution requests for that name to WOM-DNS receive the IP address of the most appropriate server, according to the health, load, and the configured balancing policy for that name.

Two balancing policies can be configured: ROUND and FAIL. The ROUND policy does a weighted round-robin of the IP addresses in the list associated with that name, distributed according to two factors: the server activity profile, as determined from the inputs provided by the WOMagents, and a server weighting factor, which is used to make allowance for different server capacity at each site - e.g., different numbers of WOMserver nodes at each site. The FAIL policy attempts to use the first IP address in the list - the preferred site - until that site becomes unavailable for some reason. At that time it switches to a weighted round-robin of the remaining IP addresses in the list, the same as in the ROUND policy.

This system allows a single logical Web site (defined by a specific DNS name) to be distributed across multiple physical locations, each of which could be a cluster of servers controlled by a WOMsprayer. Additional servers can be dynamically switched-in, if they are required, to cope with excess load. Much of the flexibility of WOM-DNS (in contrast to standard DNS) is that the name server can be dynamically re-configured through both script commands and a graphical interface. WOM-DNS also enables automatic fail-over to other servers in the event of server down-time. This can either take the form of fail-over to other servers from a preferred, or primary, server; or the omission from the round-robin schedule of any of a set of servers which is unresponsive.

3.2 WOMbat

The WOMbat is a mechanism which uses ICMP ECHO packets (i.e. like the PING program) to echo-locate a client machine, and thereby determine which of the WOMplex sites it is nearest to, in network round-trip terms. With this knowledge, the WOMserver can then divert the client to its nearest server site, where they will hopefully receive faster service thanks to the closer network connectivity to that site.

When a client browser visits a WOMplex site for the first time, the client does not have a "Browser Cookie" for that site. The WOMserver generates a Cookie, (a unique key that identifies the client) and sends it to the browser for use whenever it visits that site. The Cookie is also stored in the *Global Profile Daemon (GPD)*, which is a database of profile information, keyed by browser Cookie.

At this time, the WOMserver also invokes, asynchronously in the background, the *WOMbat Driver*, before composing the HTML page requested by the client and returning it to them. The WOMbat Driver sends details of the client's IP address and its browser Cookie to a WOMbat daemon at each geographical site in the WOMplex. Each WOMbat daemon sends out a volley of ICMP echo request packets to the client IP address, and measures the round-trip time of any responses that return within a pre-specified timeout period.

Many client IP addresses are associated with proxy firewalls, which will typically return the ICMP echo response. Since such firewalls are usually located at the same site or in geographical proximity to the client, such responses are good approximations. Unfortunately, some sites may block the ICMP echo requests, e.g., via IP packet filtering routers. In this case, an approximation of the network distance could be obtained via other (slower) tools like *traceroute*.

After collecting the ICMP echo responses from the client's network, the WOMbat daemons send to the WOMbat Driver their average network response time. The WOMbat Driver waits for a specified amount of time for the WOMbat daemons to respond, and then determines which WOMserver site is closest to the client by comparing their responses. Server availability information can also be factored into this calculation, so that the closest site might be rejected in favour of a more distant but less busy site. The identity of the preferred site is stored in the user's profile in the GPD.

Next time that a client browser requests a page, the WOMserver interrogates the

GPD, using the browser Cookie as a key. If the preferred site is the local site, the requested page is built with embedded local links and returned to the client. But if the preferred site for that client is not the local site, the page is built with all embedded URL links pointing to the preferred site.

Assume, for example, that the preferred site for a French client was given by the URL `www.hursley.atlanta.olympic.org`, and a local link containing data that is mirrored to all sites would normally have been: `<IMG SRC="/data/i.gif">`. Then the link that is sent back to this client is re-written as:
`<IMG SRC="http://www.hursley.atlanta.olympic.org/data/i.gif">`. When the client browser follows any of these URL links, they are retrieved from the preferred site, and from that point on, all mirrored URL links will point back to that site. Therefore, the service for this client has effectively been "moved" to the preferred site.

4 Local Load Sharing

The purpose of load-sharing is to reduce the average response time by smoothing out transient peak overload on some of the nodes of a distributed system [4]. The WOMsprayer is a software TCP connection router, that serves as a load-sharing front-end to a collection of back-end servers. The WOMsprayer host offers a collection of *Virtual IP Addresses (VIPA)* to the outside world. For example, the primary site exported 2 different VIPAs, one for www.atlanta.olympic.org, (204.146.176.111) and another one for results.atlanta.olympic.org (204.146.176.113). These VIPAs are shared by a dynamically changing set of servers which alias some or all of these addresses but do not advertise them via ARP. The WOMsprayer allocates requests based on their port numbers. For instance, some servers of www.atlanta.olympic.org supported only port 80, others only port 8080, and some supported both. The allocation of servers and ports was dynamically configurable.

All requests arriving to these VIPAs were routed via the WOMSprayer. When a new TCP connection request is made to any VIPA, the WOMsprayer chooses one server among the set of active back-end servers for the particular `<VIPA, port>` combination. The WOMSprayer modifies the MAC address field of the incoming IP packet to that of the chosen server's interface card, and then forwards the packet via the corresponding LAN (e.g. the internal Token Ring). The IP packet itself is not changed.

The server interface will pick up the connection request IP packet. The server then does the required accept to create the TCP connection, returning a response IP packet directly to the remote client. The network path from the server back to the client (e.g. Web browser) is via the back-end server's default route, (e.g. the ATM network). Hence the returning server-to-client packets are not intercepted by the WOMsprayer. The WOMsprayer maintains a table of currently active connections, and any further incoming packets for that TCP session, such as packet acknowledgements, are forwarded to the corresponding back-end server. Notice that the WOMsprayer only sees the client-to-server half of each TCP connection. Hence it must handle several cases of connection termination by timestamps and garbage collection.

The WOMsprayer mantains several counters that summarize the state of the connections, e.g., the number of active connections for each server. These counters, together with a set of configuration variables, could be used to prepare an SNMP MIB for monitoring the WOMsprayer. We used a combination of scripted commands, which could also be invoked from a Web browser. One of the main uses of the configuration commands was to temporarily add and remove servers from servicing a giving IP address. This allowed to dynamically upgrade the software and content of the servers without disrupting the service and without dropping existing connections.

A probabilistic weighted round-robin algorithm, implemented as part of the kernel extension to the TCP/IP stack, chooses the back-end server for each incoming TCP connection. The weights associated with each server correspond to the proportion of new connection requests that will be allocated. These weights are dynamically assigned in real-time by a user-level process, the WOMsprayer *Manager*, based on current state information. The WOMsprayer Manager takes into account several sources of information about the state of the servers, including feedback host metrics, local counters for input metrics and service metrics. Input metrics provide a current estimate of the rate of incomming connections, e.g., number of new connections in the last t seconds. Host metrics represent the load of a server, and are based on system-wide estimates, like the number of active processes or the number of allocated mbufs. Service metrics are specific to each application, for instance, the time measured to retrieve an HTML page.

When the algorithm finds that there is an imbalance in the allocation of resources it changes the weights to correct the problem. This allocation method implements a dynamic feedback loop that automatically controls the proper spread of the service requests. During the Olympic games, it reacted effectively to the bursty nature of the requests traffic, discovered and routed around server failures, and was very efficient in balancing the load among all the active servers.

The two main strategies for improving performance using load distribution are *load-balancing* and *load-sharing* [1]. Load-balancing strives to equalize the workload among servers. Load-sharing attempts to smooth out transient peak overload periods on some nodes [3]. Load-balancing strategies typically require many more resources and are better suited for large batch jobs. The WOMplex HTTP requests were a mixed load, with many short requests for a page and some very long requests (e.g. radio). The resources necessary to achieve an approximate load-balance among the WOMplex servers would have been so high that they would have outweighted any potential benefits. Hence we designed the WOMsprayer Manager to provide effective and efficient load-sharing.

Notice that at any particular point in time, there maybe some imbalance in the utilization state of the servers. This is unavoidable as the processes that consume resources (e.g. dynamic page creation) work much faster than the WOMsprayer feedback. To forward feedback fast enough to observe all relevant events would have required too much cycles and bandwidth, far outweighting any potential benefits. Still, the algorithm was very sensitive to host overload and achieved good load balancing. The local WOMsprayer feedback provided by the connection counters gave a sufficient indication of load with low overhead. Whenever a server host became overloaded ("over-heated"), it was quickly quiesced, giving it time to "cool-down" (i.e. complete all in-progress requests). When the server host was again relatively "cool" (usually a few seconds later), it was automatically returned to the working set. Any server that becomes overloaded has a large impact on the quality of service of the site, as the number of enqueued requests grows quickly. Therefore, automatically removing, cooling and returning a server to service was of paramount importance.

Similarly, any server which experienced failure problems was dynamically and automatically excluded from the WOMsprayer schedule, and no new connections were routed to it until the problem was resolved. For instance, a server may develop some hardware or software problems that resulted in connection requests accumulating in its queue. The WOMsprayer would automatically detect that the connection count for that server reached a high threshold and react by lowering the weight of the server, even to 0. Hence, this method effectively prevents incoming traffic from hitting a non-responsive server and getting "Connection Refused" error messages.

Each server host runs a WOMagent daemon, which executes a host metric script,

which determines how busy, in some sense, the server is. The metric can be any script that returns an integer. For example, the metric "`ps -ef | wc -l`" reports the number of processes currently running on the system. More elaborate metrics can be contrived, to evaluate other host performance limits. One server host metric that was used for the Olympics counted the number of ESTABLISHED socket connections on port 80 of each server. Of course, compiled programs can be used to generate the metric values. The WOMagents periodically report to a central WOMmonitor process, which collates agent responses, starts and stops the remote WOMagents, and reacts appropriately if the WOMagents do not report-in for an extended period of time. The WOMmonitor transfers a load profile of server activity to the WOMsprayer Manager, which uses it to compute the weights for each server. Any server whose WOMagent is not responding is removed from the load profile, and hence is given a non-active weight by the WOMsprayer.

For reliability, the WOMplex at the primary site included two WOMsprayers, but only one of them was active at any given time. If the active WOMsprayer host failed, the filtering router ARP table would be updated so that new requests will arrive to the secondary WOMsprayer. A detailed presentation of the *NetDispatcher* technology that was used in the WOMsprayer is given in [2]. A theoretical qualititative and quantitative analysis of adaptive load sharing algorithms is given in [3].

5 Operational Management

Operational Management of the WOMplex involved configuration of all the sites, monitoring and control of all subsystems, logging and securing all operations and systems.

5.1 Remote Configuration and Control

Network Configuration. The global configuration of the Web site required keeping configuration files for the WOMsprayers, DNS servers, and the IP address aliases of more than 80 server nodes. This was achieved using a master database, where the configuration rules where defined (e.g. which site name on which IP address was going to be served by which subset of nodes at which site). A program then took this data and built the set of configuration files, customised to each machine as required. This customisation was defined by site-specific rules, which included site-specific DNS suffixes and IP addresses. The configuration files, once built, were than copied using an encrypted protocol across the internet to the target machines. When all files were in place, a series of remote shell operations (also encrypted) triggered the various daemons to re-read the new configuration files.

Control. Operators in the Web Operations Centre (WOC) in Southbury, CT, USA, were able to control all physical aspects of the SP2 frames located across the world, through the X-windows based GUI to PSSP, a system control package. Most usefully, this gave us the ability to reset or power-cycle nodes, without having to rely on anyone being physically present at the remote site. With an operation of this size and complexity, it was essential to have everything controlled by scripts, partly so that they could easily be re-used in other situations, and partly for auditability: if the operation failed, we could look at the script to see exactly what sequence of steps had been performed.

5.2 Monitoring

System Logs. All Unix system logs (systlogd) were channeled to a single machine, where a set of filters were applied to the reams of information to make sure that the ones which were important were displayed on the screen and were not lost in a morass of trivial output. We also attached a console beep to the most important messages, which effectively gave us an audible alert when something critical was happening, such as a detected intrusion onto one of the systems, or a port scan in progress, or critical failure of hardware or software.

Performance. We positioned HTTP monitors at strategic locations round the world to frequently request pages from our various servers, and to measure the response time. After comparison with other major sites on the Internet, an alarm would be raised (Email to a pager) if a certain threshold was exceeded on the response time. This mechanism brought problems to our attention even when from our view (the server end of the operation) everything was looking good.

5.3 Security

TCP/IP Security. Such a high visibility site was an obvious target for hackers, and hence security was a major concern. The IBM Global Security Lab (*the "Ethical Hackers"* team) performed thorough penetration testing of the site before the games, and provided valuable advice on security procedures. Filtering IP routers provided the main protection against invasion. Inter-working with the remote mirror sites involved setting filter rules at the remote sites which would allow only certain IP packets in and out from certain places. The primary site (Southbury) had to have extensive access to systems inside the DMZ (De-Militarized Zone) at the remote sites, but the rest of the Internet could only have HTTP port 80 access. Also, the trusted mirror sites had to have access to hosts inside the Southbury DMZ.

DNS Security. WOM-DNS as the primary name server provided another level of security. WOM-DNS was configured to have authority over one or more specified sub-domains. A name resolution request may arrive for either a sub-domain over which WOM-DNS does not have authority or for a name which is not being used for load-balancing. In both cases, WOM-DNS passes the request over to one of two back-end, ordinary, name servers. WOM-DNS is configured with a table of sub-net specifications for "trusted" domains. The name resolution request is matched against this table. If there is a match, then the client is deemed to be internal, and is passed to one back-end DNS server, otherwise it is deemed to be external and is sent to the other DNS server. Even with a distributed, multi-site operation, it is possible for the client machines that need to have access to all of the main site's internal server names and addresses to do so. At the same time, the rest of the Internet is only able to resolve the limited set of names and addresses which the DNS administrator chooses to make available. The WOM-DNS also blocks the DNS Zone Transfer operation, which is a popular method of gaining access to lists of internal machines to be explored as hacking targets.

5.4 Centralized HTTP Logging

An important aspect of WOMplex operation is the collection of HTTP access and error logs from the servers distributed across the sites, and bringing them to a central location for monitoring and analysis. The *Grim Reaper* tool is a tree-structured "plumbing" of

sockets and Unix pipes. This tool streamed log records from each server to a confluence point at each site, and then out across the Internet to a central collection point on a primary site machine called the *Logger*. When all the log data is streaming nicely, the task of the Grim Reaper is quite straightforward. The problems start occurring when one of three things happens: (1) the servers produce log data faster than the local Grim Reaper can take it from them; (2) network delays or failures cause hold-ups in the Grim Reaper plumbing which cause a back-log of data; and (3) the final receiver of the logs can not cope with the rate at which they are arriving from the Grim Reaper. Any of these scenarios, coupled with the simple mandate that no log records may be discarded, make the Grim Reaper's job difficult.

At any stage, log streams may be diverted into a file on local disk, to hold data which for some reason can not be sent onwards. Later, when connections are restored, or congestion eases, the files are spooled out back into the network. The Grim Reaper also supported off-line transfer of batched-up log data (for example late at night when there was less network traffic), and also on-the-fly compression of data for transmission across slower network connections. After arrival at the logger machine, the log streams were transferred across a high-speed link to a parallel mainframe where they were pre-processed by a series of filters and then the individual log records were added to a DB2 database. From here, SQL queries could be used to extract useful information from the logs, including hit counts and rates, online data mining, and real-time Click Tracking. The raw log data was also spooled to tape for later analysis.

A good indication of the site performance as a whole was given by a large screen with dozens of windows displaying the httpd log files from a subset of the servers. If these windows were all in motion, then everything was good - the WOMplex was serving pages across all our server locations. If any window stopped, or if the whole screen froze, then we knew there was trouble: most often a network problem which stopped people from reaching some part of the WOMplex.

5.5 The WOMplex Boiler Room

The Boiler Room system allowed statistics to be gathered from various parts of the WOMplex, its applications, and sub-systems. The system was structured as follows. Each application to be monitored was instrumented with a funneld (funnel daemon) client. The protocol used throughout this system was IRC (Internet Relay Chat), transmitting data in a textual format, with each monitored system using a different IRC channel number. The client code connected to a funneld instance, running on each server. The funneld instances connected to other funneld's in a tree structure. At the root of the tree of funnel daemons was an IRC server.

The Boiler Room monitors were Java applets embedded in HTML pages on the WOMplex Web site. These applets used IRC client classes, written in Java, which connected back to the IRC server machine. The monitor applications then tuned-in to the IRC channels that they required to construct their displays, and waited for information to arrive. A variety of dials, counters and indicators were implemented as Java classes which could be combined in various ways to display the statistics for any system. The Boiler Room system also had facilities for tailing log files, and receiving data through sockets and IPC message queues, so it was easy to hook up monitors for almost anything. A special chart recorder client was used to log IRC channel data for subsequent analysis.

Boiler Room monitors delivered vital statistics on the loading and health of the various servers in the system. The monitoring effort was distributed amongst a number of people, as there were too many moving bars and flashing lights for any one person to take in for long periods of time. Since the Olympics, a lot of our efforts have been ap-

plied to making the management of the WOMplex-type infrastructure more integrated, using Tivoli Managed Environment products. To permit cost-effective 24x7 operation, a greatly simplified monitoring system is essential, with all commonly required operations fully scripted and automated wherever possible.

6 Conclusions

Exponential growth in Internet traffic and the increasing complexity of CPU-intensive Web-based applications, pose increasing challenges to popular Web sites. As many main Internet links (e.g. across the Atlantic) are saturated for long periods of time, geographic distribution is necessary to serve an international audience with adequate response times. Anticipating such problems for the Olympic Games Web site, we devised a combination of technologies based on TCP connection routing and name resolution load balancing. The combined technologies enable almost unlimited scalability both at the local site level (LAN) and across multiple geographical locations (WAN). These technologies were crucial in the successful global deployment of the Atlanta 1996 Olympic WOMplex, which served a record number of requests world-wide.

Operating and managing this large and complex distributed site was a very challenging task. It was essential to have everything controlled by scripts, that could easily be re-used and provided an audit trail. Automated notification of critical events by filtering of system logs enabled a small group of operators at the main site to manage the entire WOMplex. The efficiency of the WOMplex management components and tools was crucial to the success of the global Web site. Since this event, many of these technologies have been upgraded and succesfully integrated and used at many other large-scale Web sites worldwide.

Acknowledgments. The success of this Web site was due to the combined efforts of many people. Unfortunately, we cannot list all of them in the space available here, so we only acknowledge those who worked most closely with us in the technologies described in this paper. These include Michael Factor, Ed Geraghty, Dave Grossman, Jane Harper, Guerney Hunt, Sean Martin, Chet Murthy, Scott Penberthy, Frank Schwichtenberg, Graham Wallis, and Ron Woan.

References

[1] D. L. Eager, E.D. Lazowska, and J. Zahorjan. Adaptive Load Sharing in Homogeneous Distributed Systems. *IEEE Transactions on Software Engineering*, 12(5):662–675, May 1986.

[2] Germán Goldszmidt and Guerney Hunt. NetDispatcher: A TCP Connection Router. Technical report, IBM Research, Hawthorne, New York, July 1997. RC 20853.

[3] O. Kremien and J. Kramer. Methodical Analysis of Adaptive Load Sharing Algorithms. *IEEE Transactions on Parallel and Distributed Processing*, 3(6), November 1992.

[4] O. Kremien, J. Kramer, and J.N. Magee. Scalable and Adaptive Load Sharing for Distributed Systems. *IEEE Parallel and Distributed Technology*, 1(3), 1993.

[5] Bob Metcalfe. Can IBM's Womplex handle Internet traffic of Olympian proportions?, July 1 1996. InfoWorld.

[6] Bob Metcalfe. Summer Olympics to use IBM's 'extranet' 390s, but beware the Ides of July, April 8 1996. InfoWorld.

Exhibit 2

A Scalable and Highly Available Web Server

Daniel M. Dias William Kish* Rajat Mukherjee and Renu Tewari

IBM Research Division
T. J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598

{*dias, c1kish, rajatm, c1renu*}*@watson.ibm.com*

Abstract

We describe a prototype scalable and highly available web server, built on an IBM SP-2 system, and analyze its scalability. The system architecture consists of a set of logical front-end or network nodes and a set of back-end or data nodes connected by a switch, and a load balancing component. A combination of TCP routing *and Domain Name Server (DNS) techniques are used to balance the load across the front-end nodes that run the Web (httpd) daemons. The scalability achieved is quantified and compared with that of the known DNS technique. The load on the back-end nodes is balanced by striping the data objects across the back-end nodes and disks. High availability is provided by detecting node or daemon failures and reconfiguring the system appropriately. The scalable and highly available web server is combined with parallel databases, and other back-end servers, to provide integrated scalable and highly available solutions.*

1 Introduction

With the explosion of traffic on the World Wide Web [3, 12], the load on popular servers on the web is increasing rapidly. In this paper we describe and analyze various methods for building a scalable web server on a cluster of computing nodes, in which the throughput of the cluster can be increased by growing the size of the clustered system. We analyze the limits to scalability and the efficiency of these methods, and the trade-offs between the alternatives. As both the traffic and the commercial applications on the popular web servers increases, high availability of these servers becomes increasingly important. Given a clustered web server, we describe how the server can be made highly available, by taking over the load of a failed node by the remaining nodes in the cluster. Based on these methods, we describe a prototype of a scalable and highly available web server that we have built on an IBM Scalable Parallel SP-2 system.

*Consultant from Brown Cow Engineering Inc. He can be reached at kish@browncow.com

1.1 Relation to Previous Work

The NCSA prototype scalable HTTP server is described in [8, 10]. The method they use, illustrated in Figure 1, consists of having a set of HTTP servers on nodes in a cluster, that use the Andrew File System (AFS) [6] for sharing a set of HTML documents, and using the round-robin Domain Name Server (RR-DNS) [2, 13] for distributing accesses among the nodes in the cluster. Essentially, clients are presented a single name associated with the set of HTTP servers, and the RR-DNS maps this single name to the different IP address of the HTTP servers, in a round-robin manner; thus different clients will (ideally) be mapped to different server nodes in the cluster. In this way, the load is distributed among the servers, each of which access the same set of URLs through the distributed file system (AFS in this case).

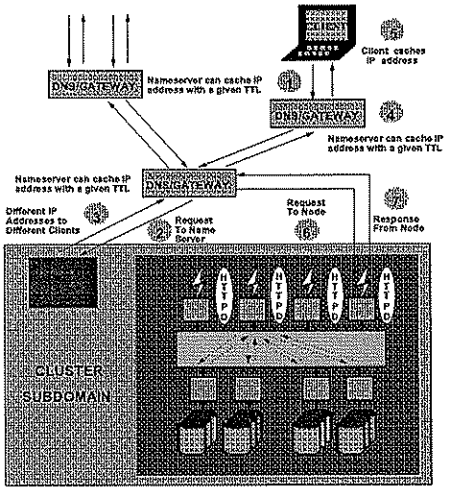


Figure 1: Domain Name Server: Flow

There are several problems that arise with this method. First, as illustrated in Figure 1, there are typically several name servers between clients and the RR-DNS that cache the resolved name-to-IP address

mapping. In order to force a mapping to different server IP addresses, the RR-DNS can specify a *time-to-live* (TTL) for a resolved name, such that requests made after the specified TTL are not resolved in the local nameserver, but are forwarded to the authoritative RR-DNS to be re-mapped to the IP address of a different HTTP server. Multiple name requests made during the TTL period will be mapped to the same HTTP server. Thus, bursts of requests from new clients can appear at the same HTTP server, leading to significant load imbalance. If the TTL is made very small, there is a significant increase in network traffic for name resolution. Therefore, name servers often impose their own minimum TTL, and ignore very small TTLs (e.g., 0) given by the RR-DNS. A second problem is that clients cache the resolved name-to-IP address mapping (henceforth referred to as address caching). Since the clients may make future requests at any time, the load on the HTTP servers cannot be controlled subsequently and will vary due to statistical variations in client access patterns. Further, clients make requests in bursts as each web page typically involves fetching several objects including text and images, and this burst is directed to a single server node, increasing the skew. We show that these effects can lead to significant dynamic load imbalance, requiring that the cluster be operated at lower mean loads in order to be able to handle peak loads. Another issue relates to load balancing is the distributed file server used by the HTTP servers. Typically, there is a skew in the file accesses, with some URLs being more popular than others, leading to a skew in the file server nodes. This becomes more significant when large documents are accessed and file system caching does not suffice.

The remainder of the paper is organized as follows: In Section 2 we present a scheme that combines the RR-DNS with a set of so called *TCP routers* [14] to better balance the load among the HTTP servers. This combination also allows for better scalability. We also describe a method that stripes the files across (data server) nodes in the cluster [5, 4, 16] thereby balancing the data access load. In Section 3, we quantitatively examine load balancing using the RR-DNS approach, and quantify the performance and scalability of the web server that we have prototyped. In Section 4 we describe how high availability is achieved in the prototype. Finally, concluding remarks appear in Section 5.

2 Scalable Web Server Architectures

The scalable web server architecture is illustrated in Figure 2, and consists of multiple web servers running on a clustered architecture. A clustered architecture consists of a group of nodes connected by a fast interconnection network, such as a switch. Each node in the cluster has a local disk array attached to it. The disks of a node can either maintain a local copy of the web documents or a share it among the nodes. For better performance and load balancing, the shared web documents are striped across multiple nodes. As shown in Figure 2, the nodes of a cluster are divided into two logical categories: (i) front-end

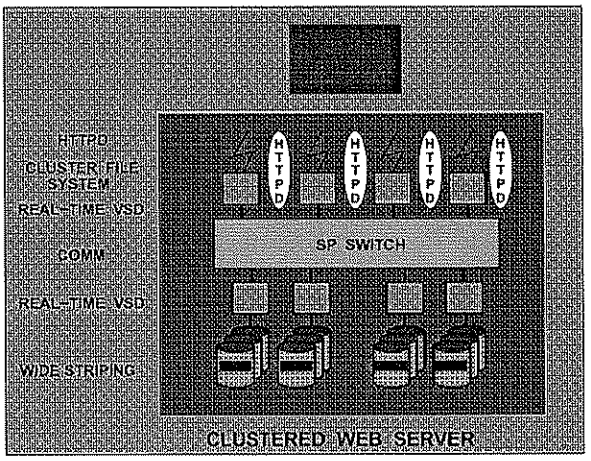


Figure 2: Scalable Web Server Architecture

(delivery) nodes and (ii) back-end (storage) nodes. A load balancing component is used to distribute incoming requests from the external network to the front-end nodes, which also run httpd daemons. The (logical) front-end node then forwards the required commands to the back-end nodes that have the data (document), using a shared file-system. We assume an underlying software layer (e.g., virtual shared disk) which makes the interconnection architecture transparent to the nodes [1]. The results are then sent back from the back-end node to the front-end node through the switch and transmitted to the user.

The clustered architecture outline above can be applied to several applications, and is similar to the design of a scalable video server. Essentially, the same architecture applies to a video server as well, with the httpd daemon replaced by a daemon that reads video objects, and plays them out to clients across the network at the required rate. For both web and video servers, two possible configurations can be used: (i) two-tier and (ii) flat. In the two-tier architecture of Figure 3, the logical front-end and back-end nodes are mapped to different physical nodes of the cluster and are distinct. In a flat architecture, on the other hand, each physical node can serve as both the logical front-end and back-end [16]. All nodes are identical, performing both storage and delivery functions. For a clustered web server we will principally consider the two tier architecture only.

2.1 Load Balancing

A key requirement in order to achieve scalability of the web server is that of balancing the load across the multiple front-end nodes and the back-end nodes.

2.1.1 Front-End

The front-end nodes run the web daemons and are connected to the external network. To balance the load among them, the client requests need to be spread evenly across nodes. Several methods can be used to

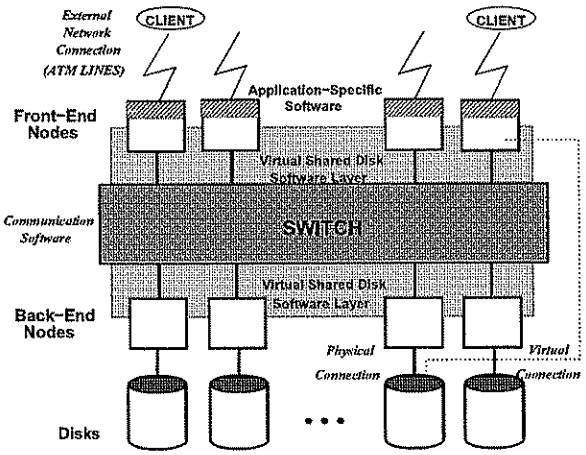


Figure 3: **Two-tier Server Architectures**

achieve this, with varying degrees of effectiveness. In Section 1, we outlined the round-robin domain name server (RR-DNS) based solution similar to the NCSA approach. To summarize, in this scheme all the front-end nodes are given a single logical name, and the RR-DNS maps the name to multiple IP addresses. However, due to address caching in the client, the bursty nature of client accesses, and TTL values imposed by nameservers, the load balancing achieved is coarse, as quantified in Section 3.

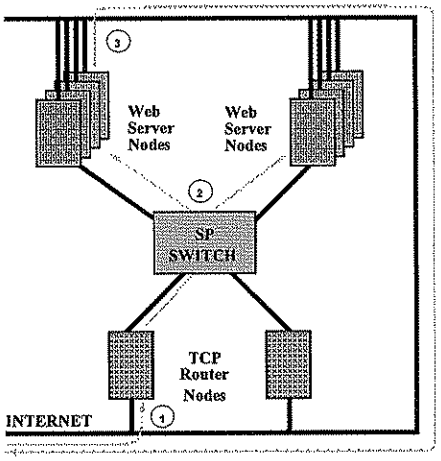


Figure 4: Scalable Web Server (Router): Flow

Another method for achieving load balancing is based on routing, and is illustrated in Figure 4. One or more nodes of the cluster serve as *TCP router(s)*, forwarding client requests to the different front-end nodes in the cluster in a round-robin order. The name and IP address of the router is public, while the addresses of the other nodes in the cluster is private. (If there is more than one router node, a single name is used and RR-DNS is used to map the name to the multiple routers.) This has been referred to as the *encapsulated cluster* in [14]. The client sends requests to the router node which in turn forwards all packets belonging to a particular TCP connection to one of the server front-end nodes. The router can use different algorithms based on load to select which node to route to, or use a simple round-robin scheme. In the prototype (see below), the router resides just above the IP layer as a kernel extension. The server nodes directly send the request back to the client without using the router. However, the server nodes change the source address on the packets sent back to the client to be that of the router node. This makes the entire routing action transparent to the clients. Note that, the response packets are large compared to the request packets, especially for images and video clips, and these bypass the router. Thus, the overhead added by the router is small (and is quantified later).

One advantage of the router scheme over the DNS based solutions is that fine grained load balancing can be achieved and there is no problem of client or name-server caching. One potential issue with both the RR-DNS and routing approaches is that the router node could become a bottleneck, as could the domain name server. These performance aspects are quantified in Section 3.

To further improve scalability, the router and DNS schemes can be combined in various ways. First, a number of router nodes can be used, as shown in Figure 4, and the RR-DNS method can be used to map different clients to different router nodes. This hybrid scheme can tolerate the coarse grained load balancing achieved using RR-DNS because the corresponding router will route any burst of requests, that were mapped by the RR-DNS to the same router, to different front-end server nodes. It achieves good scalability because (i) a long TTL can be used so that the node running the RR-DNS does not become a bottleneck, and (ii) several router nodes can be used achieving scaling beyond that of a single router.

Another hybrid router-DNS scheme combines the logical router and front-end nodes: The DNS based solution is used to get the coarse grained load balancing. Each front-end server node then acts as a router; if the load on the server is low, then the request is handled locally at that server, otherwise the router forwards the request to another node based on its load or a global load balancing algorithm. This scheme eliminates the forwarding by the router unless the load on the front-end node requires it.

2.1.2 Back-End

The back-end nodes function as the server nodes for the shared file-system (e.g., AFS, NFS etc.) that is used by the front-ends to access the data. For instance, the NCSA prototype [8] uses AFS for the data server. With traditional shared file systems, the total document set is partitioned among the different back-end servers. The partitioned approach could lead to load imbalances based on the access skew among the documents. Back-end servers that store the *hot*

documents will get overloaded, resulting in hot spots. The file-system caching at the front-end could alleviate this problem for small documents that fit in the local cache. Only the initial requests need to go to the back-end server and the remaining can be serviced from the local front-end cache. For larger documents, however, the back-end server nodes will be unevenly hit.

To balance the load among the back-end nodes different methods ranging from replication to striping can be used. If the data is replicated across each back-end nodes, the logical front-end and back-end can be mapped to same physical node making it a flat architecture. The scheme described for front-end load balancing can be used directly. Full replication of data is expensive in terms of space utilization. Also, if the data is modified frequently, some form of consistency control is required.

Instead of replication, each document is divided into logical blocks, which are then distributed among the disks in the system. The logical block represents the size of the striping unit, and could be mapped to multiple packets within the switch communication layer and to multiple physical blocks on disk (we use the term block, to refer to a logical block). The blocks belonging to a document may either span all the disks in the system (referred to as *wide striping*) or may be confined to a smaller subset of disks (referred to as *short striping*). The block sizes are typically large (> 64KB) to minimize the access latencies. Successive blocks of an object are allocated to consecutive disks using a sequential placement. The shared file system has additional support to access the striped documents.

Wide striping implicitly achieves higher disk-arm bandwidth and load balancing [5, 16]. However, small documents on the server cannot be striped across all the disks in the system. Such documents are striped only on a subset of the disks in the system and replicated on the remaining disks sets. Unlike short striping, however, in a server that employs wide striping, a single disk or node failure affects all the streams being accessed. Back-end node failures can be masked using the availability schemes described in Section 4.

3 Performance and Scalability

In this section, we study the performance issues relating to the scalability of cluster-based solutions using the RR-DNS and router. The performance of the RR-DNS was measured independently from that of the web server. Web server performance was derived from the Webstone benchmark [18], and the RR-DNS performance was measured via a stand-alone RR-DNS benchmark. Studies of the load imbalances using the RR-DNS and router strategies were performed via CSIM [15] simulations of a networked system.

3.1 Load Imbalance Due to Client and Network Effects

We have qualitatively discussed the factors that affect load balancing across nodes in Section 2. To quantitatively study the effect of dynamic load imbalance we simulate a networked system and vary the number of web page requests per client session, TTL at the nameservers and the number of nodes in the server. The simulation uses a burst model and a single level of caching nameservers between the clients and the server. For each page requested by a client a burst of small requests, representing the objects within a page, are sent to the server. Each client resolves the name of the server only once during a session. The number of remote nameservers is typically much larger than the number of nodes in the server. We illustrate the impact of the TTL with a single level of caching nameservers. The simulation parameters are shown in Table 1.

Parameter	Default
Number of Nodes	16
Number of Nameservers	150
Levels of Caching Nameservers	1
Clients per node	80
Server node capacity (requests are of the same size)	50 req./sec
Mean interarrival time (between page requests per client)	15 secs.
Mean number of bursts per page	10
Mean # of Page Requests/client session	20
TTL	300 secs.

Table 1: **Simulation Parameters**

Figure 5 shows the effect of varying the number of requests per client session on the dynamic load imbalance at the server for different TTL values at the nameservers. Each request to the server is assumed to be of the same size so that the load imbalance observed is not due to varying request size. The dynamic load imbalance is measured as a ratio of the maximum number (99% percentile) of requests serviced by a node in any sampling interval to the mean computed over the total simulation time period. (Note that, both the RR-DNS and router schemes lead to good average load balance across the nodes over an extended period of time. We focus on the dynamic load balance which can lead to very different peak loads on nodes in the cluster using the two schemes.) The inter-arrival times between page requests per client session is derived from an exponential distribution with a mean of 15 seconds. For a TTL value of 0, as the mean number of page requests per client session increases from 1 to 20, the effect of client address caching becomes more pronounced, resulting in a load imbalance increase from 8% to about 30%. Note, each page requests consist of a burst of requests (mean of 10) to the server. With an increase in the TTL value from 0 to 300 seconds, without client address caching (number of page request per client is one), the load imbalance increases from 8% to about 33%, sharply at first and then gradual. The effect of increasing TTL at the nameservers or that of increasing the duration of a client session is adverse on the load imbalance

at the server nodes. The router approach results in a much lower load imbalance of about 8% caused by the bursty nature of client requests and random arrivals.

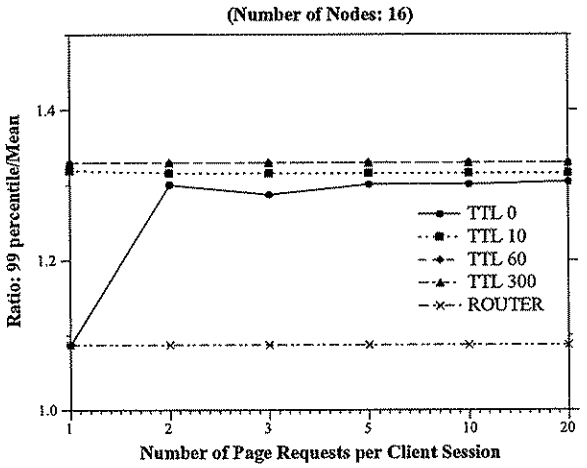


Figure 5: Load Imbalance of RR-DNS

The load imbalance is also a function of the number of nodes in the web server. As the number of nodes increases the load imbalance using the router decreases. Figure 6 shows the effect of varying the number of nodes on the load imbalance using the RR-DNS and router approaches. When the number of nodes is small (e.g., 2) the load imbalance due to the bursty nature of the requests and client random arrivals is high for both the router (22%) and the RR-DNS (30%). As the number of nodes increases, the effect diminishes for the router as it forwards the requests to different nodes. For the RR-DNS approach however, the load imbalance due to client and nameserver address caching contribute largely to the load imbalance and it gradually increases (31%) as the nodes are increased. Thus, for a large scale system a router based approach is recommended as the load imbalance is significantly smaller for the router than the DNS approach. For smaller systems both approaches are viable. Also in the simulation study each client randomly selects a nameserver, whereas in practice some nameservers may be much more utilized than others, causing larger imbalances using the RR-DNS approach.

3.2 Scalability of RR-DNS and Router

Figure 7 shows the scalability achievable via the RR-DNS solutions for different workloads (file sizes) for different ratios of new accesses to total accesses made to the web server. The RR-DNS is accessed only upon requests from new clients, since the address of the web server node is cached at the client after the first access. We varied the percentage of new accesses from 100% (every access is from a new client) to 1% (a client makes a hundred requests to the web site before retiring). Since every document typically consists of several constituent objects, the latter number reflects a few pages accessed per client.

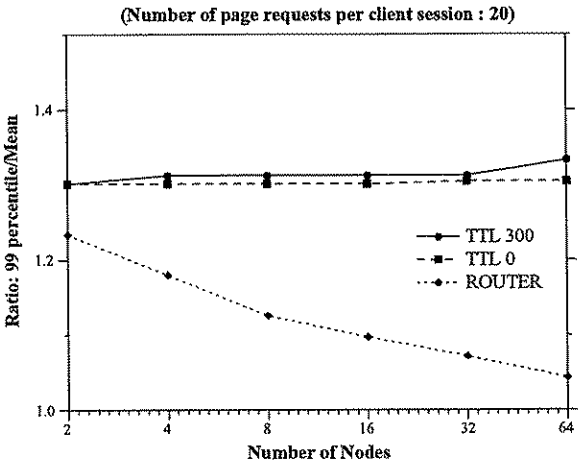


Figure 6: Effect of System Size on Load Imbalance

Since each node can handle fewer requests per second when the documents are large, we see that a single RR-DNS can handle the request to a larger number of server nodes when the average file size is larger. In absolute terms, the RR-DNS scales to a large number of nodes (100-200) even at smaller file sizes, assuming that 10-20% of the RR-DNS accesses are from new clients.

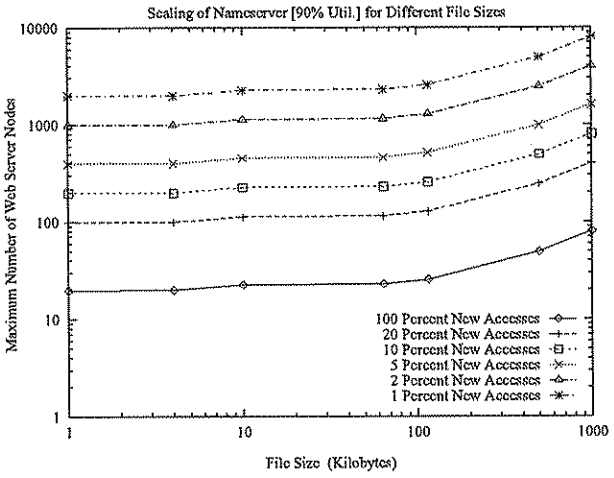


Figure 7: Scalability of RR-DNS

For the router, we measured the CPU utilization during the execution of the Webstone benchmark with four server nodes configured. For a given server node at 90-95% utilization, the CPU utilization of the router node is about 2.2% when the file-size is small (1 KByte) and 1.2% for large files (1 MByte). Thus, based on the average file size of the web server workload, a single router node (operating at 90% utilization) can support between 40 and 75 nodes, more than sufficient for practical systems. Figure 8 shows

the scaling of a single router node for documents with different file sizes. If larger configurations are desired, multiple router nodes can be used, with load balancing across the routers being achieved via RR-DNS. As shown in section 3.1, the router solution has better load balancing properties compared to the RR-DNS solution.

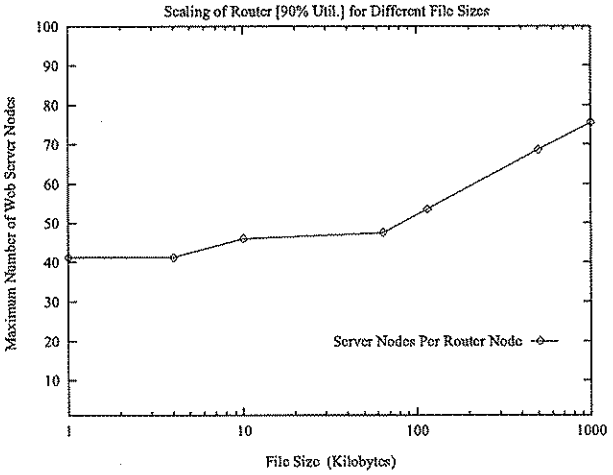


Figure 8: Scalability of a Single Router Node

4 High Availability

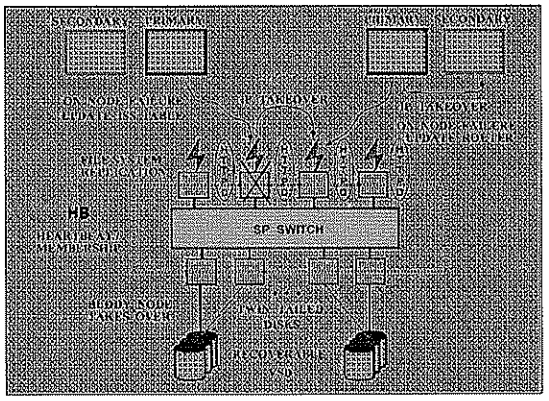


Figure 9: High Availability

As the traffic and commercial applications on popular web servers increase, providing highly available web servers becomes increasingly important. One of the advantages of a clustered system is the possibility of providing high availability by masking node and other failures. This is done by detecting failures and reconfiguring the system such that the workload is taken over by the remaining nodes in the cluster. For the scalable web server, in order to guarantee high availability, we need to mask both front-end and back-end node failure. The method we use for making the web server highly available is illustrated in Figure 9. Node failures are detected by using a heart-beat and membership protocol [7], that essentially eliminates nodes from the membership group in a consistent manner when (several) heartbeat messages from the node are not received. Network failures are detected by using heartbeats on multiple networks. Other resource failures are detected by monitors. On detection of node or other resource failures, a distributed recovery driver component [11] initiates recovery by coordinating a series of recovery commands or scripts across the remaining nodes in the system. The actions taken for specific failures is outlined below.

For front-end node failure, if the DNS or router solution is used for load balancing, the failed node is removed from either the name server tables or the router configuration table. Once these tables are updated to reflect the failure, all remote clients that try to make a connection after the failure will not be directed to the failed node. However, in the DNS approach some clients may have cached the address of the failed node and will continue to try connecting to it. Similarly, with the router based approach a router node may fail causing clients to see a failure. To handle these scenarios each front-end node and router node has a "buddy" which takes over its IP address after failure. Remote clients that have cached the address of the failed node will connect to the buddy transparently. For IP takeover the nodes have a redundant adapter. One of the adapters is used for servicing all the external IP requests and one for booting. The IP address of the main adapter is taken over by a spare adapter in the buddy node. The switching is done in software and is orchestrated by the recovery driver, as outlined earlier.

To handle the back-end server failure we use a hardware approach, namely twin-tailed disks. Other approaches like replication and software-RAID can be used for large scale servers with real-time requirements [17]. The disks attached to a node are twin-tailed to a buddy node. When a node fails the buddy node takes over the disk. The front-end nodes forward the requests to the buddy. Twin-tailing requires either special disks or extra SCSI adapters on the nodes that connect to the shareable disks. With twin-tailing the load on the buddy can double in the worst case. Multi-tailing can reduce the overload to some extent.

5 Conclusions and Future Work

Based on the analysis and design described above we have built a prototype scalable and highly available web server on an IBM SP2 system, illustrated in Figure 10. This system is essentially an instantiation of the architecture shown in Figure 2, and is a two-tier system with four front-end and four back-end nodes. The disks are connected to the back-end nodes. The disks are (logically) shared among the front-end nodes using the Virtual Shared Disk software [1]. The data for web files are striped across the back-end nodes and disks, thereby achieving load balancing across the back-end nodes based on work by [4]. The combination of the TCP router and RR-DNS described earlier is used for load balancing across HTTP daemons run-

ning at the front-end nodes. The TCP router is based on the encapsulated cluster prototype [14]. The scalability data presented in Section 3 is based on measurements on the prototype. The methods for achieving high availability outlined in the previous section have been implemented in the prototype.

The hardware and software infrastructure is shared between the scalable web server and a prototype scalable video server: the video objects are striped across the same nodes and disks, the HTTP daemons are replaced by video push daemons, and load balancing is achieved by a control server [9] that assigns the client streams to front-end nodes. We present forms with video selections on the web, and allow the web client to start up (out of band) play-out of video objects over separate high bandwidth channels. Currently, the constrained bandwidth to the client and also in the network prevent any significant real-time video access on the web. Over time, we believe that real-time video over the web will become feasible; the prototype's ability to support scalability for both web and video will then become increasingly important.

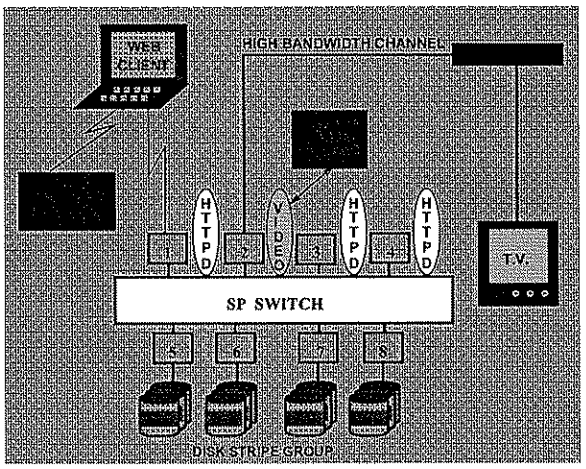


Figure 10: SP Video and Web Server Prototype

We quantified the scalability of this solution, and examined the performance of the (known) RR-DNS approach. We showed through simulations that the pure DNS approach can be significantly impacted by dynamic load imbalance among the nodes in the cluster. As a specific example, due to client caching of the mapping from the name to IP address, the peak load on nodes in the cluster can be 30% to 40% higher than the mean load on the nodes, because bursts of requests from clients lead to dynamic load imbalance. With higher values of the TTL at nameservers, the load dynamic load imbalance with RR-DNS gets even worse. This means that, using RR-DNS, the cluster nodes would need to be configured such that the mean load is less than 60% in order to be able to sustain the peak loads that occur. We showed that the router solution largely eliminates this problem, and provides excellent dynamic load balancing; this allows the cluster nodes to operate at much higher mean loads, and still sustain the peak loads.

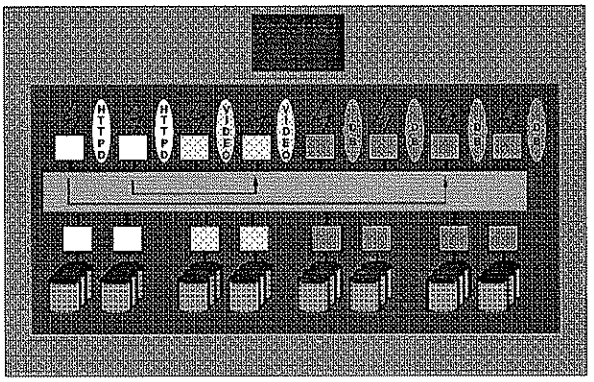


Figure 11: Integrated Scalable Solutions

In this paper we have focused on a scalable and highly available web server, with web pages stored on a shared file system. For many applications, access to back-end servers such as databases and on-line transaction processing via the web is required. The scalable web server described above can be integrated with parallel/scalable back-end servers, to create integrated scalable solutions, as illustrated in Figure 11. The figure illustrates the scalable web-server integrated in our prototype with a scalable video server and a parallel database, all in one management and high-availability domain.

Acknowledgments: The prototype draws on components developed by many people, especially C. R. Attanasio, S.C. Smith, C. Polyzois, R. Haskin, D. McNabb, F. Schmuck, J. Wyllie, M. Devarakonda and S. Fakhouri.

References

[1] C.R. Attanasio, M. Butrico, C.A. Polyzois, S.E. Smith, and J.L. Peterson. Design and Implementation of a Recoverable Virtual Shared Disk. IBM Research Report RC 19843, T.J Watson Research Center, Yorktown Heights, New York, 1994.

[2] T. Brisco. DNS Support for Load Balancing. RFC 1794, Rutgers University, April 1995.

[3] Berners-Lee T. et al. The World-Wide Web. *Communications of the ACM*, 37(8):76 – 82, August 1994.

[4] Roger Haskin. Personal Communication, 1994.

[5] Roger Haskin and Frank L. Stein. A System for Delivery of Interactive Television Programming. In *COMPCON*. IEEE, March 1995.

[6] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West. Scale and Performance in a

Distributed File System. *ACM Transactions on Computer Systems*, 6(1), February 1988.

[7] F. Jahanian, S. Fakhouri, and R. Rajkumar. Processor Group Membership Protocols: Specification, Design and Implementation. In *Proceedings of the 12th Symposium on Reliable Distributed Systems*, pages 2 – 11, Princeton, NJ, October 1993. IEEE Computer Society.

[8] Eric Dean Katz, Michelle Butler, and Robert McGrath. A Scalable HTTP Server: The NCSA Prototype. *Computer Networks and ISDN Sytems*, 27:155 – 163, 1994.

[9] Martin Kienzle. Personal Communication, 1995.

[10] Thomas T. Kwan, Robert E. McGrath, and Daniel A. Reed. Ncsa's world wide web server: Design and performance. *IEEE Computer*, pages 68 – 74, November 1995.

[11] Avraham Leff, Richard P. King, and Daniel M. Dias. HAV: Infrastructure for Building Highly Available Clustered Systems. In Preparation, 1996.

[12] Robert E. McGrath. What We Do and Don't Know
About the Load on the NCSA WWW Server. URL < http://www.ncsa.uiuc.edu/ Information-Servers/Colloquia/28.Sep.94/ Begin.html>, September 1994.

[13] P. Mockapetris. Domain Names - Implementation and Specification. RFC 1035, USC Information Sciences Institute, November 1987.

[14] Attanasio C. R. and Smith S.E. (a virtual multiprocessor implemented by an encapsulated cluster of loosely coupled computers). *IBM Research Report RC18442*, 1992.

[15] Herb Schwetman. CSIM Reference Manual (Revision 16). MCC Technical Report ACT-ST-252-87, Microelectronics and Computer Technology Corporation, Austin, Texas 78759, May 1992.

[16] Renu Tewari, Daniel M. Dias, Rajat Mukherjee, and Harrick Vin. Real-Time Issues for Clustered Multimedia Servers. IBM Research Report RC 20020, T.J. Watson Research Center, April 1995.

[17] Renu Tewari, Rajat Mukherjee, Daniel Dias, and Harrick Vin. High Availablity in Clustered Multimedia Servers. In *International Conference on Data Engineering*, New Orleans, February 1996. IEEE.

[18] Gene Trent and Mark Sake. WebSTONE: The First Generation in HTTP Server Benchmarking. URL < http://www.sgi.com/Products/ WebFORCE/WebStone/paper.html >, February 1995.

digest of papers

COMPCON '96

Technologies for the Information Superhighway

Forty-First IEEE Computer Society International Conference

Sponsored by — The IEEE Computer Society

February 25–28, 1996
Santa Clara, California



IEEE Computer Society Press
Los Alamitos, California

Washington • Brussels • Tokyo

IEEE Computer Society Press
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264

Copyright © 1996 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved.

Copyright and Reprint Permissions: Abstracting is permitted with credit to the source. Libraries may photocopy beyond the limits of US copyright law, for private use of patrons, those articles in this volume that carry a code at the bottom of the first page, provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.

Other copying, reprint, or republication requests should be addressed to: IEEE Copyrights Manager, IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331.

The papers in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and, in the interests of timely dissemination, are published as presented and without change. Their inclusion in this publication does not necessarily constitute endorsement by the editors, the IEEE Computer Society Press, or the Institute of Electrical and Electronics Engineers, Inc.

IEEE Computer Society Press Order Number PR07414
ISBN 0-8186-7414-8
ISSN 1063-6390

IEEE Order Plan Catalog Number 96CB35911
Order Plan ISBN 0-8186-7415-6
Microfiche ISBN 0-8186-7416-4

Additional copies may be ordered from:

IEEE Computer Society Press
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264
Tel: +1-714-821-8380
Fax: +1-714-821-4641
Email: cs.books@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: +1-908-981-1393
Fax: +1-908-981-9667
mis.custserv@computer.org

IEEE Computer Society
13, Avenue de l'Aquilon
B-1200 Brussels
BELGIUM
Tel: +32-2-770-2198
Fax: +32-2-770-8505
euro.ofc@computer.org

IEEE Computer Society
Ooshima Building
2-19-1 Minami-Aoyama
Minato-ku, Tokyo 107
JAPAN
Tel: +81-3-3408-3118
Fax: +81-3-3408-3553
tokyo.ofc@computer.org

Editorial production by Mary E. Kavanaugh
Cover by Joseph Daigle
Printed in the United States of America by KNI

The Institute of Electrical and Electronics Engineers, Inc.

Exhibit 3

Architecture and Design Of High Volume Web Sites

(A Brief History of IBM Sport and Event Web Sites)

Paul Dantzig
IBM T.J. Watson Research Center
19 Skyline Dr.
Hawthorne, NY 10532
001 (914) 723-5686
pauldantzig@us.ibm.com

ABSTRACT

Architecting and designing high volume Web sites has changed immensely over the last six years. These changes include the availability of inexpensive Pentium based servers, Linux, Java applications, commodity switches, connection management and caching engines, bandwidth price reductions, content distribution services, and many others. This paper describes the evolution of the best practices within IBM in architecting sites that handle millions of page views per day. Discussed is the transition to multi-tiered architectures, the use of publish/subscribe software to reduce Web site hits, the migration from static to dynamic content, and techniques for caching of dynamic and personalized content.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software, *Distributed Systems, Performance evaluation (efficiency and effectiveness)*. H.3.5 [**Online Information Services**]: Online Information Services, *Web-Based Services*.

General Terms

Performance, Design, Reliability.

Keywords

Web Serving, Content Management, Content Distribution, Proxy Caching, Java Messaging Services, Publish and Subscribe

1. INTRODUCTION

Architecting and designing high volume Web sites has changed immensely over the last six years. Popular Web sites today can receive a million or more hits per minute. If a significant percentage of those requests require dynamically generated data, those requests can consume orders of magnitude more resources than requests for static data. The initial design of high volume sites consisted of clustering a set of Web servers together and either using round-robin Domain Name Servers [6,8] (RR-DNS) or routing at the TCP level. All the Web servers were fully replicated mirrors of each other (containing the same database, application software, Web serving software, etc.). Users of such systems often experienced inconsistencies in a site's content due to lags or failures in replication as different servers served each page. The cost of replicating databases on the serving nodes consumed cycles needed for serving, resulting in inconsistent response times.

To reduce the consistency and performance issues sites began implementing two tiered architectures. The first tier consisted of a set of systems containing just Web servers and second tier consisted of backend servers such as databases, payment systems, and other legacy systems. This type of architecture provided significant relief from the above problems. Reduced numbers of database replications improved consistency and consumed significantly fewer resources. Response time for static requests was not impacted by the dynamic requests, which were now run in the backend systems. Some high volume sites grew to contain hundreds of front-end nodes. Scaling the backend usually consisted of installing bigger and bigger machines to avoid the database replication problems. It was quickly realized the primary difficulty of running hundreds of nodes was not the hardware cost but the personnel cost to manage them.

Many companies entered the Web server accelerator market (Cisco, Cacheflow, Network Appliance, IBM, etc.) to market new devices to reduce the number of servers. These devices typically handle an order of magnitude or more traffic than a single Web server running on a general-purpose operating system. These devices run as reverse proxy servers with large amounts of memory and disk. In addition, specialized switches and routers came on the market to distribute connections to the Web servers or the Web server accelerators. Newer devices on the market function both as proxies and connection routers, which use heuristics such as number of connections, server load, content, to distribute connections to multiple servers.

In addition to specialized hardware, new software techniques were developed to improve the performance of dynamic pages. One technique that was quite radical at the time was to cache dynamic pages the first time they were created. Methods for constructing pages from individual fragments allowed expensive content to be generated once and shared among many pages. However, successful deployment of these techniques requires much stricter cache consistency. In order to maintain the consistency of the cached data, the server needed to explicitly manage the cache

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
The Fourteenth International Conference on Software Engineering and Knowledge Engineering ' 02, July 15-19, 2002, Ischia, Italy.
Copyright 2002 ACM 1-58113-556-4/02/0700...$5.00.

contents. Algorithms for keeping cached dynamic data consistent are described in the referenced papers. [3,4]

Dynamic data was realized to be nothing more than rapidly changing static data. To keep certain types of content up to date required reducing the expiration times to tens of seconds. Some Web sites adopted automatic refresh of pages creating enormous numbers of hits. The new problem was how to reduce the number of hits without sacrificing the better response time users had become accustomed to.

For some types of content, in particular sports scoring results and financial market data, even better response times were considered necessary. Sport and Financial consoles have been created which invoke Java Applets, which open a long-term connection to a "publish and subscribe" system using either private protocols or published protocols such as Java Message Service (JMS) [10]. As content changes the publish and subscribe system pushes changes to those users interested in the change. These systems have become enormously popular with hundred of thousands of active consoles being supported.

Many high volume Web sites require 100% availability. Geographically redundant sites and content distribution services (such as Akamai) place servers and proxy servers closer to the user's ISP connection. In addition to improving availability, using multiple geographic sites or content distribution services can reduce the latency for delivery of graphic and streaming content.

The following sections of this paper describe how the IBM Sport and Event Web sites have been impacted by the above technologies.

2. 1996 Atlanta

Like many first of a kind Web projects during the early years the 1996 Atlanta Games Web Site [11] was architected, designed, and programmed in a hurry. The project started in September of 1995 and was never "completed". Some development continued even as the games were running. Much of the architecture and software came from a Web based digital library joint project between IBM and The Florida Center for Library Automation in Gainesville, Florida. [7] Figure 1 shows the basic architecture for the part of the Atlanta site for handling results on the Web.

The basic system setup was a cluster of twenty IBM RS6000s, rack mounted in two SP2 Frames, and two experimental TCP/IP router boxes for routing the incoming HTTP traffic. Each node contained identical software including the Apache HTTP server, a Fast CGI Apache extension, a connection manager daemon, a cache manager daemon, a database, and twenty pre-started processes preconnected to the database that performed SQL queries in the database if the memory cache did not contain the pre-built page. Each sport and the medals information were allocated a separate cache memory. If a node were restarted the caches would come up completely empty.

The first request for a page would invoke the following actions:

1. The HTTP requests were distributed to a server node based on system load from the TCP/IP Router to the Apache process.
2. Apache called the Fast CGI Apache Extension.
3. The Fast CGI Apache Extension queried the Cache Manager with the unique cache ID (part of the URL).
4. If the page was cached in the appropriate Sport or Medal Cache the cache manager returned the data, which was then returned to the user.
5. If the page was not cached; the connection manger was called to negotiate for a pre-started database process if available.
6. The pre-started process negotiated for and retrieved information from database and generated an HTML result page. The Cache Manager was invoked to store information in the appropriate cache. Finally the page was returned to the user.

In order to maintain cache consistency, whenever an update was made to one of the database tables related to a Sport or Medal a database trigger invoked a process to delete all entries in the specific cache for the affected sport.

As indicated in the introduction, node-to-node consistency and performance problems were both quite visible to the user community. The characteristics of sporting events made the performance problem even more obvious. Users would note when an event would end and inundate the system with requests. This was precisely when the result arrived in the database, causing the entire sport cache to be invalidated. All the requests would have to be resolved from the database until that data finally arrived in the cache. There was more than an order of magnitude difference between a cache hit and a database hit in response times.

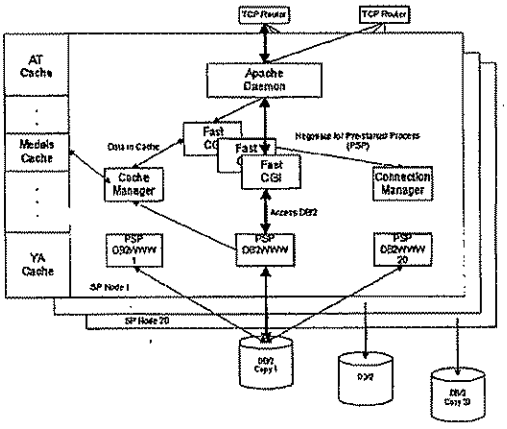


Figure 1. 1996 Atlanta Web Site Architecture

Over the 17 days the 1996 Atlanta site received over 190 million hits. 114 million of those hits were result requests. Peak result server hits was 9 million on August 1, 1996. Cache hit ratio averaged 80% and peak of 45 cache transactions per second per node was achieved.

3. 1998 Nagano

By 1998 IBM had signed up to sponsor several major sporting and other events. These new events included the US Open Tennis Tournament, Wimbledon, the French Open Tennis Tournament, the Grammy Awards, the Tony Awards, etc. It was decided to invest in building a permanent infrastructure that could last several years to handle the Sport and Event Web Sites. This new infrastructure was to be architected, designed, built, and tested before the Nagano games [12].

To achieve a high level of redundancy three permanent geographic locations were chosen close to major Internet backbone sites. One East Coast and two Midwestern United States sites were constructed. In addition a fourth temporary site for Asian traffic was established in Tokyo. Each site consisted of either three or four racks of eleven RS6000 nodes (one SP2 Frame). Figure 2 shows the configuration of one rack of nodes.

One of the criticisms of the Atlanta Web site was that finding information required too many page fetches. The Atlanta site was organized by results, news, photos, etc., and by sports and events. Although country information and athlete biographies were provided, results corresponding to a particular country or athlete could not be collated [2]. Consequently, the Nagano Web site organized results, news, photos, etc. by countries and athletes as well as by sports and events. Figure 3 shows the level of detail contained in each athlete and country page. Each page contained recent results, news, photos, and quotes. The Web results displayed all of the information related to an event (the Atlanta site in many cases only had summaries). Some of the pages now

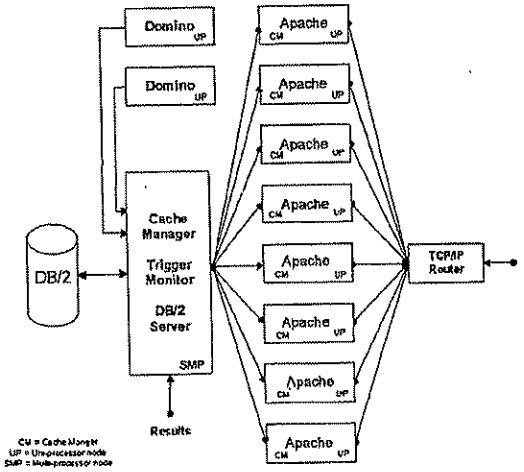


Figure 2. 1998 Nagano Web Site Architecture

were complex enough to require up to thirty seconds to produce the HTML tables (Ice Hockey results are a notable example). To reduce computational load Country and Athlete pages were only generated every fifteen minutes.

Shown in Figure 2 is the Nagano architecture. This architecture is a combination of two architectures: one to pre-generate content and perform content distribution, and the other to serve Web requests. Each time a change was made to news, photos, and related content, the Domino server triggered a change. Each time a schedule, bio, or result changed the database triggered a change.

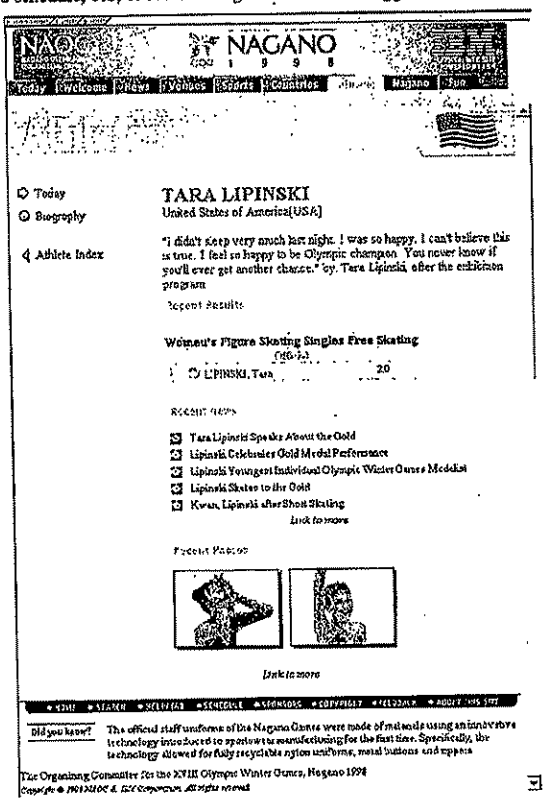


Figure 3. 1998 Nagano Athlete Page

The pieces of information flowing into the system (each news item, result, etc.), when converted to a Web object (HTML, image, etc.) were now considered a fragment of information and were usable anywhere in the site. A new technology (Object Dependency Graph (ODG)) was developed to keep track of the relationship between fragments and pages. [3,4]

The Cache Manager on each node took on the role of a distributed file system with no persistence. In theory 100% cache hit-ratio could have been achieved with this type of system. Because it was a memory-based cache, pages infrequently requested, such as athlete biographies and country information pages were frequently invalidated. Except for the country and athlete pages, the new model became pro-active page replacement in which a page is replaced in cache and never invalidated. Using fragments and the Object Dependency Graph, only the pages affected by the trigger were updated. By separating page rendering from page serving each server node now had consistent response time. Note: When an SMP node goes down the cache information is lost for all the

nodes; to avoid performance problems key pages in the site were primed before the nodes were put back online.

Based on the lessons learned from Atlanta, database replications were minimized. Each site contained no more than four databases (one for each rack). Users were directed to one of four sites by using standard router algorithms (e.g. Japanese traffic to Tokyo, British traffic to the East Coast). If one site failed or needed to be down for maintenance the routers automatically shifted load to the other sites.[2] The only data inconsistencies were between sites so users were unaware of the fact that the Asian site and East Coast site were not consistent with each other.

Nagano was an immense success. All of the goals for the site were achieved: 100% availability, good and consistent response times, every country and athlete treated equally, and no single points of failure. At the end of the event the publishing system contained approximately 50,000 pages in English and Japanese. The site had built 10,645 news article fragments, 14,355 photo fragments, 5,988 result fragments and total combination of the previous into 23,939 Sport, Country, and Athlete pages. The average number of pages rebuilt each day was 20,000 and on the peak day 58,000 pages were rebuilt.

The traffic to the site broke two Guinness records [9]. ***The Most Popular Internet Event Ever Recorded*** based on the officially audited figure of 634.7 million requests over the 16 days of the Games and ***The Most Hits On An Internet Site In One Minute*** based on the officially audited figure of 110,414 hits received in a single minute around the time of the Women's Figure Skating Free Skating (see Figure 3). Hits on the peak day were 55.4 million.

4. 1999 Wimbledon

Even before Nagano, many of the large Web sites were already looking for cheaper and faster ways of serving static content. By 1997 many of them had switched from using expensive proprietary Unix systems (SUN, IBM, HP) to Intel based systems running operating systems like Free-BSD and Linux with Apache to reduce costs and improve performance. In 1997 IBM Research prototyped a combination router and proxy server cache that could support an order of magnitude more hits that a server on the Sport and Event Web sites could handle. The product was transferred to development and made available to the Sport and Event environment after Nagano was complete.

After a number of trial runs during events in 1998 four production Web server accelerators (IBM 2216s) were put into production for the 1999 Wimbledon event [13]. Figure 4 is a graph of the Web hits to the Wimbledon site on July 2, 1999. Along the bottom edge of the graph is time the 10:00 to 16:00 GMT. At about 10:45 the Bethesda Web server accelerator #1 achieved a rate of 66K hits per minute. At 13:32 the four Web server accelerators combined achieved a rate of 140K hits per minute. The Schaumburg, Columbus, and Bethesda bars on the graph are the combined total of 53 Web server nodes. The peek point 13:20 on the graph is 375K hits per minute (i.e. one web server accelerator is equivalent to about fifteen serving nodes).

To get high cache hit ratios (93%) result-scoring content was pushed into the Web server accelerators thru a proprietary interface whenever content changed using the Trigger Monitor daemon. The proxy server built-in to each Web Server Accelerator requested other site content. Each Web server accelerator contained 500M of memory and the working set for Wimbledon was less than 200M of memory.

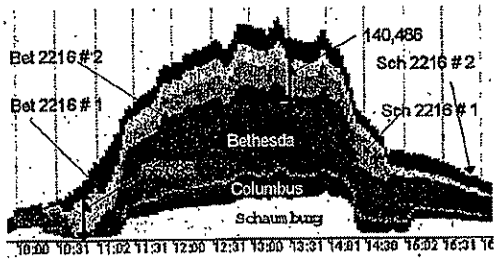


Figure 4. 1999 Wimbledon July 2, 1999 Hits Per Minute

The 1999 Wimbledon event ended with the officially audited figure of 942 million requests over the 14-day event. The peak-hit rate of 430K hits per minute was received on 6/30/99. 71.2 million page views and 8.7 million visits to the site were received. This event proved that inexpensive Web Server Accelerator hardware could reduce the size and cost of the infrastructure even with the enormous rate of growth of these events.

Many Web server accelerators (including the IBM 2216) produce no Web server logs either because the logging function does not exist, or because logging impacts the performance of the accelerator. However, most do support SNMP interfaces, which can be used to record hits, misses, memory usage, and so on. The information for Figure 4 was obtained using the SNMP interface. Without standard Web server logs information such as page views, unique visitors, visits, etc. would be lost. To retain this type of information every page in our event Web sites was modified to add a new request called UC.GIF (uncachable one pel gif). This request is forced to go all the way through to the server. Added to the request are parameters that identify the user, which page was requested, implicit hits, etc. The Web server log processing programs add this information to the standard statistics collected for the site.

5. 2000 Sydney

Based on what we had learned from the previous events the key to the success of Sydney [14] would be making Web Server Accelerator cache misses low cost. Two critical pieces were already prototyped. Learned from Nagano was how to make a system for pre-building pages (Trigger Monitor) and distributing pages. Learned from Wimbledon was how to make a really good edge-serving infrastructure using Web Server Accelerators. Both of these systems needed critical improvements to handle the bigger loads anticipated in Sydney.

Installed after Wimbledon were sixty-five Web server accelerators. A fourth temporary server site in Sydney was planned to augment the three permanent server sites which now had one East Coast, one Midwest, and one West Coast United States locations. An agreement was made with a telecom to collocate a large number of the Web server accelerators at five major Internet hubs in the USA. The installed capacity of the accelerators was between 3 and 4 million hits per minute. Figure 5 shows the server and Web server accelerator locations. The East

Coast would handle the European traffic and the West Coast the Asian traffic. The origin servers were located at IBM Global Services Hosting locations nearby. Experiments had been run during other events, which demonstrated that if **ALL** of the content (HTML, GIFs, JPEGs, etc.) were given **correct expirations**, content would not have to be pushed to the caches; the accelerators would simply proxy the content on a miss. The cache-hit ratio typically dropped to about 75%, which was within the hit capacity of the server sites (Figure 6).

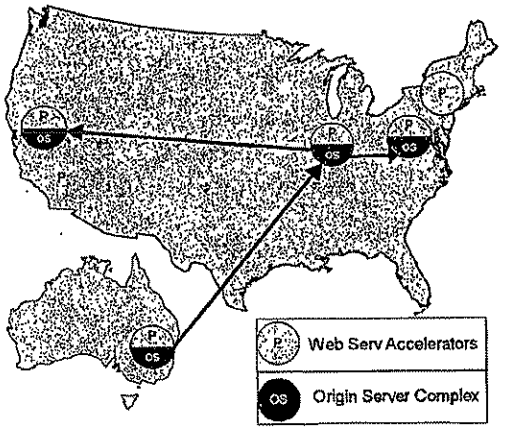


Figure 5. Sydney Origin Servers and Web Server Accelerators Locations

Because of the experimental nature of the dynamic page generation system, both the Atlanta and Nagano sites were designed and implemented as two separate sites, one of which served relatively static content, and the highly dynamic results delivery system. Shortly before the games started an ad-hoc effort was made to merge the sites and present the illusion of a single unified site. It became quite clear during the merge phase of the Nagano site that the Sydney site would need to be constructed from the beginning as a single, unified site.

The combination of building a complete, unified site based on fragment technology, as well as improvements in the technology itself produced an unforeseen problem: how to manage a very large number of fragments. Early estimates for the size of Sydney content were based on simply taking the number of sports, events, countries, athletes, etc. and multiplying Nagano times three (75K pages and 150K rebuilt pages a day). No more than four fragments per page and no imbedded fragments were supported in the Nagano publishing system.

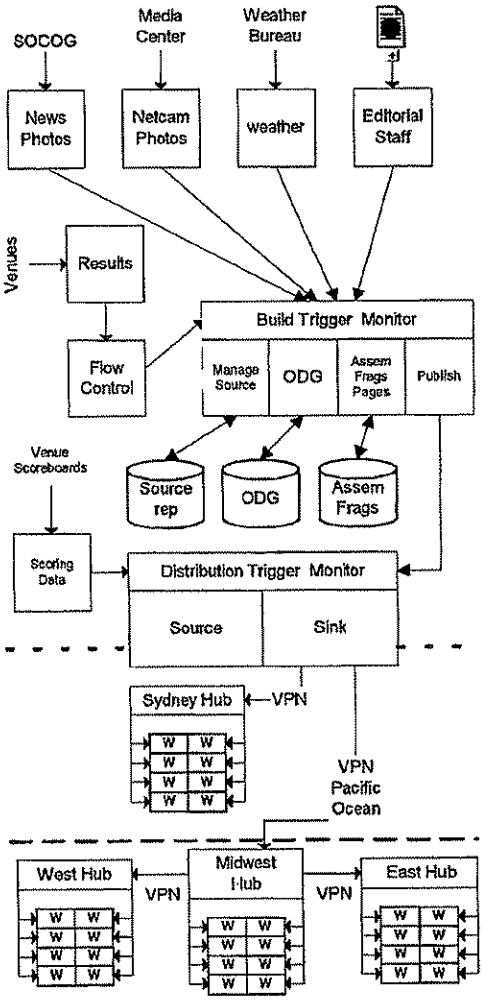


Figure 6. Sydney Content Management and Distribution

Some of the design concept pages for Sydney had thirty fragments on a page with a multiple levels of imbedded fragments. Some updates to the content during the games were expected to affect hundreds of pages (For example: Baseball and Basketball player statistics). To reduce the time required to build these large and complex page updates (Figure 5) source fragments, the ODG, and assembled fragments were saved in a persistent repository called Hash Table on Disk (HTOD). HTOD is a lightweight repository for objects that is both faster and easier to manage than files or a database. [5]

The Trigger Monitor daemon from Nagano was completely rewritten in Java and put into production nine months before the games to insure that all content for the site could be managed by a single content management system.

The content distribution system was also redesigned. In Atlanta the results were delivered to each node for database update processing by using a distributed file system. Static content was also served from the distributed file system. In Nagano, the caching system performed the task of the distributed file system for the fast changing results while Web objects such as news, photos, and quotes, which changed at much lower rates of change, were kept in a distributed file system. Distributed file systems were used effectively for Sport and Event sites with only a few thousand pages from 1996 till 2000. The requirement to deliver tens and thousands of pages in less than thirty seconds or better to large number of servers could not be met by distributed file system, however.

Figure 6 shows the architecture of the Sydney publishing system, showing the interaction of the content management system and the content distribution system. (Much of the complexity of the system has been removed from this diagram.) The complete content management system (everything above the dotted line) was duplicated to avoid any single point of failure. The HTOD was used everywhere. Not shown in the diagram are the HTOD repositories used to maintain the queues of Web objects for the distribution Trigger Monitor, Sydney, Midwest, East Coast, and West Coast hubs. The final distribution point for the Web objects was the local file system of the Web serving nodes (Ws). The Web server infrastructure in the United States is shown below the dashed line. Notice the shift between Nagano and Sydney from many expensive, replicated engines at every site to a single, large, centralized multiprocessor engine for content management and many inexpensive Web server accelerators to serve the site. Note also that in Sydney there was only a single instance of any database, finally eliminating all replications.

Figure 7 shows the Sydney 2000 Web site Australian Country page. It illustrates the depth of the information available to the web site users. If all one was interested in was how athletes from Australia were doing you could bookmark this one page. All results were available for every sport and event that Australians participated even if they finished in fourteenth place. The same result fragments was used on the athlete's pages. Available thru the Athlete pages were all the preliminary heats, games, statistics, etc. that athlete has participated in. All of this information typically was available in less than one minute after the result system triggered a new result.

At the end of the event, the fragment repository contained 277K fragments occupying almost 2G of disk space. The site finished the event with over 107K published Web objects (78K pages and 29K images). On the peak day, the content management and content distribution system processed 291K Web objects (nearly five times Nagano). 600K total trigger transactions were processed during the sixteen days creating 3.6M new and changed Web objects, which occupied 125G of data.

More than seven terabytes of data was distributed by the content distribution system to the fifty-seven nodes. Scoreboard updates were delivered from Sydney to the East Coast serving nodes in less than 6 seconds. Given automatic delays built into most television broadcast of 30 seconds, users discovered that the site had scores on the Web before the broadcasters. Even the complex

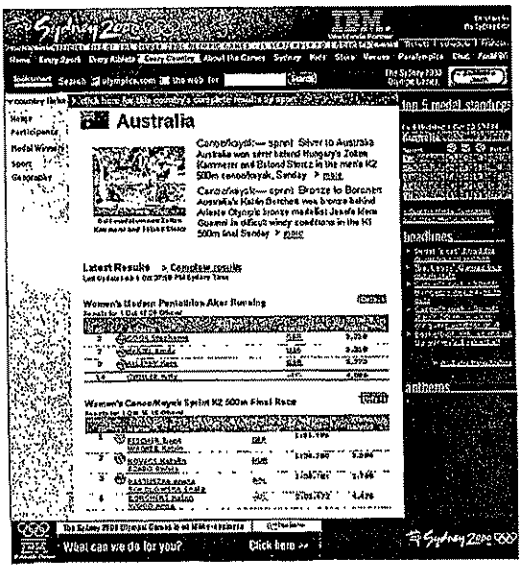


Figure 7. Sydney 2000 Australia Country Page

result updates with hundreds of pages changed were delivered on average in less than 40 seconds to the serving nodes.

It should be no surprise that every event record was broken again. The traffic to the site broke the most hits on a sport event site in a single minute based on officially audited figure of 1.2M hits. Total traffic for the event over the sixteen days was 11.3 billion hits. The Web server accelerators handled less than half (5.5 billion) of the hits; the ISP proxy caches and the user browsers handled the rest (due to the expiration policy). The Web server accelerators had 80-90% cache hit ratio during the entire event. Thirty percent of the log entries on the serving nodes were UC.GIF records. Other significant statistics include the peak traffic day of 875M hits, 2.8M visits, and 18M page views and a total for the event of 35M official visits and 222M page views. Sydney was completely successfully and fulfilled the promises of being bigger than any of the previous events.

6. 2001 Wimbledon

Sports fans are a difficult audience to satisfy. For Atlanta almost all the scores were delivered after the event was over, for Nagano many events provided periodic updates, for Sydney all the major sports had sport consoles with thirty second updates. With Wimbledon 2001 [15], technology finally caught up with the fans desire for real-time scoring consoles.

With the new content distribution system, Web objects could be reliably distributed to all the geographic server sites in seconds. This enabled very short expirations of thirty seconds to be set on data for the scoreboard applets. With scoring data delivered to the most distant serving node in about six seconds and the scoring data's thirty-second expiration, the scoring console on the user's workstation is nearly always up to date. New technologies based on publish *and subscribe* were added to the event infrastructure to

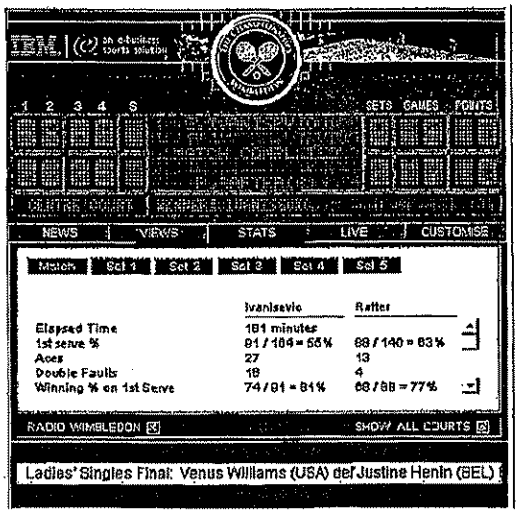


Figure 8. Wimbledon 2001 Tennis Console

support a real-time scoreboard. [1] Figure 8 shows a sample tennis console for Wimbledon 2001. The console is a Java Applet and uses Java Message Service (JMS) as the protocol to subscribe with the message broker. Each user acquires a permanent TCP/IP socket to a message broker. The trade off of the cost of hits to a Web server accelerator or a permanent connection to a message broker is based on update rate. Using current technology update rates of less than about thirty seconds messaging tends to be cheaper (although research into the exact "break-even" point is needed). For the 2001 Wimbledon finals between Rafter and Ivanisevic over 230K consoles were active. Two other sporting event records were broken during this event with peak hits per minute at 1.4M and peak page-views in a day at 24M.

7. 2002 Sport and Events

All the hardware and telecommunication equipment is in the process of being completely replaced. The servers are now IBM Pentium based processors running Linux. The Web server accelerators are being replaced with commercially available equipment. Multiple vendors provide telecommunication bandwidth directly (OC-12s) into each of three geographic sites. Figure 9 shows one aspect of the huge growth the sites have received. As the Web audience has increased worldwide the sites have continue to receive more hits, page-views, visits, etc. year after year.

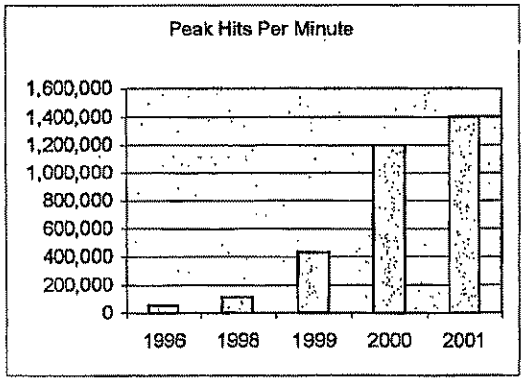


Figure 9. Sport and Events Peak Hits Per Minute

8. ACKNOWLEDGMENTS

My thanks to Jim Challenger, Arun Iyengar, Paul Reed, Cameron Ferstat, Karen Witting, Daniel Dias, and Jerry Spivak.

9. REFERENCES

[1] Banavar, G. Chandra, T. Mukherjee, B. Nagarjarao, J Strom, R.E. Sturman, D.C. An Efficient Multicast Protocol for Content-Based Publish Subscribe Systems, International Conference on Distributed Computing Systems, 1999

[2] Challenger, J. Dantzig, P.M. Iyengar, A. A Scalable and Highly Available System for Serving Dynamic Data at Frequently Accessed Web Sites, Proceedings of SC98, 1998.

[3] Challenger, J. Dantzig, P.M., Iyengar, A. A Scalable System for Consistently Caching Dynamic Web Data, Proceedings of IEEE INFOCOM'99, 1999

[4] Challenger, J. Ferstat, C. Iyengar, A., Reed, P. Witting, C. A Publishing System for Efficiently Creating Dynamic Web Content, Proceedings of IEEE INFOCOM'00, 2000

[5] Challenger, J. Iyengar, A. Jin, S. Techniques for Allocating Persistent Storage, IEEE Symposium on Large Scale Storage in the Web, April 2001.

[6] Colajanni, M. Dias, D. Yu, P. Scheduling Algorithms for Distributed Web Servers, Proceedings of International Conference on Distributed Computing Systems, 1997.

[7] Dantzig, P.D. Liu, Y. Ni, L M. Wu, C.E. A Distributed Connection Manger Interface For Web Services on IBM SP Systems, The Fourth IEEE International Conference on Parallel and Distributed Systems (ICPADS'96), 1996.

[8] Dias, D. Kish, W. Mukherjee, R. Tewari, R. A Scalable and Highly Available Web Server, Proceedings of IEEE Computer Conference, 1996.

[9] Guinness World Records, Guinness Book of Records, 1998.

[10] Java (TM Sun Microsystems) Message Service Documentation, http://java.sun.com/products/jms/docs.html

[11] 1996 Atlanta Olympic Games Web Site, July 19 – August 04, 1996, http://www.atlanta.olympic.org, site is not available.

[12] 1998 Nagano Olympic Games Web Site, February 7 – February 22, 1998, http://www.nagano.olympic.org, site is not available.

[13] 1999 The Championships Wimbledon, June 21 – July 4, 1999, http://www.wimbledon.org, site is not available.

[14] 2000 Sydney Olympic Games Web Site, September 15 – October 1, 2000, http://www.olympic.com, site is not available.

[15] 2001 The Championships Wimbledon, June 25 – July 8, 2001, http://www.wimbledon.org, site is not available.

Exhibit 4

Atlanta Olympics WOMplex



By Andy Stanford-Clark

BEST AVAILABLE COPY

This article describes the major components of the Atlanta Olympics WOMplex and attempts to give a flavor of some cutting-edge technology developed for the project.

For the summer of '96, a team of IBM programmers and engineers, based in Southbury, Connecticut, built the official Internet Web site for the Atlanta Olympic Games. The project was immense in every respect—from the amount of content in the site, to the volume of traffic through the site, to the amount of equipment deployed, and the number of hours worked by the team.

WOMplex is an umbrella term that describes the combination of the Web Object Manager (WOM) Internet content and application server, the scalable architecture, the requisite scalable parallel processing hardware, the content authoring environment, the load-balancing systems, the user-level applications on the Web site, and the subsystems that support those applications. In other words, a WOMplex is a scalable, manageable, globally distributable Internet content and application server comprising both hardware and software.

Statistics

The URLs http://www.atlanta.olympic.org and http://results.atlanta.olympic.org together constituted the largest Web server in the world. During the 17 days of the Olympics, the site received just over 192 million hits, with peak daily hits of 16.9 million. Drilling down to the "Sneak Peek" application, which provided a constantly changing montage of captured images from 38 video feeds from the Games in Atlanta, the servers handled up to 21,600 images per hour. What makes these figures especially impressive is that every page served was built dynamically from its component objects at the instant the page was requested.

To provide this level of service 24 hours a day required non-trivial amounts of equipment. The Atlanta site was served by 50 nodes of a distributed IBM RS/6000™ Scalable POWERparallel™ system, spread across five locations in four countries. Most of the processors were at the primary WOMplex site at one of Integrated Systems Solutions Corporation's (ISSC's) major Facilities Management Centers in Southbury. This site provided the required infrastructure for a project of this scale—Uninterruptable Power Supplies(UPS), good physical security, high-bandwidth connectivity into major U.S. Network Access Points (NAPs), and raised floor.

Several nodes—typically some portion of an RS/6000 SP frame—joined the WOMplex at four mirror sites:

- Cornell Theory Center at Ithaca, New York
- IBM U.K. Laboratories at Hursley, U.K.
- University of Karlsruhe near Mainz, Germany
- Keio University near Tokyo, Japan

Each mirror site replicated most of the content of the Olympic site. Notable exceptions were the results system and the rapidly changing Sneak Peek images, which were served only from the primary site at Southbury.



Presented at Get Connected Technical Interchange '96, IBM Hursley U.K., October 1996

BACK TO CONTENTS

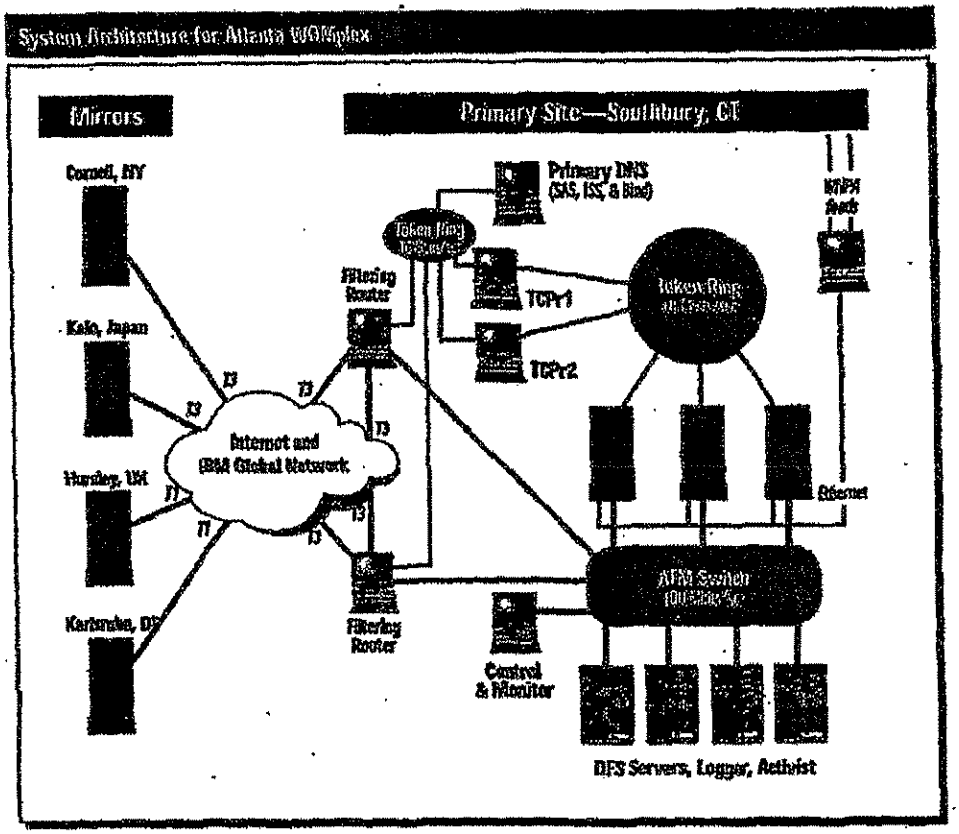


Figure 1. System architecture used for the Olympic games in Atlanta

During the last few months before the games, tickets for seats to events at the games were sold over the Internet by a server based on the IBM Net.Commerce server. In all, this generated over $5 million in ticket sales.

System Architecture

Figure 1 shows the overall system architecture for the Atlanta WOMplex. The "cloud" combines the Internet and the IBM Global Network. T3 connections are DS3 links providing 45 Mbits per second. Filtering routers are Cisco® 7500 Series routers with sets of carefully crafted filter rules to allow only the desired packets in and out from desired sources and destinations. TCPr1 and TCPr2 are the TCP routers, or WOMsprayers—these will be discussed later. Together SAS, ISS, and Bind form the name resolution service. The WNPA feed is a 56 Kbits per second leased-line connection to Atlanta, which provided the feed of results to the databases that were used to drive the Internet Results System. Several leased lines were installed for redundancy. The roles of the machines labeled Logger and Activist are discussed below.

Infrastructure

The IBM RS/6000 SP system provided the WOMplex infrastructure. At the main Southbury site, three frames of the RS/6000 SP were filled with mainly "thin" nodes (equivalent to the RS/6000 Model 390). We did not use the RS/6000 SP high-speed switch in this application—all communication between processors was done over the Asynchronous Transfer Mode (ATM) network.

Control workstations for the main SP frames were in the machine room near the RS/6000 SP. Access to the SP control workstations at the remote mirror sites was handled through X-Windows sessions running across the Internet, using

BACK TO CONTENTS

BEST AVAILABLE COPY

the machines were generated and then frequently changed. The apparent inconvenience of complex password schemes was easy to justify in the high profile environment of the Olympics.

Transmission of system control information, including machine root passwords, across the Internet was a potential major exposure. Early adoption of Secure SHell (SSH) for all terminal communication between machines solved this problem. AIX® and Windows® 95 clients for SSH allowed developers to use SSH as a direct replacement for the telnet they would have used otherwise. A version of the SSH client that worked through a SOCKS firewall allowed developers inside the IBM firewall to access the servers and development lab machines.

ISS provided another level of security with the "caching, routing name server," which was deployed as the primary name server for the Olympics site. In this Domain Name Server (DNS) emulation mode, ISS had authority over one or more specified subdomains. If ISS received a name resolution request for a subdomain over which it did not have authority or a name that was not being used for load balancing, then ISS passed the request to one of two backend, ordinary, name servers—typically knowledgeable DNSs that could resolve nearly any request.

The IP address of the original request determines the backend name server that receives the request. ISS configuration has a table of subnet mask specifications. The client IP address, or name server, requesting the name resolution is matched against this table. If a match occurs, the client is considered internal, and passed to one backend DNS; otherwise, it is considered external and sent to the other one.

From the practical standpoint, even with a distributed, multi-site operation, the appropriate client machines can access all of the main site's internal server names and addresses, whereas the rest of the world can only access the limited set of names and addresses (possibly only the Internet address) that the DNS administrator made available. The ISS name server also blocks the DNS Zone Transfer operation, a popular method to access lists of internal machines that are considered hacking targets.

The WOM system has been iteratively developed to support high-volume Web sites for sporting events.

The WOM Web Server

The Atlanta Web site used the latest version of IBM's Web Object Manager system, known as the *WOMserver*. The WOM system has been iteratively developed to support high-volume Web sites for sporting events and IBM external Web sites since 1994. Notable events have included Wimbledon® Tennis, Masters Golf, U.S. Open™ Tennis, and the ACM Chess Challenge (Kasparov versus Deep Blue).

Object-oriented Perl was the language used for the Atlanta server. Supporting applications and subsystems on the site were written in C, Perl, and Java™. Several prototype versions of the WOMserver written entirely in Java were also deployed in the WOMplex for experimental, demonstration, and testing purposes. A specialized binary server, based on the Apache Web server and run on a separate port number, handled binary objects—GIF, JPEG, AVI, MPEG, WAV, Java classes, and Bamba audio and video. All references to binary objects in the HTML source dynamically served as pointers to this port (for example, http://www.atlanta.olympic.org:8080/).

The data flow for entering authored content into the WOMserver had several steps. Content was authored by several types of systems, including traditional HTML editors, image editing tools, and a new generation of WOMtools for directly authoring into the WOM internal object format. This content was then "checked in" to the WOM database. Other systems, based on IBM Digital Libraries technology, generated Web pages from dynamically changing data, such as sports results.

WOMwire was the protocol used for communication among the various tools, including several batch upload tools. This protocol defines the way in which objects and their metadata are transmitted to the WOM system. The WOM database used the DB2/6000™ relational model to store the data and metadata object.

The WOMserver can access data and metadata stored in several formats. For the Atlanta WOMplex, the data and metadata were exported as flat files, called *servables*, from the DB2® database into the DFS filesystem. The DB2 export mechanism also triggered the DFS fileset replication system to push the updated files out to all servers at each WOMplex site.

Web-based applications for the Atlanta site would traditionally have been written as Common Gateway Interface (CGI) programs. However, they were written for the WOMserver in an object-oriented Perl or Java environment. They were run inside the server to avoid the overheads associated with executing CGI applications on traditional Web servers.

WOM architecture allows an executable program to generate any part of a served page;

the applications also fit within this paradigm. Detailed discussion of the WOM Network Application Platform is outside the scope of this article.

The WOMserver is *multi-homed*—it can support several distinct Web sites concurrently in the same server. This makes the WOMplex, with global load-balancing systems, a potential home for many Web sites at the same time—ideal for providing content-hosting services.

Centralized Logging

An important aspect of WOMplex operation is collecting HTTP access and error logs from the servers distributed across the world and bringing them into a central location for monitoring and analysis. The "Grim Reaper" performs this function. The Grim Reaper, a tree-structured plumbing of sockets and UNIX® pipes, streamed log records from each server to a confluence point at each site, then across the Internet to a central collection point on a machine called the *logger* (see Figure 1).

When everything is streaming nicely, the task of the Grim Reaper is straightforward. The problems occur when one of three events occurs:

- The servers produce log data faster than the local Grim Reaper can take it from them.
- Network delays or failures cause holdups in the Grim Reaper plumbing, causing a back-log of data.
- The final receiver of the logs cannot cope with the rate at which they are arriving from the Grim Reaper.

Any of these scenarios, coupled with the simple mandate that no log records can be discarded, make the Grim Reaper's job very difficult. As a result, the Reaper is a sophisticated technology. At any stage, log streams can be diverted into a file on a local disk to hold data, which for some reason, cannot be sent on. Later, when connections are restored or congestion eases, files are spooled back into the network. The Grim Reaper also can deal with requirements such as off-line transfer of batched log data (for example, late at night with less network traffic) and on-the-fly data compression for transmission across slower network connections.

After arriving at the logger machine, the log streams were transferred across a high-speed protocol Common Link Access to Workstation (CLAW) link to an 8-way CMOS 390 Squadron mainframe. The logs were then pre-processed by a series of pipes and filters, then implemented using MVS® Pipes. Individual log records were added to a table in a DB2/MVS database. From here, SQL queries could extract useful information from the logs, including hit counts and rates, online data mining using Activist, and real-time "click tracking." The raw log data was spooled to tape for later analysis.

Content Authoring Environment

The WOMwire protocol defines how authoring and site management tools communicate across the wire with the WOM database. The client part of the WOMwire protocol was implemented as an ActiveX control. The authoring tools, developed for the Windows 95 and Windows NT™ environments in Visual Basic®, C, or C++, could trivially access the WOM database by including the WOMwire control.

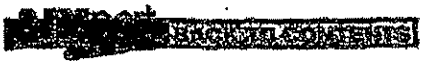
WOM objects can contain text, graphics, sound, video, page layout information, or executable code. These objects and their associated metadata can be checked in and out of the system, similar to a programmer's source code control system. The WOMbler created and edited content and managed metadata. WOMexplorer was used to navigate and reorganize a WOMplex site.

WOM objects can contain text, graphics, sound, video, page layout information, or executable code.

Each page on the site fits into an information taxonomy, described by a set of taxonomy tags. These tags position the page somewhere in an *information space*, which has a different hierarchy than the *data space* hierarchy imposed by the HTML links between a set of pages. This exciting new approach to Web site design is outside the scope of this article.

A Visual Basic application called the *Wit-tool classifier* was used to classify existing pages on the site. Users could classify each page into the information taxonomy by choosing from sets of predefined categories that applied to the page in question.

With the WOMpix tool, users could process images by cropping, resizing, reformatting, and labeling, before checking them into the WOM database. WOMpix dealt with many thousands of images that appeared on the site during the Olympics. It also allowed metadata, such as the source agency and photographer, to be recorded with the image. WOMpix could also generate thumb-nail images and image water-marking.



Load Distribution

Load distribution and balancing within the WOMplex uses a two-tier architecture. At each site, local load balancing distributes incoming traffic toward a single IP address across multiple backend WOMservers. Between sites, a global load-balancing system allows logical Web sites to be moved between physical server locations.

Interactive Session Support

Both tiers of the load-balancing system use software called Interactive Session Support (ISS). This is the interactive component of the RS/6000 SP LoadLeveler™ package for batch and interactive load balancing across a set of RS/6000 SP nodes or across other UNIX workstations, notably RS/6000, HP™, Sun®, and Silicon Graphics.

ISSmonitor collates agent responses, starts and stops remote agents (using inetd), and reacts appropriately if agents do not report in for an extended period of time.

Each server runs the ISSagent, which runs a user-defined metric that determines how busy the server is. The metric can be any executable program or pipeline of UNIX commands, providing it returns an integer at the end. For example, a popular choice is `ps -ef | wc -l`, which reports the number of processes currently running on the system and provides a good measure of machine activity. Far more complex metrics can be contrived, including an executable program to generate the metric value. The metric used for the Olympics counted the number of established socket connections on port 80 of each server.

ISSagents periodically report to a central ISSmonitor, which collates agent responses, starts and stops remote agents (using `inetd`), and reacts appropriately if agents do not report in for an extended period of time. The ISSmonitor also uses Internet Control Message Protocol (ICMP) ping packets to ensure that servers are still "alive" before recommending them for use. The ISSmonitor transfers a *load profile* of server activity to the WOMsprayer via a socket. The WOMsprayer uses the load profile to compile a weighted round-robin schedule for incoming traffic. Any server whose ISSagent does not respond is eliminated immediately from the load-balancing schedule to prevent incoming traffic from hitting a non-responsive server and receiving `Connection Refused` error message.

In another operational mode, ISS works as a load-balancing Domain Name Server (DNS). A generic Internet name, such as www.atlanta.olympic.org, is assigned to the cluster of servers. When a client machine or another name server in the hierarchical DNS system requests the IP address for that name, ISS ensures that the name server returns the IP address of the least heavily loaded server to provide the required service. ISS can work in DNS mode in one of two ways.

ISS can cooperate with an existing DNS, by regularly updating the "name to address mapping" database of the name server. Although this works well for low volume, long-lived connections, it is not responsive enough for high-volume, short-lived transactions usually experienced by a Web server.

ISS also operates as a high-performance, load-sensitive DNS, totally replacing the existing name server. This mode of operation can handle Web server traffic and provide a weighted round-robin mechanism for distributing high-volume load evenly across the backend servers.

ISS also offers an enhanced security, caching name server facility, described in the Security section above.

Local Load Balancing—WOMsprayer

Load-balancing software technology from IBM Watson Research has several names: TCP router, Encapsulated Cluster, WOMsprayer, ShockAbsorber, and Network Dispatcher. The *WOMsprayer*, a network router between two physical networks, offers a single IP address to the world. When a TCP connection is made to that address on the inbound network, the WOMsprayer software modifies the Medium Access Control (MAC) address field of the incoming connection request packet and transfers the packet to its outbound network. The MAC address change redefines the final packet destination, and the WOMsprayer arranges for it to be delivered to one of several backend servers.

The backend machines have unique IP addresses and also have the IP address of the WOMsprayer aliased onto their inbound network interface. Consequently, when this modified connection request packet arrives at a backend machine, it appears that the packet was destined for this server; the server does the required "accept" to create the TCP session. The WOMsprayer maintains a table of currently active connections, so any further incoming packets for that TCP session, such as packet acknowledgments, are routed to the correct backend server.

A weighted round-robin schedule determines which backend server receives an incoming session request. This schedule is constructed from the server activity profile that the ISSmonitor periodically supplies to the WOMsprayer. If ISS has not heard from a server or has reason to believe that a server is having problems, the

server is quickly excluded from the WOMsprayer schedule. No new connections are routed to that server until it rejoins the cluster.

The backend server's default route serves as the network path from the server to the client browser. This route generally does not use the same machine as the WOMsprayer. Even if the same server is used, the WOMsprayer does not intercept the returning packets.

Global Load Balancing

Two mechanisms allow Web traffic to be spread effectively across the multiple sites comprising the WOMplex: ISS-DNS and the WOMbat.

ISS-DNS

In its DNS emulation mode, ISS allows the DNS administrator to configure a table of Internet domain names to IP address mappings. This differs from an ordinary DNS in that more than one IP address may be associated with any given Internet name. An ISSagent runs on each server that is designated to serve any given name. According to the health, load, and the configured balancing policy for that name, incoming resolution requests for that name receive the IP address of the most appropriate server.

In the WOMplex, the ISSagents run on the machine that hosts the WOMsprayer for each site. Two balancing policies can be configured: Round and Fail.

Round. This policy does a weighted round-robin of the IP addresses in the list associated with that name. It is distributed according to two factors:

- The server activity profile, as determined from the input provided by the ISSagents
- A server weighting factor, allowing for different server capacity at each site—different numbers of WOMserver nodes in the RS/6000 SPs at each site, in the case of the WOMplex

Fail. This attempts to use the first IP address in the list—the "preferred" site—until that site becomes unavailable. Then, ISS switches to a weighted round-robin of the remaining IP addresses in the list, the same as in the Round policy. This system allows a single, logical Web site (defined by a specific Internet name) to be distributed across multiple physical locations, which could be a cluster of servers controlled by a WOMsprayer. Adjustments to the system configuration can be made dynamically, allowing extra servers to be switched-in, if necessary, to cope with excess load. The system enables automatic fail-over to other servers if a server experiences downtime, which can be fail-over to other servers from a preferred or primary server or omission of unresponsive servers from the round-robin schedule.

WOMbat

WOMbat uses ping to "echo-locate" a client machine and determine the nearest WOMplex site in network round-trip terms. With this knowledge, WOMserver can then divert the client to its nearest server site, and hopefully receive better service, thanks to the better network connectivity to that site.

When a client browser visits a WOMplex site for the first time, the server knows it has not seen that client before, because the client does not have a browser *Cookie* for that site. The server generates a Cookie, a unique key to identify that client in the future, and sends it to the browser to use whenever it visits that site. The Cookie is also stored in the Global Profile Daemon (GPD), a database of profile information, keyed by browser Cookie. The server also invokes— asynchronously in the background—the WOMbat Driver before composing the HTML page requested by the client and returning it to the client.

The WOMbat Driver sends details of the client's IP address and its browser Cookie to a WOMbat daemon at each site in the WOMplex. The WOMbat daemon sends a volley of ICMP echo request packets (ping) to the client IP address and measures the round-trip time of any responses that return within a pre-specified time-out period.

WOMbat daemons send back to the WOMbat Driver the average network response time taken to ping the client machine. The WOMbat Driver waits for a specified length of time for the WOMbat daemons to respond, then determines which server is closest to the client by comparing the responses. Server availability information also can be factored into this calculation, so that the closest server might be rejected in favor of a more distant, but less busy server. The user's profile in the GPD contains the identity of the preferred server.

The next time the user requests a page from the Web server, the server daemon interrogates the GPD using the browser Cookie as a key, and discovers that the preferred server for that client is not this server. The WOMserver then builds the requested page for the client, but rewrites all embedded links to follow-on pages and to

In its DNS emulation mode, ISS associates more than one IP address with any given Internet name.

embedded images as URLs on the preferred server rather than on local links. For example, if the preferred server was www.hursley.atlanta.olympic.org and the local link would normally have been the following:

```
<IMG SRC="/data/image1.gif">
```

then, the link sent back to the client is rewritten as follows:

```
<IMG SRC="http://www.hursley.atlanta.
olympic.org/data/image1.gif">
```

When the client browser follows any of these links, they are retrieved from the preferred server. From that point on, all links will point back to that server, so the client has effectively been "moved" to that server.

WOMplex separates logical services from physical servers, which allows a Web site to be hosted from any site in the WOMplex.

Virtual Addressing

AIX and other UNIX operating systems can alias multiple IP addresses onto a single network interface; that is, the server can "pretend" to have several different IP addresses on the same network at the same time. This was used two ways in the Atlanta WOMplex: multi-homing and service virtualization.

Multi-Homing. A single WOMserver instance can serve several Web sites. The IP site addresses are aliased onto the server's Internet network interface. The WOMserver configuration has separate "trees" of URL data for each hosted address. When a Web page request arrives, the WOMserver interrogates the socket connection to identify which IP address it was sent to, then serves the appropriate page from the correct data tree. Many "standard" Web servers consider this a standard feature, such as Virtual Hosting in Apache.

Service Virtualization. WOMplex separates logical services from physical servers, which allows a Web site to be hosted from any site in the WOMplex. IP aliasing within a site separates logical from physical and frees any dependence on specific hardware. All externally visible IP addresses are aliases, not the "real" base IP address of any machine. Therefore, if a server fails, the service IP addresses supported by the failed server can be transferred readily to another machine. One additional requirement, apart from adding the IP address aliases, is to flush the arp cache on the nearest network router, which forces it to reestablish the new IP to MAC address mappings. The System Network Management Protocol (SNMP) interface to the router can often automate this operation.

Global Profile Daemon

The Global Profile Daemon stores and caches client profile information, using a client browser Cookie as a retrieval key. Any WOM application or the WOMserver itself can store profile information in GPD. Profile data is held as a file of lines containing `name = value` pairs. Keys used for the Atlanta WOMplex were constructed from the concatenation of the following:

- Three randomly chosen letters
- Cookie creation timestamp (seconds since Epoch, plus an offset)
- Cyclic Redundancy Check (CRC) checksum digits

GPD uses a multi-level cache to balance the global distribution requirements of profile data and fast access to user profiles by the WOMserver. Closest to the WOMserver, GPD maintains an in-memory cache with a Least Recently Used (LRU) policy; therefore, people who are known to be currently moving around the Web site are most likely to be in memory.

To locate profiles efficiently, the directory structure has directory names 'A' through 'Z'—three levels deep. For example, the profile with the key ADF1234567123 would be found in the file with the following full path: /dfs/profiles/A/D/F/3217654321FDA.

Note that the name given to the profile file was the reverse of the key string, that is, 3217654321FDA. This is because DFS uses its own hashing algorithm for efficient searching of directory contents—based on the first character of the filename. Consequently, the first character of the filename should be a random character. This avoids having all files in a directory starting with the same first character, and thus all falling into the same hash bucket. We chose three directory levels and 26 entries per level, knowing that DFS caching is optimized for a certain maximum directory content size.

At the next level, DFS caches recently retrieved profiles on the local disk of the server machine. At a site level, recently accessed profiles are stored in a DFS fileset that physically resides on disks at that site. This avoids trans-Atlantic connections for many profiles. Finally, the main DFS server at the primary WOMplex site in Southbury stores the global GPD repository.

When a client comes back to the Web site (at any physical location) with a previously assigned Cookie, the WOMserver requests information on that client's profile from its local (on the same machine) GPD instance. GPD checks the in-memory LRU cache and returns the information if it is there. If not, GPD must fetch the profile from the global repository because that client has not been to this physical site recently.

GPD reads the profile from the global DFS directory and copies the file to a local DFS fileset located physically on the local site. Then GPD sends a UDP broadcast to other servers at its local site informing them that the client with a specified Cookie has come to the site, and may soon request pages from them. Other GPD instances at that site receive the broadcast and go to the local DFS fileset to read the profile information into the memory cache. Thus, only the first GPD at a site has to do the trans-Atlantic journey to get the profile data of a returning client. The other servers (who are likely to receive requests from that client in the near future, thanks to the WOMsprayer) pick it up from a fileset that is local to their physical site.

The CRC digits are built into the Cookie. This makes it more difficult for anyone to tamper with a Cookie while trying to obtain profile information about another user by reverse engineering their Cookie. Cookies with invalid checksums are not accepted by the WOM system.

Applications

This section describes applications that were designed and built for the Atlanta Web site. For various reasons, not all of these went into full production on the site. They were all working in the lab, so we have included them here for the sake of completeness.

Sneak Peek

Sneak Peek was the largest "Net Cam" ever. Live video feeds from 38 locations at the Atlanta Games were fed to "The Fishbowl" in Atlanta. Here they went through a bank of "frame grabbers" and were checked into the WOM system. Images were captured approximately every five seconds from each feed. Because of the high volatility of this data, the images were not stored into the WOM DB2 database, but went directly into DFS. HTML pages to render the latest set of images and to navigate between Olympic venues were generated on-the-fly to pick up the most recent image file names.

The human operator in the Fishbowl received all feeds concurrently and could select channels that currently had activity on them. The navigation options reflected the choices presented by the dynamic HTML pages in the Sneak Peek section of the site. The operator could choose the most stunning images and build a daily "scrapbook" of images worth keeping. These images were titled, cropped, assigned metadata using the WOMpix tool, and checked into the main WOM database.

Sneak Peek images were not mirrored across the world because of their highly volatile nature—they were all served from the primary WOMplex site at Southbury. This became an extremely popular application, and many hundreds of thousands of images were downloaded.

Bamba

Bamba is IBM's streaming audio and video technology for the Web based on G.723 low bit-rate streaming standards. For more information about the technology and to download the decoders and encoders, visit the Bamba home page on the IBM AlphaWorks™ Web site (http://www.alphaworks.ibm.com/).

Sneak Peek images were all served from the primary WOMplex site at Southbury.

Bamba has three advantages over other Internet audio and video formats:

- Data is streamed, so you can view and download it simultaneously, avoiding huge waits while the whole file downloads.
- The content is encoded at a low bit rate, making it suitable for transfer over a standard 28.8 Kbit modem.
- G.723 is a standard.

Three forms of Bamba content were used at the Atlanta Olympics:

- **Audio clips:** Recordings of many interviews with athletes, Olympics organizers, and news reports. A regular Web server handles Bamba audio clips in the same way as GIF images. On the Atlanta site, they were one data type served by the binary server.
- **Video clips:** Encoded highlights of the sports events. Bamba video clips are also regular binary content that can be served by an ordinary Web server. They were served from the binary server during the Olympics.
- **Live audio:** Atlanta local radio station, WGST Atlanta, was broadcast over the Internet 24 hours a day during the Olympics. Bamba

Reflector allowed many people to listen to the radio station concurrently.

Bamba Reflector

In typical use, a small Pentium™ PC would do real-time Bamba audio encoding. It is not realistic to have hundreds or thousands of clients connecting to this machine to listen to the audio stream, so Bamba Reflector was created. The Reflector connects to the transmitter as the only client to that machine. It then accepts connections from Bamba clients. Whatever the Reflector receives from the transmitter is transferred to the clients. There must be a certain amount of buffering and flow-control to deal with clients connected over different speed links. The Reflector intelligently drops Bamba packets without losing stream synchronization for any clients who fall too far behind in receiving Bamba packets. Each Reflector can support many hundreds of client connections concurrently. One Reflector can act as a source to other reflectors, so a whole world-wide tree of Reflectors can potentially serve many thousands of clients across the globe.

The Reflector configuration can limit connections to a specified upper limit or it can be off-line—not broadcasting at the moment. When a connection is not possible, it can send out an information message in one of three forms: a pre-recorded audio message built into the client, a text message sent from the Reflector, or an audio message sent from the Reflector. Various combinations are also possible.

ISS load balancing can be used in front of a bank of Bamba Reflectors to balance incoming connection requests to a single advertised Internet name across the available servers to ensure an even load across the Reflectors.

Users could specify to the WOMserver the specific sports, countries, and athletes of interest to follow as the Games progressed.

WOMplex Boiler Room

The Boiler Room monitoring system allowed statistics to be gathered from various parts of the WOMplex, its applications, and subsystems. The system structure is described below.

Each application to be monitored had a `funneld` (funnel daemon) client. To make this a simple task, developers had client code for Java, Perl, and C available. The protocol used throughout this system was Internet Relay Chat (IRC), transmitting data in a textual format with each monitored system using a different IRC channel number. The client code connected to a `funneld` instance ran on each server. The `funneld` instances connected to other `funnel` daemons in a tree structure. An IRC server was at the root of the tree of `funnel` daemons.

The Boiler Room monitors were Java applets embedded in HTML pages on the WOMplex Web site. These applets used IRC client classes written in Java that connected back to the IRC server. The monitor applications then tuned in to the IRC channels required to construct their displays and waited for information to arrive. A variety of dials, counters, and indicators were implemented as Java classes that could be combined in various ways to display the statistics for any system.

Since the Boiler Room system could also tail log files and receive data through sockets and Interprocess Communications (IPC) message queues, it was easy to hook up monitors for almost anything. A special *chart recorder* client logged IRC channel data for subsequent analysis.

Information Taxonomy

Early in this project we realized that a very large site needs some kind of information taxonomy, or hierarchy, mapped to the site; otherwise, it becomes completely unmanageable and almost impossible to navigate. This important issue required the skills of an information architect.

The metadata for every page, image, and multimedia clip checked into the WOM database included a set of category tags. These tags position the object in the *information space* of the Web site. This information space is dimensionally exclusive from the *data space* imposed by the HTML hyperlinks built into the documents. Tools, such as the Wu-tool, categorize pages on the site. Once the whole site has been categorized, facilities can be provided to navigate the site in information space, as well as the traditional HTML navigation. For example, someone looking at pages about Men's Canoeing might also be interested in other pages that are "close" in the information space. For example, in this case, Kayak events might be of interest.

As an interesting new area for Web site navigation and management, this will undoubtedly play an important role in the future.

Personal Home Page

The combination of generating dynamic pages in the WOMserver and identifying individual clients from their browser Cookie allowed a personal home page to be created for each visitor to the site.

Users could specify to the WOMserver the specific sports, countries, and athletes of interest to follow as the Games progressed. If they provided

Secure SHell (SSH) (described below) for secure connections. The Operations Center in Southbury could control every processor in the WOMplex—right down to power cycling.

ATM Network

The outbound network for data from the Web site was routed over a 100 Mbits per second ATM network. ATM uses centralized switching technology to provide a high-bandwidth network. The site networking was highly complex, with multiple Token-Ring, Ethernet™, and ATM networks connected to the Internet through two filtering routers, IBM 8260 switching hubs enabled this level of connectivity. A 16 Mbits per second Token Ring handled the inbound networking. *Note:* Network requirements for a Web site are highly asymmetrical—one packet that contains an HTTP request can provoke the transfer of potentially many megabytes of data from the servers.

Transarc DFS Cell

Transarc's Distributed File System (DFS) ensured that all WOMplex Web servers had access to up-to-data information. (For those familiar with Andrew File System (AFS®), think of DFS as the "grown up" AFS.) DFS is based on the fileset concept, a collection of files handled in the same manner as a single group. Each fileset in a DFS cell resides on a nominated server; it may have mirrors on other nominated servers. Fileset replication happens in two ways: on a timer interval or (often more useful) on a manual "push" from the primary server, when the application or user knows that a batch of updates has been completed.

Each WOM server in the WOMplex was a DFS client configured to request files from the nearest DFS server. At remote mirror locations, the clients retrieved their data from a local server that was kept in sync with the main servers in Southbury. One big advantage of DFS in the Web server environment is client caching. During the Olympics, most clients achieved a 98% cache hit rate.

The primary DFS server in Southbury was distributed across three RS/6000 Model 570 servers and two RS/6000 SP wide nodes. It used 128 GB of IBM 7137 RAID storage, configured with replication and mirroring to give 32 GB of effective storage with three-way redundancy in addition to the RAID failsafe.

The entire cell was administered centrally from Southbury. Problems with filter rules on routers and TCP/IP routing across the Internet make setting up a very large distributed DFS cell a complex process; although when it is running, the system works extremely well. The single DFS cell spanned six locations: Southbury (the main server), Cornell, Keio, Hursley, Karlsruhe (the mirror sites), and Atlanta, where the content was being generated for the Web site.

Internet Connectivity

To ensure the availability of enough bandwidth for the Olympics, four DS3 circuits (T3—45 Mbits per second) were connected into four major U.S. Network Access Points (NAPs): Mae-East, Mae-West, Sprint NJ, and Ameritech™ Chicago. Regardless of the origin of a request, by using a routing algorithm that minimized hop counts, the response was sent to the major NAP closest to the source of the request. Most of the world's major non-U.S. service providers connect directly into one of these four NAPs, so the site was well connected. Multiple circuits also protected the site against any link failure. Packets were automatically routed over other links or could be manually routed around trouble spots, which occurred from time to time at various NAPs.

Security

The entire Atlanta Internet site, the software development environment, and all DFS infrastructure were located outside the IBM firewalls. Consequently, site security was always a high priority for everyone because such a high visibility site was an obvious target for hackers.

High security must be built into a site from the ground up.

High security must be built into a site from the ground up; it cannot be retrofitted onto an existing site. The filtering routers provided the main protection against invasion, and the filter rules for these machines were a major design and maintenance effort. Interworking with the remote mirror sites involved setting filter rules at the remote sites to allow only User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) packets on specific port numbers in and out from specified Internet Protocol (IP) addresses.

A major problem for the primary site at Southbury was the need for extensive access to systems inside the DMZ ("De-Militarized" Zone) at the remote sites; the rest of the world could only have port 80 World Wide Web access. Heterogeneous router technology across the world ensured that many people learned a lot about writing filter rules in a short space of time!

Kerberos and authentication servers were used to secure DFS. Sophisticated password schemes for

broadcast. The client applet received the play list and modified its display behavior accordingly, adding new items as they were transmitted and deleting old items that were no longer in the play list. The producers relied heavily on the information taxonomy to determine which new items being checked into the site were of interest to their programs.

The client applet used HTTP to retrieve non-text items, such as images, from the main Web site, using a URL sent over IRC. The applet also used HTTP to retrieve a file containing the initial set of current content when the applet was first invoked.

Conclusion

The Atlanta Olympics Web site was the largest Web site in the world and handled an immense amount of traffic during the 1996 Summer Games. The WOMplex technology was pioneered to support large sporting events on the Internet. The Olympics gave the WOMble Team a chance to push forward the technology frontiers, and has paved the way for products using WOM technology in the future.

The WOMserver was designed as an Internet application platform, of which a sporting event server was just an application. The suitability of this architecture as an application environment has been proven in the "trial by fire" experienced during the Olympics.

This experience greatly advanced load balancing and global scalability. The systems deployed for the Olympics showed that a Global WOMplex is a viable concept.

The author gratefully acknowledges the contributions of Sean Martin and David Grossman to this article, and also wishes to thank the Team at the IBM Southbury lab.



Andy Stanford-Clark, IBM Corporation, 150 Kettletown Road, Southbury, CT 06488. Internet: andysc@vnet.ibm.com. Dr. Stanford-Clark is an Internet advanced technology consultant in IBM's Internet Division. His responsibilities include exploring emerging Internet technologies and working with customers to implement Internet-based projects. Dr. Stanford-Clark is currently in the U.S. on international assignment from IBM Hursley, U.K. He has a BS in Computing and Mathematics and a PhD in Parallel Computing from the University of East Anglia in the U.K.

Exhibit 5

IBM

Think Research

FEATURED CONCEPT

Managing the Web

In Brief:

Handling a high-profile Web site can be a Webmaster's nightmare. However, the Atlanta Olympics Web site at the 1996 summer games ran like a dream, handling nearly 200 million hits over the course of 17 days. It was created and managed using a new, evolving IBM technology called Web Object Manager (WOM). This builds pages dynamically from "objects" stored in an underlying database as they are requested, even customizing them to the needs of the user or the characteristics of the clients browser. The Olympics site had some 35,000 objects. WOM also incorporates a new approach to load balancing and scalability, which allows additional processing power to be easily added to an existing site.

In what seems like an almost inconceivably short time, the Internet has changed from a useful resource for the few to an indispensable tool for the many. Much of the credit goes to the introduction of the World Wide Web, which provides hyperlinked access to data and programs on a vast number of servers around the world. Yet, despite their growing centrality in our lives, both the Internet and the Web remain mysterious, more easily talked about and used than understood. Nor is that surprising. One cannot point to the Internet and say it is here or it is there. Rather, like the ghost of Hamlet's father, it is here and there and, indeed, everywhere. It is natural, then, that efforts to characterize the advance of Internet technology tend to seize on the most visible elements. Browsers, new content and electronic commerce along with the perennial quest for greater bandwidth appear to be the key factors determining the future of the Web. While no one would deny that these are among the forces driving the popularity of the Internet, the continued expansion of the Web depends on addressing a quite different set of issues.

As more than one Webmaster charged with creating and running a Web site has discovered, popularity can pose problems of its own. The goal of managing content and presenting site "visitors" with information tailored to their needs grows ever more elusive as the site expands, and as the number of users increases. The more pages a site creates, the more work there is in updating them,

ensuring consistency, and simply keeping track of whats already been created. At the same time, as Web sites are called on to handle not just a greater number of daily hits for simple files but requests that require the server to execute programs such as traditional Common Gateway Interface (CGI) scripts the pressure on resources can soar. Web servers designed to meet an anticipated level of service may work admirably up to a point and then stumble or crash under a sudden flurry of hits because they are unable to grow, or be scaled up, in a simple way.

Content management and scalability are two of the key problems that the Internet now faces. What is the solution? The desperate one of simply throwing more programmers and more hardware at the problem can be extremely costly and inefficient: ultimately, the need for new content and updates will always outstrip the practical resources available, whereas hardware brought in to handle peak loads sits by idly the rest of the time. The first steps toward a novel solution began to emerge at IBM in 1994. Sean Martin, a member of IBM's UK Client Server Group based in London, and Frank Schwichtenberg, of IBM Germany, helped create a Web site AixServe for the Europe-wide Installation Support Center that would provide product information and technical help. As he focused on the task, Martin was struck by the existing limitations of Web servers.

A Web site, he realized, is essentially a collection of digital files texts, images, programs, and so forth that are typically treated as independent entities. The same navigation bar, for example, may appear on all the pages, but it has to be recoded each time. In the course of discussing the problem with IBM Austin's Rob Gore, the architect of AixServe, it dawned on Martin that there was a better way. How much simpler, he reasoned, if the pieces needed to assemble a page were regarded as the basic entities, and pages were created from a pool of reusable parts each time they were requested.

By the end of 1994, Martin and Schwichtenberg had implemented the first version of a technology that treated Web pages as entities that can be degenerated dynamically on the fly (see "Serving Web Requests on the Fly," below). That pioneering Web server was the basis of what has come to be called Web Object Manager (WOM).

A match for any sport

In early 1995, Martin gave a talk at an internal Internet meeting in Thornwood, New York. As a result, he soon received an invitation to work on IBM's 1996 Summer Olympics project, which brought together leaders in Internet technology from around the company to create the games official Web site (see "An Olympic Victory," below). The group "warmed up" by working on other sporting events, starting with a Web site for the 1995 Wimbledon tennis tournament in June.

New challenges they faced at that time included the quirkiness of different browsers: not all could display Web pages created with a given version of Hypertext Markup Language (HTML), the set of

tags used to format a Web page. "For Wimbledon," says Martin, who is currently in the United States on assignment at the Internet Division, "we added a new level of personalization to the capability of dynamically generating pages on the fly. By merging a request for specific information with the required HTML format required by a clients browser, a customized page was sent to the user."

It became immediately apparent that dynamic page generation was especially attractive for such rapidly changing sites, in which scores need to be continually updated. Together with Andy Stanford-Clark a member of the UK's Hursley Laboratory currently on assignment at the Thomas J. Watson Research Center Martin wrote a new version of WOM for the PGA golf tournament in August, and still another version for the U.S. Open tennis tournament the following month. A core technical team at Watson has continued to develop the underlying concepts of WOM, and Chet Murthy has now rewritten the code in Java on top of IBM's DB2 relational database. As it evolves, the advantages of WOM's ability to simplify content authoring and management has become increasingly evident. "Beyond the immediate benefits to Webmasters, the kind of personalization WOM makes possible represents a huge extension of the most natural and desirable way of doing business" says Scott Penberthy, one of the WOM team co-leaders at Watson. "In the long run, it will allow companies to have a one-to-one relationship with customers on a massive scale."

Balancing the load

Popular sporting events make for popular Web sites. In addition to the normal heavy demand experienced by the site servers, key matches often generate huge peaks. WOM itself adds to the computational load because of its dynamic page generation capability. With the request to build multiple sporting event Web sites staring him in the face, Martin began thinking about ways to allow the server to run on multiple processors.

For Wimbledon, there wasn't time to do more than modify traditional load-balancing techniques, which were used to shift requests between a server in White Plains, New York, and one at the Hursley Lab, in the south of England. But Martin did introduce the first version of WOMbat. This technique ensures that each time a request is received from a new user, all the servers supporting a site "ping" the client machine just like a bat. The first server to receive an "echo" then becomes the default server for that user. In August, while preparing for the PGA Golf tournament, Martin teamed up with Stanford-Clark to design a more general load-balancing scheme. The previous January, Stanford-Clark, then at Hursleys Commercial Parallel Center building parallel applications for customers using the IBM SP scalable parallel system, had had his own epiphany. "I saw that, as people began putting up Web sites with more advanced applications, such as catalogs, there would be a need for a scalable Web server," he recalls.

Using load-leveling software called Interactive Session Support (ISS) that Stanford-Clark and Graham Wallis had written a couple of years before, Martin and Stanford-Clark designed and built a scalable Web server based on an SP. "ISS allows a number of computers such as the various processor nodes of an SP to behave as though they were a single machine," says Stanford-Clark. A key additional component, which enabled essentially unlimited scalability, was a new kind of router code recently renamed Network Dispatcher that had been developed at Watson earlier (see "Scalability without Limits,"below).

Linking to legacy data

In the course of its evolution, the vision of what WOM will enable has considerably enlarged. Martin credits Murthy with making him aware of WOM's true potential. "Chet saw how WOM might become an Internet middleware layer that would allow applications to find their home in the WOM programming environment," he explains.

In Murthy's vision, WOM provides the means to link users to legacy, or enterprise, data most of which runs on mainframes. Such a link is important both for intranets companywide networks based on Internet technology as well as the Internet as a whole. "For example," says Jim Russell, the other co-leader of the Watson WOM team, "it might be desirable to allow employees to access their 401K account data online, not only to look at their balance but to modify their allocations. That means having an intranet application that enables one to access the database, have a "page" of data sent to one, modify it and send back the updated version."

Potentially, transactions over the Internet could be even more far-reaching. They could range from generating a personalized "catalog" from a retailers stock to transferring funds in and out of ones bank account or making and paying for an airline reservation. For that to happen, two things are needed: gateways to access legacy data and Web applications that have the robustness and the reliability that users have come to expect for their mission-critical applications. "When one thinks about it," says Murthy, "IBM built the original intranet transaction system inside the company a long time ago, with distributed clients running all around the world using CICS (Customer Information Control System) to carry out transactions remotely and interact with mainframe databases. Our goal is to merge the acknowledged capabilities of CICS and its associated legacy software with the world of new media content." WOM technology is becoming available to selected customers who face the challenge of running large, complex Web sites. As the rollout continues, WOM will inevitably continue to grow closer to realizing the vision of its developers and in the process change the nature of networked computing.

More Information:

Serving Web Requests on the Fly

Scalability Without Limits

An Olympic Victory

Serving Web Requests on the Fly

When a traditional Web site receives a request from a browser, it returns a previously composed page, which may be one of hundreds or thousands. Each page stored at the site must be created separately, even though certain elements of the pages such as a common site logo, the navigation bar, and so on may be the same. In some instances, a users request requires the output of a Common Gateway Interface script a program that can be invoked from a browser to run on the server.

Web Object Manager (WOM) approaches both flat files and scripts in a very different way. It replaces the concept of a Web page with an object called a canvas, which provides a common look and feel to all the pages served by the Web site. The canvas contains special tags for embedding page parts that constitute the various components of the page. For example, page parts such as headings, formatted in Hypertext Markup Language (HTML), are common to every page. Other HTML parts contain content specific to the page or to the user who requests the page. Finally, there are parts that invoke Java code that can perform business logic functions, such as querying a database and returning the data in an HTML format.

Each page part also contains a pointer that enables the actual part to be retrieved on the fly from an underlying database. The fully constituted page is then sent to the user. Page parts not only enable reuse of common presentation and business logic elements across multiple applications, but they also allow automatic customization. For example, a single page part representing a companys balance sheet might render itself as an HTML table or a Java spreadsheet, depending on the capabilities of the Internet browser or on the users preference.

In addition to the page parts themselves, which constitute the primary data, WOM stores "metadata" about the objects. That can include a variety of information, such as the name of the author who created the part, the date when it was created, the date after which it is no longer valid, lists of names of system users authorized to modify the part, and status information indicating whether the part is being modified, reviewed and so forth. Different versions of a part can be tracked in this way, and the entire task of managing and updating the content of a Web site can be greatly simplified. WOM can also hold arbitrary metadata associated with a part that can be used by an application, such as showing confidential information only to approved users.

Scalability Without Limits

The goal of scalability is to provide sufficient resources to meet demand seamlessly, without any interruption of service or inconvenience to the client. For that reason, simply replacing a computer with a more powerful one is not a form of scalability, since in most cases that single machine will itself run out of steam at some point.

Traditionally, Web sites have run on a single server. Individuals wishing to access the site must know the name of the server. When the Internet name is entered by the person interacting with the Web through a browser, the name is sent to a domain name server (DNS) somewhere in the network, which "resolves" the name into an Internet Protocol (IP) address and then sends it back to the users machine, the client. It is that address that is then sent on to the server to initiate a session with a particular Web site.

Inherent in this scenario is the principle that every server has one and only one address. If one adds additional servers, one would need to share a single address among multiple machines. One would also need to implement a means of sharing the load the incoming requests among the various servers. IBM implemented such a scheme to meet these requirements for its Olympic Web site.

Once the address is returned to the client by the DNS, the request is sent to a machine running a software technology known as Network Dispatcher (ND). ND is not itself a Web server and so cannot return any data. "What it does" says Andy Stanford-Clark, who helped develop the scalability scheme "is pass on requests to backend servers. In this way, multiple servers can share a single Internet address."

The ND also runs a load-balancing software program called Interactive Session Support (ISS). Each backend server runs an ISS agent, which provides feedback to the ND about the current load of the server. The ND then passes on requests in a weighted round-robin fashion, starting with the server having the smallest load.

An ND and the backend servers constitute a cluster, and when needed as did www.atlanta.olympic.org during the 1996 summer games a Web site can be scaled even further by interconnecting a group of clusters, which can be at widely separated locations. In this configuration, the ISS can emulate the function of a domain name server. However, unlike an ordinary DNS, ISS enables the DNS to associate a single Internet name with more than one IP address, each of which is that of a particular cluster.

The clusters ISS agent sends continual updates on the status of the cluster to ISS. When ISS receives a name resolution request from a client, it determines which IP address should be returned. "In this way," notes Stanford-Clark, "a single logical Web site defined by a specific www... name is distributed across multiple physical

locations."

An Olympic Victory

To a degree unusual for technology barely out of the prototype stage, the concept of the Web Object Manager (WOM) and the associated ideas for scalability and application development have undergone a remarkable proof of concept. In the spring and early summer of 1996, an IBM team based in Southbury, Connecticut, used WOM to build the official Internet Web site of the Atlanta Olympic Games as part of IBM's corporate sponsorship of the event.

The site comprising 50 nodes of a distributed IBM RS/6000 SP scalable parallel system spread over five locations and four countries easily qualified as the largest Web server in the world. Over 17 days, it logged 192 million hits, with a peak of nearly 17 million in a single day. And each of these pages was generated dynamically.

Even before the Olympics opened, a server based on the IBM Net.Commerce server was used to sell more than $5 million worth of tickets to events at the games over the Internet. During the games, a results system and server technology developed by a team at Watson led by Paul Dantzig made it possible to provide Internet users with real-time results for all of the 37 sports.

Most of the processing power was located at Southbury. But the four mirror sites at Cornell Theory Center in Ithaca, New York; the IBM Hursley Laboratory; the University of Karlsruhe, near Mainz, Germany; and Keio University, near Tokyo, Japan served nearly exact replicas. The Sneak Peek application, however, which furnished a constantly updated supply of images from 38 venues at the rate of 21,000 per hour, was maintained only at Southbury. Audio and video clips, as well as live audio from WGST Atlanta, were also made available using a streaming technology known as Bamba (see "Net Results").

Personal home pages could also be created for visitors, based on WOMs dynamic-page-generation capability and information supplied by a users browser "cookie," which identifies the end user. By prompting users to specify the sports, athletes and countries they were most interested in following, a Home Page application would build a personalized page. Another personal touch was provided by an online data mining tool that tracked users "clicks" and streamed all the data into DB2 tables by means of an 8-way IBM System/390® Sysplex. The information was then used to compile a user profile so that a person could request links to information based on his or her browsing patterns.

A particularly useful, if not necessary, requirement for being able to navigate, as well as manage, a large site is a way of categorizing the various pages. Traditional HTML hyperlinks provide one way of doing this. But the metadata category tags in the WOM database made it possible to build an "information space," in which pages that contain conceptually similar material

would be found "close" together. That would allow someone following, say, the Canoeing pages to readily look at those related to Kayaking, although there might be no HTML link between them.

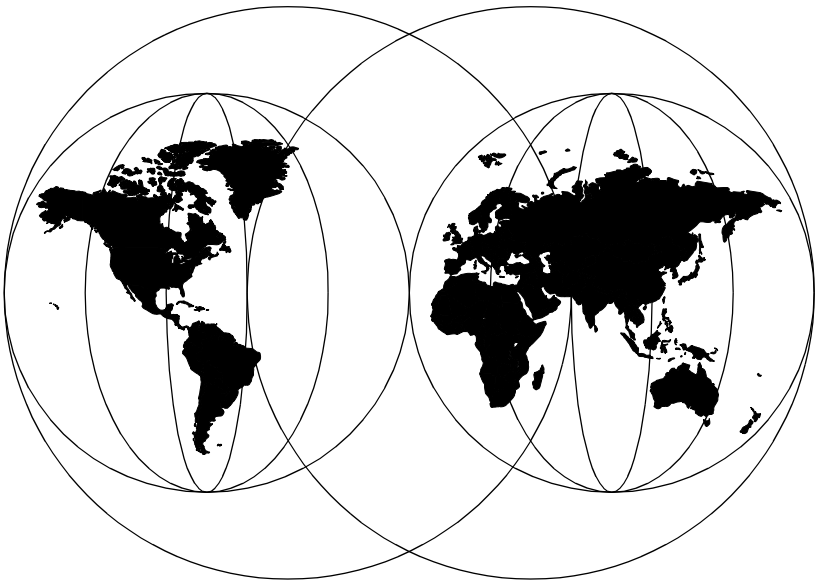
These features represent just a few of the applications developed for the Olympics site. They illustrate the power of the WOM architecture to enhance the usefulness of the Web and simplify its management.

Exhibit 6

SG24-4993-00

Load-Balancing Internet Servers

December 1997



IBM

International Technical Support Organization
Austin Center

SG24-4993-00

International Technical Support Organization

Load-Balancing Internet Servers

December 1997



Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix D, “Special Notices” on page 131

First Edition (December 1997)

This edition applies to IBM Interactive Network Dispatcher Version 1.2 for AIX.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Building 045 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1997. All rights reserved
Note to U.S Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Contents iii

Figures vii

Tables ix

Preface xi
How This Redbook is Organized xi
The Team That Wrote This Redbook xiii
Comments Welcome xiv

Chapter 1. The Big Picture 1
1.1 Internet and Intranet Sizing 1
1.2 Sizing for Growth 2
1.3 Dealing with the Growth 3
1.4 Interactive Network Dispatcher Overview 4
1.5 Why Do I Need Interactive Network Dispatcher? 5
1.6 Where Interactive Network Dispatcher Is Being Applied 6
1.7 Basic Terminology 7
1.7.1 Interactive Network Dispatcher 8
1.7.2 Hypertext Markup Language **(**HTML) 8
1.7.3 Hypertext Transport Protocol (HTTP) 8
1.7.4 World Wide Web (WWW) Server 8
1.7.5 File Transfer Protocol (FTP) Server 8
1.7.6 Domain Name System (DNS) Server 8

Chapter 2. Interactive Network Dispatcher Concepts 9
2.1 Introduction 9
2.2 Interactive Network Dispatcher Functions 9
2.2.1 The Network Dispatcher Function 9
2.2.2 The Interactive Session Support (ISS) Function 9
2.2.3 How Network Dispatcher Works 10
2.2.4 How Interactive Session Support (ISS) Works 12

Chapter 3. Interactive Network Dispatcher in Detail 17
3.1 The Network Dispatcher Function in Detail 17
3.1.1 Interaction between Network Dispatcher Components 17
3.1.2 Controlling Network Dispatcher Components 19
3.1.3 Proportions 21
3.1.4 Ports Used by Network Dispatcher 22
3.2 Interactive Session Support (ISS) Function in Detail 23
3.2.1 ISS Cells 23

© Copyright IBM Corp. 1997

3.2.2 ISS Observers . . . 24
3.2.3 ISS Selection Methods . . . 27
3.3 ISS Configuration File . . . 28
3.3.1 Cell and Its Attributes . . . 28
3.3.2 Node . . . 30
3.3.3 ResourceType . . . 31
3.3.4 Service . . . 31
3.3.5 Observers . . . 32
3.4 Metrics . . . 32
3.4.1 RoundRobin . . . 32
3.4.2 Internal Metric . . . 34
3.4.3 External Metric . . . 35
3.4.4 Writing Your Own Metrics . . . 36

Chapter 4. Installation and Operation . . . 39
4.1 Interactive Network Dispatcher User's Guide . . . 39
4.2 Installing for AIX Installing for AIX . . . 39
4.3 Configuring Network Dispatcher . . . 40
4.4 Configuring the Interactive Session Support . . . 42

Chapter 5. WWW Server Scenario . . . 47
5.1 Testing Environment Considerations . . . 47
5.2 Load Generated by WWW Servers . . . 48
5.3 How the Tests Were Performed . . . 48
5.4 The Tests and Their Results . . . 49
5.4.1 Round-Robin Configuration . . . 49
5.4.2 ISS with Best SelectionMethod . . . 51
5.4.3 Network Dispatcher . . . 54
5.5 Conclusion . . . 59

Chapter 6. FTP Server Scenario . . . 61
6.1 Testing Environment Considerations . . . 61
6.2 Load Generated by FTP Servers . . . 61
6.3 How the Tests Were Performed . . . 63
6.4 The Tests and Their Results . . . 63
6.4.1 Testing with ISS . . . 63
6.4.2 Round-Robin Configuration . . . 64
6.4.3 ISS with Best SelectionMethod . . . 66
6.4.4 Network Dispatcher . . . 68
6.5 Conclusion . . . 70

Chapter 7. Other Server Scenarios . . . 71
7.1 Telnet Server Scenario . . . 71
7.2 Domino Server Scenario . . . 72

Chapter 8. High Availability Server Scenario 75
8.1 ISS High Availability 75
8.2 Failure of a DNS Server 75
8.2.1 One Possible Solution 77
8.2.2 Pros and Cons 78
8.3 Network Dispatcher High Availability 78
8.3.1 Configuring Network Dispatcher High Availability 79
8.4 Testing Environment 81
8.5 Conclusion 85

Chapter 9. Two-Tier Network Scenario 87
9.1 Front-End Tier Configuration 89
9.2 Back-End Tier Configuration 91

Chapter 10. Configuration Files - Samples 93
10.1 The Test Environment 93
10.2 Network Dispatcher Configuration 97
10.3 ISS Configuration 101
10.3.1 RoundRobin 101
10.3.2 Best - SelectionMethod 103
10.4 High Availability Configuration 106

Appendix A. Minimal Configuration 111
A.1 ISS 111
A.1.1 RoundRobin Metrics 111
A.1.2 Custom Metrics 112
A.2 Network Dispatcher 112

Appendix B. Domain Name System (DNS) 115
B.1 Naming 115
B.2 Naming Authority 115
B.3 Naming Conventions 116
B.4 Name Servers 117
B.5 Planning for Domain Name Resolution 118
B.6 Configuring Name Servers 119
B.6.1 Configuring a Primary Name Server 120
B.7 Delegating Domains to Other Name Servers 124

Appendix C. Hints and Tips 127
C.1 ISS 127
C.2 Network Dispatcher 128

Appendix D. Special Notices 131

Appendix E. Related Publications 133
E.1 International Technical Support Organization Publications 133
E.2 Redbooks on CD-ROMs 133
E.3 Other Publications 133

How To Get ITSO Redbooks 135
How IBM Employees Can Get ITSO Redbooks 135
How Customers Can Get ITSO Redbooks 136
IBM Redbook Order Form 137

Abbreviations 139

Index 141

ITSO Redbook Evaluation 145

Figures

1. Network Dispatcher with ISS Providing Servers Load Information 12
2. Multiple Services Appear as One Virtual Server 14
3. Typical Flow of Information in Network Dispatcher 17
4. ISS Configured with a NameServer Observer 25
5. ISS Configured with an ISSNameserver Observer 26
6. ISS working with a RoundRobin SelectionMethod. 33
7. ISS Using an External Metric 36
8. Sample Network with Network Dispatcher and a Group of Servers 41
9. CPU Load - NameServer Observer: Round-Robin for Dynamic Pages. . . 50
10. Network Traffic - NameServer Observer: Round-Robin for Static Pages . 51
11. CPU Load - ISS with Idle CPU ResourceType for Dynamic Pages. 53
12. Network Traffic - ISS with Idle CPU ResourceType for Dynamic Pages . . 54
13. CPU Load Using Network Dispatcher for Dynamic WWW Pages 56
14. Network Traffic Using Network Dispatcher for Dynamic WWW Pages . . . 57
15. Network Traffic Using Network Dispatcher for Static WWW Pages. 58
16. Interaction of the Client and Servers with File Transfer Protocol (FTP). . . 62
17. Network Traffic- ISS with the Round-Robin Configuration. 65
18. CPU Load - ISS with the Round-Robin Configuration 67
19. Network Traffic - Network Dispatcher - FTP Simulation of Load 69
20. Configuring a Client to Use Multiple Name Servers. 76
21. Example of Multiple ISS Servers in a High-Availability Configuration 77
22. Network Dispatcher High-Availability Feature 79
23. Network Traffic with a High-Availability Configuration 83
24. CPU Load with a High-Availability Configuration. 84
25. Redirecting Requests on a Two-Tier Configuration of Clusters. 88
26. ISS Front-Tier Configuration 91
27. Different Configuration Option for Network Dispatcher and ISS 92
28. SP Configuration Used for Interactive Network Dispatcher Tests 94
29. Example of Minimum Configuration for ISS in Round-Robin. 112
30. Example of Domain Structure 116

© Copyright IBM Corp. 1997

Tables

1. Summary of Functions for Network Dispatcher Components 18
2. Possible Sources for External Metric Values 37
3. ISS Round-Robin FTP Simulation Results 64

© Copyright IBM Corp. 1997

x Load-Balancing Internet Servers

Preface

Written for systems administrators, Webmasters, and others, this redbook provides a comprehensive overview of IBM's Interactive Network Dispatcher product, the definitive product of its type for creating balanced Internet server environments.

A variety of configurations and environments, including World Wide Web, File Transfer Protocol and high-availability scenarios, are presented as are all the concepts involved in providing a highly efficient, cost-effective solution.

Some practical examples are presented to demonstrate how to design and configure an environment with Interactive Network Dispatcher.

The examples provided throughout this document are not supported by IBM, but are provided as a guide to allow you to gain from the experience and knowledge of the people who wrote this redbook. Without a doubt, sharing their experiences with Interactive Network Dispatcher will shorten the time it takes you to implement your own load-balancing solution.

How This Redbook is Organized

The chapters in this redbook are designed to provide a guided approach to understanding the concepts involved in, and functions provided by, Interactive Network Dispatcher; the book is organized as follows:

- Chapter 1, "The Big Picture" on page 1

  Introduces the Internet Server environment and explains the need for large-capacity, scalable servers and load balancing

- Chapter 2, "Interactive Network Dispatcher Concepts" on page 9

  Explains what the IBM Interactive Network Dispatcher is and how it works and introduces most of the concepts that will be explored in the following chapters

- Chapter 3, "Interactive Network Dispatcher in Detail" on page 17

  Explores Interactive Network Dispatcher, explaining the parameters that can be used to customize the configuration files to make them appropriate for your own needs

- Chapter 4, "Installation and Operation" on page 39

  Contains a brief explanation on how to plan, install and operate the Interactive Network Dispatcher product, presenting a few guidelines based

© Copyright IBM Corp. 1997

on the information obtained from the *Interactive Network Dispatcher User's Guide*

- Chapter 5, "WWW Server Scenario" on page 47

  Describes the specific use of Interactive Network Dispatcher to balance the load of World Wide Web (WWW) servers

- Chapter 6, "FTP Server Scenario" on page 61

  Describes the specific use of Interactive Network Dispatcher to balance the load of an File Transfer Protocol (FTP) server

- Chapter 7, "Other Server Scenarios" on page 71

  Summarizes the use of Interactive Network Dispatcher to balance the loads of some specific Internet scenarios

- Chapter 8, "High Availability Server Scenario" on page 75

  Discusses options and considerations for increasing the availability of services using Interactive Network Dispatcher

- Chapter 9, "Two-Tier Network Scenario" on page 87

  Discusses the Two-Tier configuration option that allows load-balancing across a collection of servers. This is most commonly used to serve geographically dispersed groups of users and servers

- Chapter 10, "Configuration Files - Samples" on page 93

  Presents the configuration files and the parameters used for the tests performed

- Appendix A, "Minimal Configuration" on page 111

  Shows the minimal configuration needed to perform a minimal test or to demonstrate Interactive Network Dispatcher in a restricted environment, with very few machines

- Appendix B, "Domain Name System (DNS)" on page 115

  Covers the relevant concepts for configuring a DNS server, concentrating particularly on the requirements for delegating domains to other name servers

- Appendix C, "Hints and Tips" on page 127

  Presents the hints and tips that were gained from our experiences writing this book

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Luis Ferreira is an Advisory System Engineer at the International Technical Support Organization, Austin Center. Before joining ITSO in mid-1997, Luis worked for IBM Brazil as an IBM Certified I/T Specialist in the Open System Center providing pre- and post-sales support for AIX and Internet customers. Luis has been working with the UNIX environment since 1983 as a software developer. In 1990, he joined IBM to work for the Field System Center team supporting AIX, system performance, high availability, and system administration.

Cameron Ferstat is the Webmaster and Systems Architect for the IBM Olympic Internet Team in New York. Before joining this group in mid-1997, Cameron worked at the International Technical Support Organization, Austin Center, writing redbooks and teaching IBM classes worldwide on AIX in general and on Internet servers in particular. Before joining the ITSO, Cameron worked for IBM Australia providing pre- and post-sales support to AIX customers. Cameron has been working with AIX and the RS/6000 for over eight years and running WWW servers for four years.

Laura Castia is an AIX System Engineer at IBM Italia, NC Competence Center, Cagliari. She holds an electrical engineering degree from the University of Cagliari, Italy. Laura has been working with AIX, the Internet, Intranet, and RS/6000 for two years.

Soo-Young Lee is a Network Specialist in Seoul, where she provides support and services for Internet customers on AIX and Windows NT. Soo-Young has worked at the Network Solution Service in IBM Korea for one year.

Marcus Vinicius Itala Ferreira is an AIX Product Systems Specialist at IBM Brazil, where he provides post-sales services for customers in AIX, Internet and RS/6000 SP areas. Marcus has three and a half years of experience in the AIX field and two and a half years of experience in the Internet field.

Gerardo Vega is an AIX and network computing I/T Specialist at IBM Mexico, where he provides pre- and post-sales support to customers. He has seven years of experience in AIX and three years in the Internet field. He has worked at IBM for seven years. He has written extensively on AIX, DCE and Internet Web servers.

Thanks to the following people for their invaluable contributions to this project:

Earl Mathis
Chris Gage
George McDaniel
John Balogh
Steve Roma
Steve Fonts
IBM Research Triangle Park, North Carolina

Curtis Hoskins
Crystal Rothaupt
IBM Poughkeepsie, New York, U.S.A.

Martin Murhammer
International Technical Support Organization, Raleigh Center.

Andreas Hoetzel
Heinz Johner
International Technical Support Organization, Austin Center

Marcus Beattie
IBM Hursley Park, Winchester, U.K.

Marcus Brewer, Editor
International Technical Support Organization, Austin Center

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in the ITSO Redbook Evaluation to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

  For Internet users `http://www.redbooks.ibm.com/`
  For IBM Intranet users `http://w3.itso.ibm.com/`

- Send us a note at the following address:

  `redbook@US.ibm.com`

Chapter 1. The Big Picture

In the business world of the 1990s, the computer environment most often means a network environment. More and more computers are involved with integration and communication. The Internet model, structure and protocol are here to help us implement these needs.The Internet is the world's largest network, and it has become essential in organizations such as government, academia and commercial enterprises. Transactions over the Internet are becoming more common, especially in the commercial arena.

The information that organizations have on their traditional or legacy business applications may now be published and accessible to a wide audience. This access may include a person checking a bank savings account, making a hotel reservation or buying tickets for a concert. Making this information or service available for their customers is a competitive advantage for any organization. However, regardless of the innovation and potential benefits provided by a company's Internet solution, its value is greatly reduced if the information cannot be accessed in a reasonable response time.

The Internet model allows the distribution of services among different servers, called Internet Servers. It is not necessary to tie an application to one specific server. Instead, the service belongs to a group of servers; so an additional computer can be added or removed when necessary. By grouping the set of servers in a single entity, a big issue comes up—*load balancing*.

Optimum performance is achieved by balancing the load of these servers, and the IBM Interactive Network Dispatcher software is the best solution for addressing load-balancing issues.

1.1 Internet and Intranet Sizing

When a server is established on the Internet, one of the most difficult tasks is to predict the number of accesses it will receive. For some businesses, putting a site on the Internet is like opening a new branch office in every city around the world. Although the load on the site will probably start at a low level, it can grow at unpredictable rates. If a Web site is chosen as a "pick of the day" or "cool site of the day," the hits on a server can increase from hundreds to tens of thousands overnight.

In addition, Internet-style services such as the World Wide Web and the Usenet news servers are increasingly being utilized within companies on their private corporate intranets. This is due to the ease and speed with which

© Copyright IBM Corp. 1997

these services can be implemented and integrated with existing computer systems to provide very efficient intra-company communications and new internal computer systems accessed through a common (Web browser) interface. In fact, the use of such services on intranets is growing up to ten times faster than Internet implementations.

It is easier to size a server for an intranet because the potential number of users can be determined more accurately. Since the client and servers are normally connected to the same networks, the speed of the connection is usually not a problem. This means can be provided richer content (higher quality graphics, multimedia, and so on) in the services and information offered. But the richer the content, the more load it puts on the server; so there is still a strong requirement for scalability in intranet servers.

1.2 Sizing for Growth

The load on an Internet site is unlikely to remain constant. The number of accesses on a Web server or FTP server can increase for several reasons.

1. Most companies add their Web site's address to television, radio and print advertising and to product catalogues and brochures. As these "Web-aware" publications circulate and replace the previous "URL-free" versions, awareness of the Web site grows.
2. As time passes, the Web site gains better coverage in the on-line search engines such as Yahoo or Alta Vista.
3. Assuming the site is providing useful information or a useful service to customers, repeat visitors should increase.
4. Most Web sites begin simply, with fairly modest content—mostly text, with some images. As the site designers grow in confidence, more resources are allocated, and as Web users in general increase their modem speeds, most sites move towards richer content. Thus, not only do hit rates increase, but the average data transfer per hit also rises.
5. Most sites begin as presence sites—providing corporate visibility on the Internet and making information about the company available to potential customers. Most present sites use predominantly static HTML pages. Static pages are generated in advance and stored on disk. The server simply reads the page from the disk and sends it to the browser. However, many companies are now moving towards integration applications that allow users of the Web site to directly access information from the company's existing applications. This could include checking the availability of products, querying bank account balances or searching problem databases. These applications require actual processing on the

server system to dynamically generate the Web page. This dramatically increases the processing power required in the server.

1.3 Dealing with the Growth

There are several ways to deal with the growth of your Internet site:

1. Purchase an initial system that is much too large.

   This is one way to deal with Web site growth; however, most companies are not willing to invest large sums of money in a system that is much larger than they require—particularly since the benefits that they will gain from the site have yet to be proven. Most prefer to purchase a minimal initial system and to upgrade in the future as the site demonstrates its worth to the company.

2. Replace the server with a larger system.

   This is certainly a possible solution. The RS/6000 suits this solution better than many other platforms since binary compatibility is maintained across the entire range. The same operating system and applications can be run, and the same management skills will apply on the replacement system. However, this method is ultimately limited by the size of the largest system manufactured, which may not be large enough for some Web sites.

3. Purchase an upgradable SMP system.

   Purchasing a symmetric multiprocessing (SMP) system provides a low-disruption upgrade path for your Web server. However, there is a limitation to the amount of scaling that can be achieved. Most SMP systems can be upgraded to at most eight times their original performance level. This may not be sufficient alone, but may well be used in conjunction with the final method.

4. Perform load balancing between multiple servers.

   In this case, the load for the overall site is balanced between multiple servers. This allows scaling beyond the maximum performance available from a single system and allows for easy upgrading by simply installing additional servers and reconfiguring the cluster to use the additional servers. This solution can also provide the added benefit of higher server availability. The load-balancing software can automatically allow for the failure of a single server and balance the load between the remaining sites. This is the solution provided by the Interactive Network Dispatcher software.

1.4 Interactive Network Dispatcher Overview

The Interactive Network Dispatcher is a superior load-balancing software that evenly distributes load among mirrored servers by routing TCP/IP session requests to different servers within a group of servers. It increases the performance of servers, and it balances the requests among all the servers by operating transparently to users and other applications. It is useful for applications such as e-mail servers, World Wide Web servers, distributed parallel database queries, and other TCP/IP applications.

Interactive Network Dispatcher enables multiple Web servers to efficiently function as a single system to better manage high volumes of information and electronic transactions over networks. This optimizes Web site performance, maximizes existing hardware investments, simplifies the administration of Web servers and improves availability of Web site resources and end user satisfaction.

Interactive Network Dispatcher (IND) consists of two components:

- Interactive Session Support balances the load on servers within a local or wide area network using a round-robin approach or a user-specified approach, with or without a DNS name server.
- Network Dispatcher balances the load on servers within a local or wide area network using a number of weight and measurements that are dynamically set by Network Dispatcher.

These two components can be used separately or together to implement a powerful, flexible, and scalable load-balancing solution to address the Internet or intranet server needs of the largest enterprise. If visitors to your site can't get through at times of greatest demand, IND can automatically find the optimal server to handle incoming requests, thus enhancing your customers' satisfaction and your profitability.

For more information on Interactive Network Dispatcher, see Chapter 2, “Interactive Network Dispatcher Concepts” on page 9, and Chapter 3, “Interactive Network Dispatcher in Detail” on page 17.

Interactive Network Dispatcher is a member of the e–business enhancer category of the IBM Network Computing Framework. The official site of this product is at the following URL:

`http://www.ics.raleigh.ibm.com/netdispatch/`

1.5 Why Do I Need Interactive Network Dispatcher?

The number of users and networks connected to the global Internet is growing exponentially. This growth is causing problems of scale that can limit users' access to popular sites.

Currently, network administrators are using numerous methods to try to maximize access. Some of these methods allow users to choose a different server at random if an earlier choice is slow or not responding. This approach is cumbersome, annoying and inefficient. Another method is the standard round-robin approach, where the domain name server selects servers sequentially to handle requests. This approach is better, but still inefficient because it blindly routes traffic without any consideration of the server workload. In addition, even if a server fails, requests continue to be routed to it. For more information on approaches to managing server workload, including animated examples, go to the following URL:

```
http://www.ics.raleigh.ibm.com/netdispatch/examples.htm
```

The Interactive Network Dispatcher software offers numerous benefits over earlier and competing solutions:

Scalability

As the number of client requests increases, servers can be added dynamically, providing support for tens of millions of requests per day, on tens or even hundreds of servers.

Efficient use of equipment

Load-balancing ensures that each group of servers makes optimum use of its hardware by minimizing the hot-spots that frequently occur with a standard round-robin method.

Easy integration

Interactive Network Dispatcher uses standard TCP/IP protocols. It can be added to an existing network without making any physical changes to the network. It is simple to install and configure.

Low overhead

Interactive Network Dispatcher needs only to look at the inbound client-to-server flows. It does not need to see the outbound server-to-client flows. This significantly reduces its impact on the application compared with other approaches and can result in improved network performance.

Non-invasive technology

Interactive Network Dispatcher does not modify any packets, nor does it require any modifications to the operating system on which it runs.

High Availability

Interactive Network Dispatcher offers built-in high availability solutions; the primary Network Dispatcher has one or more backups that receive the status of all connections. When the primary Network Dispatcher machine fails, back-up machines take over immediately. See Chapter 8, “High Availability Server Scenario” on page 75.

1.6 Where Interactive Network Dispatcher Is Being Applied

Interactive Network Dispatcher is used on IBM's Web site with resulting throughput that is almost four-times faster than the previously used domain name server approach. Access the site at:

`http://www.ibm.com/`

During the 1996 Summer Olympics in Atlanta, Internet services were handled by Interactive Network Dispatcher, which routed requests among nearly five dozen hosts in five locations around the world. In the 17 days of the Olympics, the site received just over 192 million hits. At peak times, the Olympics site received 17 million hits per day, including real-time audio requests lasting an hour or more, as well as requests for graphic images and other very large files.

Interactive Network Dispatcher is an integral part of the architecture used to serve the official Web site of the upcoming XVIII Olympic Winter Games in Nagano, Japan. Access the site at:

`http://www.nagano.olympic.org/`

The same infrastructure is serving IBM’s Olympic Web site—a site devoted to explaining IBM’s role as a sponsor of the 1998 Winter Olympic Games. Access this site at:

`http://www.ibm.com/Olympic/`

The following high-volume sites have also used Interactive Network Dispatcher to optimize servers usage:

- American School Directory Project

  `http://www.asd.com/`

- Masters Tournament

  `http://www.masters.org/`
- PGA Championship

  `http://www.pga97.com/`
- Roland Garros, The French Open

  `http://www.frenchopen.org/`
- The Australian Open

  `http://www.ausopen.org/`
- The US Open

  `http://www.usopen.org/`
- Wimbledon Championship

  `http://www.wimbledon.org/`

Sizing for the Internet - The ACM Chess Challenge

IBM's chess-playing supercomputer, Deep Blue, challenged World Chess Champion Gary Kasparov to a battle between man and machine. For the duration of the tournament, IBM ran a Web site to provide interactive coverage, including live images and audio, and full commentary.

In the 1996 tournament, the Web site was originally sized for an estimated 200,000 hits, and a uniprocessor Web server was installed. During the first game, the site received 5 million hits! The Web server was unable to handle the load.

With 48 hours before the start of the second game, the IBM Web team installed an RS/6000 SP system using six nodes running as Web servers. The load was balanced among the nodes using a development version of the Interactive Network Dispatcher product discussed in this book.

As the tournament progressed, hit rates continued to climb, and three additional nodes were added to handle the extra load. The IBM Web site ensured that the matches could be watched by millions of people all over the planet.

1.7 Basic Terminology

This section explains the basic terminology used in discussing Interactive Network Dispatcher.

1.7.1 Interactive Network Dispatcher

Interactive Network Dispatcher is a scalable, highly-available load-balancing software solution for HTTP, FTP or other TCP-based servers. It balances the load of Internet servers on a variety of operating systems, such as AIX, Windows NT and Sun Solaris.

1.7.2 Hypertext Markup Language (HTML)

HTML is a language for creating the hypermedia documents that are the foundation of the World Wide Web.

1.7.3 Hypertext Transport Protocol (HTTP)

HTTP is a TCP/IP protocol used by World Wide Web servers and Web browsers to transfer hypermedia documents across the Internet.

1.7.4 World Wide Web (WWW) Server

HTTP is a TCP/IP application that allows Web browsers to transfer hypermedia documents across the Internet.

1.7.5 File Transfer Protocol (FTP) Server

FTP is a TCP/IP application that allows a user to send or retrieve files from a remote computer.

1.7.6 Domain Name System (DNS) Server

DNS is a TCP/IP application protocol that provides mapping between IP addresses and names. The names are arranged in hierarchical domains, where the top of hierarchy is the last element of the name (for example, www.ibm.com).

Chapter 2. Interactive Network Dispatcher Concepts

This chapter explains what IBM Interactive Network Dispatcher is and how it works.

2.1 Introduction

As previously seen in Chapter 1, “The Big Picture” on page 1, the World Wide Web’s rapid growth as an information and transaction medium is overwhelming the computing infrastructure. The number of users and networks connected to the global Internet is growing exponentially. This growth is causing scalability problems that can limit a user’s access to popular sites. Network administrators are using numerous methods to try to maximize these accesses. As explained, there are several methods to improve the service capability, but the most powerful and flexible solution is the Interactive Network Dispatcher implementation.

Interactive Network Dispatcher is a server load-balancing software that boosts the performance of servers by routing TCP/IP session requests to different servers, thereby (within a group of servers properly configured) balancing the clients’ requests among all servers. This routing is transparent to users and other applications such as e-mail servers, World Wide Web servers, distributed parallel database queries, and other TCP/IP applications.

2.2 Interactive Network Dispatcher Functions

Interactive Network Dispatcher consists of two functions that can be used separately or together to provide superior load-balancing results.

2.2.1 The Network Dispatcher Function

The Network Dispatcher function balances the load on servers within a local area network by using a number of weights and measurements that are dynamically set by Network Dispatcher. This function provides load balancing at a level of specific services, such as HTTP, FTP, SSL, NNTP, POP3, SMTP, and Telnet.

2.2.2 The Interactive Session Support (ISS) Function

The Interactive Session Support (ISS) function balances the load on servers within a local or wide area network by using an intelligent round-robin approach or a more advanced user-specified approach. Load-balancing is performed at the machine level.

© Copyright IBM Corp. 1997

ISS can also be used to provide server load information to a Network Dispatcher machine.

When used for load-balancing, ISS works in conjunction with the Domain Name System (DNS) server to map DNS names of ISS services to IP addresses. When ISS is used to provide server load information, a name server is not required.

Using them together

Using the Interactive Session Support (ISS) and the Network Dispatcher functions together, it is possible to balance the load on servers within both local and remote networks.

2.2.3 How Network Dispatcher Works

Network Dispatcher creates the illusion of having just one server by grouping systems together into a cluster that behaves as a single, virtual server. The service provided is no longer tied to a specific server system; so you can add or remove systems from the cluster, or shutdown systems for maintenance, while maintaining continuous service for your clients. The balanced traffic among servers seems for the end users to be a single, virtual server. *The site thus appears as a single IP address to the world***.** All requests are sent to the IP address of the Network Dispatcher machine, which decides with each client request which server is the best one to accept requests, according to certain dynamically set weights. Network Dispatcher routes the clients' request to the selected server, and then the server responds directly to the client without any further involvement of Network Dispatcher. Network Dispatcher can also detect a failed server and route traffic around it.

Network Dispatcher for AIX contains a kernel extension that provides routing of Transport Control Protocol (TCP) and User Datagram Protocol (UDP) traffic requests. The Network Dispatcher receives the packets sent to the cluster. These packets have a source and a destination address—the destination address is the IP address of the cluster. All servers in the cluster and in the dispatcher system have their own IP address and an alias for the IP address of the cluster. The dispatcher system checks which server is the least busy and routes the packet to that server. Since all servers in the cluster have an alias for the cluster's IP address, Network Dispatcher routes this request based on the hardware address of the network adapter (MAC address) of the chosen server. It changes the hardware address of the packet to the hardware address of the selected server and sends the packet to the server. The server receives the packet and responds directly to the client.

This makes it possible to have a small bandwidth network for incoming traffic (like Ethernet or token ring) and a large bandwidth network for outgoing traffic (like ATM - Asynchronous Transfer Mode or FDDI - Fiber Distributed Data Interface).

The server machines in the cluster could be a mixture of heterogeneous servers of different sizes and types, such as UNIX, Windows NT or OS/2 machines.

Network Dispatcher consists of three components:

executor This component supports port-based routing of TCP and UDP connections to servers based on the type of request received (for example HTTP, FTP or SSL). This module is an AIX kernel extension and always runs when the Network Dispatcher function is being used.

manager The Manager sets weights used by the Executor based on internal counters in the Executor itself and feedback from the Advisors and ISS monitoring (if ISS is used as a monitoring tool). Each input given to the Manager by the Advisors, ISS and Executor factor has a relative proportion of importance; so you can give more importance to one input over the others, or totally ignore one or more inputs. Using the Manager is optional, but if the Manager is not used, load balancing is performed using weighted, round-robin scheduling based on the current server weights.

advisors Advisors send requests to the back-end servers to measure actual client response time for a particular protocol. These results are then fed to the Manager to adjust the load-balancing weights. Currently, there are Advisors available for HTTP, FTP, SSL, SMTP, NNTP, POP3, and Telnet. Using the Advisors is optional, too.

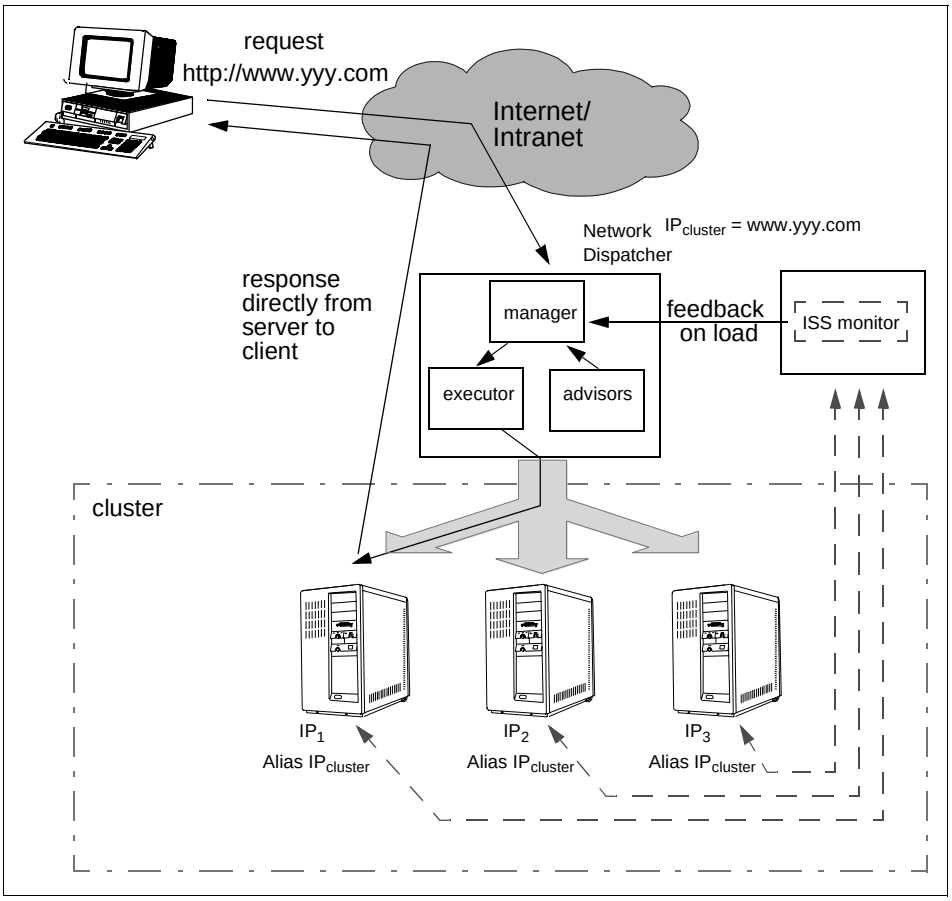


Figure 1. Network Dispatcher with ISS Providing Servers Load Information

2.2.4 How Interactive Session Support (ISS) Works

As said above, the ISS function can be used to provide server load information to a Network Dispatcher machine or to provide load-balancing by itself.

- If ISS is used to collect server load information, there is an ISS monitor that collects this information from the ISS agents that must run on the individual servers. The ISS monitor then forwards this information to the Network Dispatcher that uses this load information, along with other sources of information, to perform load balancing.
- If used for load balancing, ISS works in conjunction with a Domain Name System. This may be either an actual DNS name server or—if you set up a

small, separate subdomain for a name server—a replacement name server provided by ISS. The DNS name server daemon and the `issd` daemon running in the monitor mode can reside in different nodes. Every ISS service has its own DNS name defined in the DNS configuration files. A client submits a request referencing that DNS name. Then ISS resolves that name to the IP address of the selected server of the cell and routes this IP address to the client.

ISS periodically monitors the level of activity on a group of servers and detects which server is the least heavily loaded. The administrator must decide the criteria to be used to measure the load on the servers, depending on the environment being monitored. After every monitoring period, ISS checks the DNS configuration files to make sure that the mapping between the DNS name of the ISS service and the corresponding IP address matches the IP address of the selected server. It can also detect a failed server and route traffic around it.

The previous release of ISS introduced the concept of a *pool*, which was defined as a group of machines performing the same function. An ISS *cell* is defined as one that contains groups of servers, and each group performs one function called a *service*, which is similar to a *pool*. A server—or *node*—can run a single service (HTTP, FTP or Telnet) or many services. Moreover, *pools* of servers running the same protocol can differentiate each other for the criteria chosen to balance the load among the servers of the *service*.

For end users, a *service* is a single computational resource representing a single virtual server, and ISS balances the workload across the various systems in the service to provide optimum productivity.

The *service* is a group of servers that are equally suitable for a particular purpose. A service is identified by a hostname. A user that wants to access a service uses this hostname to do so. All servers providing one service must be indistinguishable from the user's point of view. This means they must offer the same data to the user, and if applicable, the user must be able to log in using a consistent user name and password. This can be accomplished using a distributed file system such as AFS, NFS/NIS or DFS/DCE or by using a shared disk architecture. ISS allows you to have multiple services for one cell, and a server can be a member of a number of different services, as shown in Figure 2 on page 14.

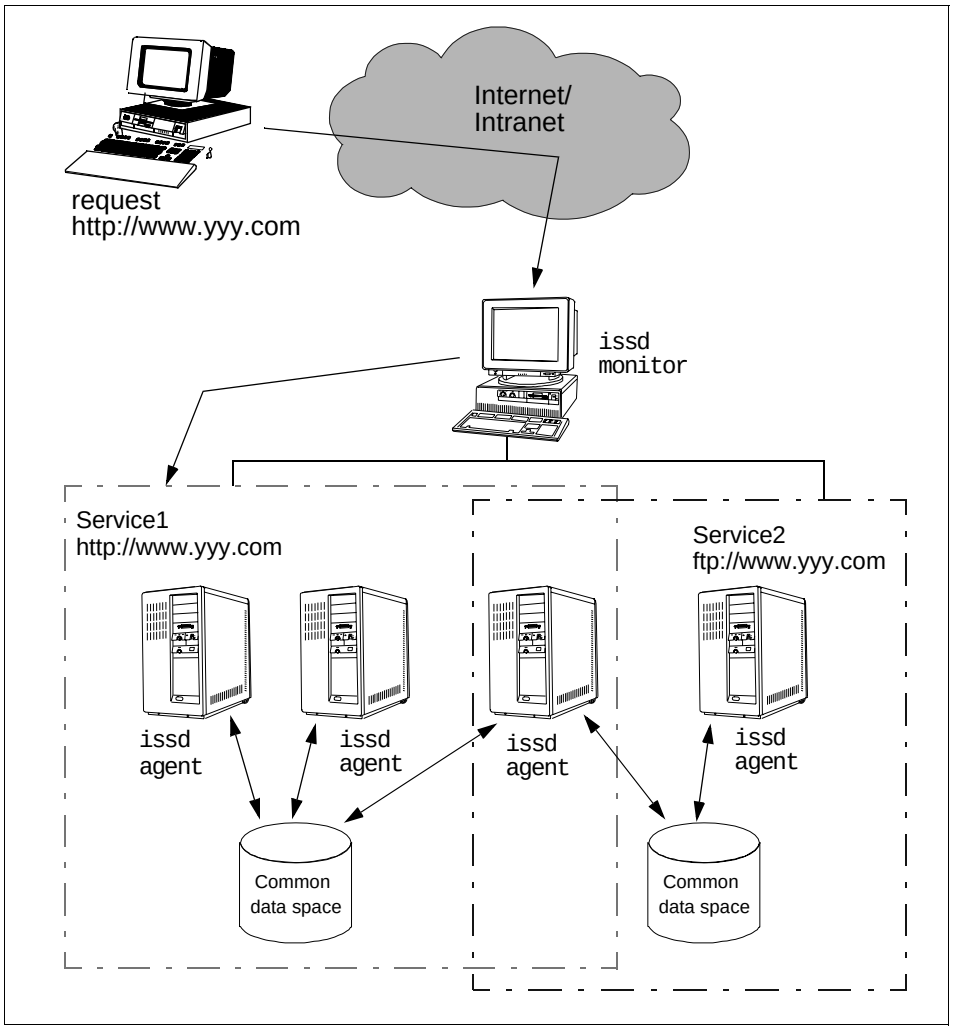


Figure 2. Multiple Services Appear as One Virtual Server

One of the main advantages of ISS is that it offers finer granularity for load balancing than many other solutions. Interactive sessions can be balanced according to the site requirements.

ISS can be used for load balancing, as the figure above shows, or to collect server load information.

In the first case, the ISS works in conjunction with a TCP/IP domain name server to manage the workload. When a user sends a request to a service in

one ISS cell, it must first resolve the hostname of the service to an IP address. This name resolution is performed either by the DNS `named` daemon or by the ISS software, and the IP address of the most appropriate server is returned.

In the second case, ISS works to provide load server information to a Network Dispatcher machine, which does not affect name resolution. This solution gives greater flexibility in the possible configurations that can be achieved, especially in environments such as large Web sites, which are characterized by a large number of clients submitting a high volume of interactive requests.

Chapter 3. Interactive Network Dispatcher in Detail

This chapter explores Interactive Network Dispatcher and explains the parameters that can be used to customize the configuration files to make them appropriate for your own needs. There are many parameters that can help you to create the most appropriate configuration to guarantee an accurate distribution of the load over your servers.

The Network Dispatcher functions are examined first; then we look at the Interactive Session Support (ISS) functions.

3.1 The Network Dispatcher Function in Detail

This section focuses on the Network Dispatcher function and its components.

3.1.1 Interaction between Network Dispatcher Components

As explained previously, Network Dispatcher consists of three components: Executor, Manager and Advisors—the last two are optional. First, let's see how they interact.

The typical flow of information used in Network Dispatcher is shown in Figure 3.

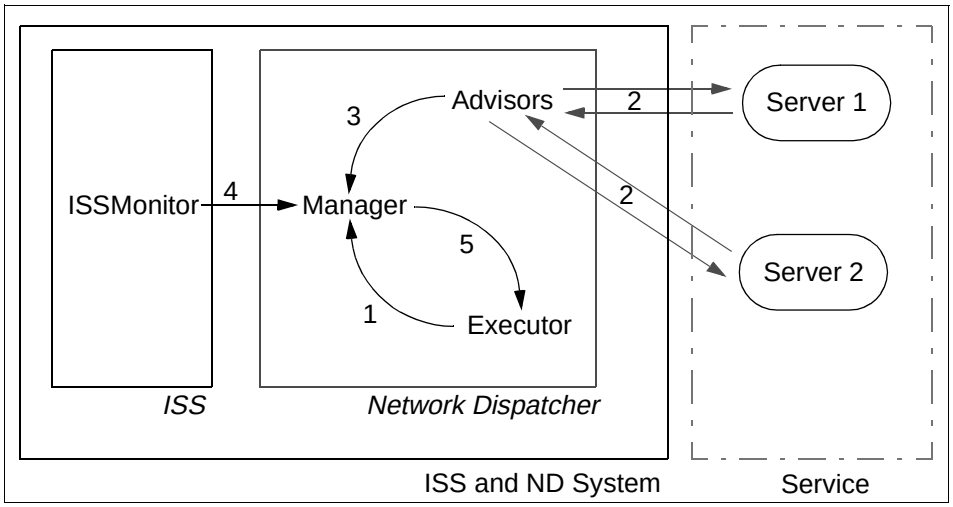


Figure 3. Typical Flow of Information in Network Dispatcher

© Copyright IBM Corp. 1997

In Figure 3, we are assuming that Network Dispatcher is using Advisors and ISS to collect servers' load information, and therefore, the Manager is running to receive this information.

As you can see, Advisors are a key component of Network Dispatcher. They are responsible for monitoring each back-end server, determining its availability and response time. After the information is gathered and processed, it is forwarded to the Manager to be used as one factor for the calculation of the least-loaded server.

Each Advisor is specific for a service. Interactive Network Dispatcher 1.2 for AIX comes with seven Advisors: HTTP, FTP, SSL, SMTP, NNTP, POP3, and Telnet.

The Executor supplies information about new and active connections based on its internal counters to the Manager (1). The Advisors collect information about response time and availability from the servers for each service and each port (2). After processing this information, they send it to the Manager (3). If active, the ISS component also sends information about the servers' load according to a specific metric to the Manager (4). Then, the Manager uses all the information it receives to calculate the new weights to be used in connection routing and sends the results to the Executor (5), which uses these new weights for TCP and UDP routing. If the Manager and Advisors are not running, the Executor does the routing based on its internal counters.

A summary of the functions of each component is found in Table 1.

Table 1. Summary of Functions for Network Dispatcher Components

Component	Function	Sends Information to:
Executor	Routes TCP and UDP requests based on weights. Monitors number of active, new and finished connections. Performs garbage collection of RESET connections.	Manager
Manager	Collects information from internal counters of the Executor, ISS Dispatcher Observer and sent by Advisors. Performs load balancing decisions based on collected information. Updates weights used by the Executor.	Executor

Component	Function	Sends Information to:
Advisors	Gather information about availability and response time for each service in each port for servers.	Manager
ISS component (if present) set up as a Dispatcher Observer	Collects information about server load based on specific metrics.	Manager

3.1.2 Controlling Network Dispatcher Components

Network Dispatcher handles requests from the command line to the Executor, Managers and Advisors. The first step that must be done, after installing and configuring the Network Dispatcher, is to run it using the following command:

```
ndserver start
```

A command line interface is provided by `ndcontrol` to configure and manage the Executor, Advisors and Manager.

The Executor is the main component; it controls a set of clusters. You can set the maximum number of clusters, the maximum number of ports per cluster, the maximum number of servers per port, and so on.

For example, to set the maximum number of clusters, type:

```
ndcontrol executor set maxcluster 4096
```

to start and stop the Executor, type:

```
ndcontrol executor [ start | stop ]
```

To configure the Advisors, the `ndcontrol` command line interface must be used again.

For example, to start and stop an Advisor, use the following command:

`ndcontrol advisor [start | stop]` *service port*

where *service* indicates the name of the service to be monitored (HTTP, SSL, FTP, SMTP, POP3, NNTP, or Telnet), and *port* indicates the port number to be monitored for that service. Advisors can monitor several port numbers for the

same service name—it is necessary to issue just one `ndcontrol advisor start` command for each port to be serviced.

Observation: To start an Advisor, it is necessary for the Manager to be already running. This can be done using:

```
ndcontrol manager start
```

From time to time, each Advisor asks for a status from the servers on the monitored ports and sends the results to the Manager. This interval of time is configurable. Its default value is seven seconds. To change this value, use the following command:

`ndcontrol advisor interval` *service port interval_time*

The *`interval_time`* must not be smaller than the Manager interval time because if it is, some information delivered by the Advisor to the Manager will not be used. This value must be carefully chosen because the Advisors actually make a request for information to the servers to measure the time it takes for the request to be completed.

In order to avoid the use of out-of-date information supplied by the Advisor, the Manager will not use any information that is older than the time set in the Advisor time-out. The Advisor time-out should be greater than the Advisor pooling time to prevent the information from being discarded before being used by the Manager. To set the time-out period for an Advisor, type:

`ndcontrol advisor timeout` *service port timeout_period*

where *`timeout_period`* is either a number of seconds for the time-out period or the word `unlimited`, which means that the information supplied by the Advisor does not expire. The default value is unlimited. If you set up a specific time-out and an Advisor sends server failure information to the Manager, Network Dispatcher will use this information even if the Advisor information has timed out.

When started, Advisors create a log file to keep a log of their actions. There is a log file for each Advisor—that is, for each service and each port number. The name for the log file is: /usr/lpp/ind/nd/logs/advisor_service_port.log. For example, for an HTTP Advisor using port 8080, the file name would be /usr/lpp/ind/nd/logs/http_8080.log. If there is a file with the same name of the log file, it will be overwritten. The name of the log file is set when the Advisor is started. To use a non-default log file name, the following command must be used to start the Advisor:

`ndcontrol advisor start` *service port logfilename*

where *`logfilename`* is only the name of the new log file. To change the directory where the logfile is located, the /usr/bin/ndserver file must be edited.

The maximum size of the log file and a log level can also be set. The maximum size of the log file sets the greatest size the log file may grow to. If the file size reaches this size, the next entries of the log are written on the top of the file (in a round-robin fashion). The log level says how much information is saved on the log file. It is a number from 0 to 5, where 0 means only errors are logged, and 5 means everything is logged for debug purposes. To change the loglevel:

`ndcontrol advisor loglevel` *service port level*

where *`level`* is the log level desired. The default log level is 1.

To change the maximum log file size:

`ndcontrol advisor logsize` *service port logsize*

where *`logsize`* is either the maximum size of the log file in bytes or the word `unlimited`, which means that there is no maximum log file size. The default is unlimited.

When using Advisors and/or ISS, there is another important parameter that can be changed, the `smoothing` value, which decides how smooth the change of the servers' weights will be. The default is 1.5, but a higher value means that the Manager will change the weights less drastically limiting the amount that a server's weight can change. To change the value enter:

`ndcontrol manager smoothing value`.

For more information on all the possible options of the `ndcontrol` command, see "Interactive Network Dispatcher User's Guide" on page 39.

3.1.3 Proportions

As was shown in "Interaction between Network Dispatcher Components" on page 17, the factors that are used in the calculation of the least loaded server in a pool are:

- Number of active connections
- Number of new connections
- Information supplied by Advisors

- Information on load supplied by system monitoring tools, such as the ISS component

The way to establish how important each of the factors are in the load-balancing process—that is, how the Manager uses the information given to it —is by using *proportions of importance*. Each of these factors is attributed a number, from 0 to 100, that can been seen as a percentage. Zero means that the factor is not used. One hundred means that the factor is the only one used. The sum of all proportions must always be 100.

To set the proportions, use the following command:

`ndcontrol manager proportions` *active new advisor system*

where *`active`* means the proportion of weight to be given to the active connections, *`new`* to the new connections, *`advisor`* to the information provided by the Advisors, and *`system`* is the proportion of weight to be given to the information provided by system-monitoring tools such as ISS. The sum of *active+new+advisor+system* must always be 100. The default proportions are 50 50 0 0.

To set all the factors to be equally important, set the proportions to 25 25 25 25.

Since Advisors provide accurate information on how loaded a specific server is for one service and one port, it is suggested that its proportion be set to a number different than zero.

When there is no Advisor for a specific server (Telnet, for example), the proportion given to the Advisor input is ignored. So, for example, if the proportions are set to 33 33 34 0 for a Telnet port, the proportions that will be effective are 50 50 0 0.

3.1.4 Ports Used by Network Dispatcher

Network Dispatcher needs to use some ports to perform its tasks. By default, it uses certain port numbers. Sometimes, it is necessary to use port numbers other than the default (as for example when the Network Dispatcher server is an RS/6000 SP node).

If you need to change the port used by `ndserver`, which is 10003 by default, you have to modify the ND_PORT value in the usr/bin/ndserver and /usr/bin/ndcontrol files.

Moreover, the default port used by the Manager to receive load information by ISS is 10004. To change it, you have to enter the following command in the Network Dispatcher machine when you start the Manager:

`ndcontrol manager start` *logfile metric_port*

If you specify a *`metric_port`*, you must specify a *`logfile`* name.

Moreover, when you define the Dispatcher Observer in the ISS configuration file, you should indicate the port you have decided to use:

`Dispatcher` *hostname metric_port*

3.2 Interactive Session Support (ISS) Function in Detail

This section is about ISS and its concepts.

3.2.1 ISS Cells

An ISS *cell* is a group of nodes administered as a single, logical unit.

A *service* is a group of servers that perform the same function. Each server represents a *node* and can belong to one or more services (HTTP, FTP, Telnet). When configuring ISS and editing the configuration file, you must first declare all the nodes in your site that will be part of the ISS cell; see Chapter 10, “Configuration Files - Samples” on page 93.

When you declare a node, you can use the `Node` or the `Node NotMonitor` keyword. In this release of the Interactive Network Dispatcher, you can configure one node running the `issd` process in monitor mode and one or more others as backups of that node, defining a different priority number for each one. In a backup node, `issd` usually runs in agent mode, and the node can only be a server that provides a service. Only if the monitor node fails will the first backup node in the list take over, and the `issd` run in monitor mode. As soon as the previous node is available again, it will become the monitor because of its higher priority number.

Next, you declare the services that will be provided and their parameters. These parameters include the `NodeList` that comprises the nodes that will provide the service and the `ResourceList` that specifies which resource will be used to select the right server in the cluster.

A resource may be a CPU, a disk, a process, and so forth. A single resource may be used repeatedly; so it is defined in the file after listing the nodes and before the services definitions. The `ResourceType` parameter is used with its

own keywords: `Metric`, `MetricNormalization`, `MetricLimits`, and `Policy`. We will speak more about these parameters later in this chapter.

After configuring nodes and services, you should configure the observers. For a description of this concept, see “ISS Observers” on page 24.

3.2.2 ISS Observers

The ISS Observer uses information about the status of nodes in the cell. It provides three options for load balancing. Depending on the requirements of your particular environment, you may choose one or more of these options. Each option corresponds to one type of ISS Observer. Their names are the same as the keywords you must use to define in the ISS configuration file:

```
NameServer

ISSNameserver

Dispatcher
```

You can use one or more ISS Observers in the same cell.

3.2.2.1 NameServer

When using `NameServer` type, the ISS Observer works in conjunction with a DNS name server. ISS calls the name server to map DNS names of ISS services to IP addresses of the most appropriate server. The `issd` running in monitor mode can run in a different machine from the `named` daemon.

In this way, you can configure one or more servers of your cell as backup `issd` monitor nodes. You can define a different priority number for every node of the cell you decide to use as a backup `issd` node. When the first one fails, the next on the priority list takes over. Until then, the `issd` runs only in agent mode.

ISS monitors the load on each server of the cell and ensures that the server currently used for a particular service is the one with the lightest load. Of course, the criteria of selecting the best server for each service depends on how you have configured ISS and which `SelectionMethod` keyword you have decided to use.

There are two possible selection methods: `RoundRobin` and `Best`. The second one chooses the best server of a cluster depending on the metric you have previously defined in the `ResourceType` sections of the ISS configuration file.

To use this method, you must first configure a DNS server or have one already configured. You can add the servers to a domain that already exists or create a new subdomain. ISS will then update the files that DNS uses for name resolution and reverse name resolution. Figure 4 on page 25 shows ISS operating with a `NameServer` Observer. In this figure, the `issd` monitor works as a `NameServer` Observer, too. If ISS is configured to use the `Best SelectionMethod`, each `issd` running as agent provides server load information to the `issd` monitor.

The `NameServer` Observer involves some load on the DNS server machine. Every time a new server is selected as the top ranked server, the DNS configuration files are updated. A signal is then sent to the `named` daemon to reload the name resolution data files. If these files are large, it can take a significant amount of time and processing for the `named` daemon to process them. During this processing, the `named` is unable to respond to name resolution requests.

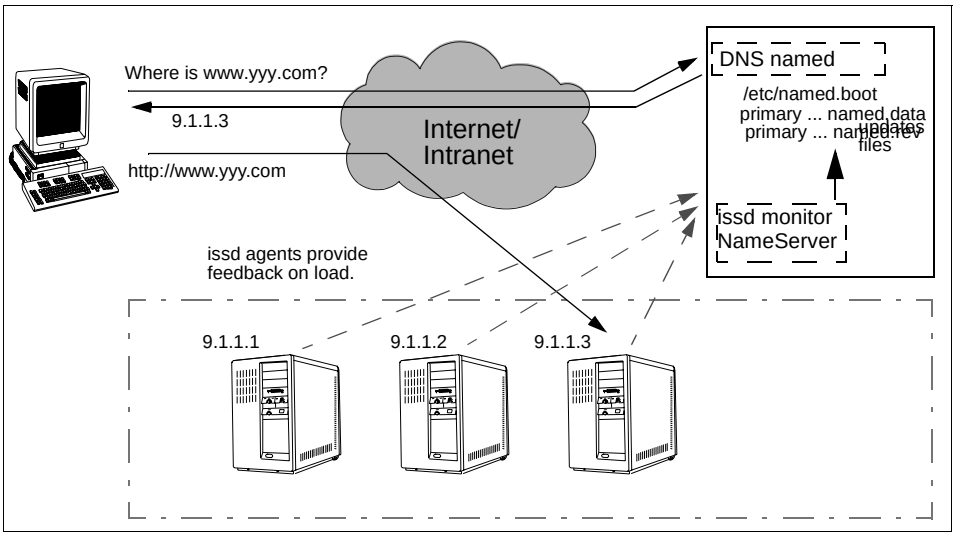


Figure 4. ISS Configured with a NameServer Observer

3.2.2.2 ISSNameserver

When using `ISSNameserver` type, the ISS Observer works as a DNS name server. In this case, ISS uses the DNS configuration files of the DNS name server, but the `named` daemon is not running. This method allows ISS to provide balancing on a per-request basis without the need to continually refresh the DNS `named`. The ISS software replaces the `named` daemon and

performs the name serving function by providing minimal name server implementation.

Since using `NameServer` Observer involves some load on the DNS server machine, a good option is to use `ISSNameserver` Observer with a new subdomain for your pool of servers.

The `ISSNameserver` Observer will provide forward resolution (from name to IP address), but not reverse resolution (from IP address to name). As for the `NameServer` Observer, you can define one or more machine to be an `issd` backup for the one that is currently running in monitor mode.

The selection methods available to configure load balancing are the same as with the `NameServer` Observer: `RoundRobin` and `Best`.

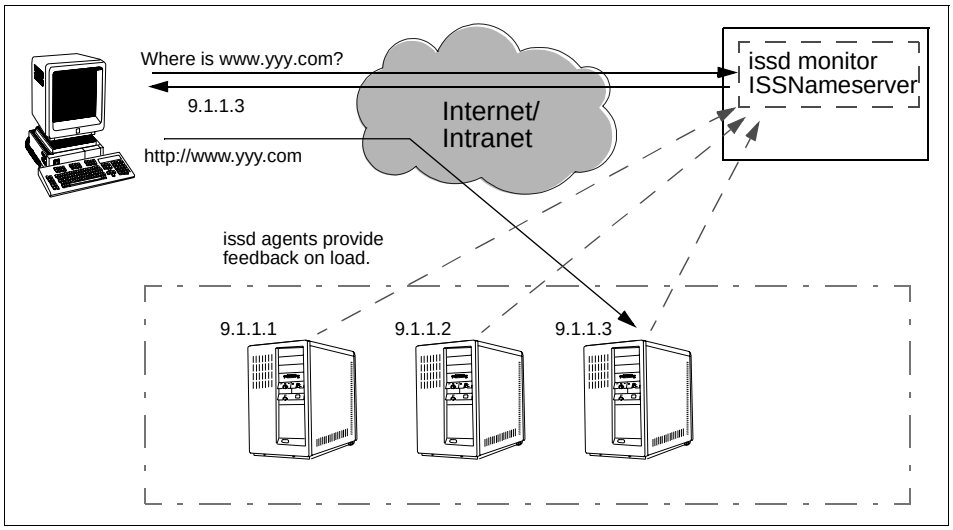


Figure 5. ISS Configured with an ISSNameserver Observer

3.2.2.3 Dispatcher

When using `Dispatcher` type, the ISS Observer works in conjunction with Network Dispatcher to perform the load-balancing function. Neither a DNS name server nor an ISS Observer working as DNS is required.

The previous approaches (`NameServer` and `ISSNameserver`) are the simplest to use to balance the workload, but they have their disadvantages. Most applications and browsers do caching of hostnames to efficiently handle subsequent requests. This may lead to unbalanced requests to the servers. `Dispatcher` Observer can be used to solve this problem giving more

granularity when routing requests to a server. `Issd` running in monitor mode is only used to provide load feedback from the `issd` agents running on each server to the Network Dispatcher. Only Network Dispatcher provides load balancing. Network Dispatcher allows you to link many individual servers into large, virtual servers. This configuration is called a *cluster*. All requests are sent to the single IP address of the Network Dispatcher, which then routes the requests to one of the servers. Naturally, Network Dispatcher uses the load information from ISS along with all its other sources of information.

You can have one or more machines defined as backup `issd` in monitor mode.

As with previous Observers, you can choose between the `RoundRobin` and `Best` selection methods.

3.2.3 ISS Selection Methods

Each service maintains a ranking of the best nodes to perform that service. You can set a particular selection method using the `SelectionMethod` keyword in the ISS configuration file. This keyword may have one of the following values:

- `RoundRobin`

  The load among servers is distributed on a round-robin basis. For example, if you have three servers providing a service (A, B and C), the first request will be sent to server A; the second will be sent to B, and the third to C. The fourth request will again be sent to A, the fifth to B, and so on. In this method, the load on each of the servers is ignored. The round-robin metric can be used for services in which all requests are similar in frequency and duration. The main advantage of this method is that the servers do not have any monitoring overhead. The main disadvantage is that this metric does not notice when the servers are unevenly loaded. But a server will not be recommended if it appears to be down. Read more about round-robin configuration in “RoundRobin” on page 32.

- `Best`

  The `issd` monitor calculates the best server for every defined service once each update interval based on the type of measurement you have decided to use to measure the load on the servers. The load metric is defined using the `ResourceType` keyword. See “ISS Configuration File” on page 28, for how to define your load metrics.

For every provided service you have, you define how to measure the load on the servers. Groups of servers providing the same service can be balanced

with different metrics. For example, A round-robin service can be configured and additionally, can provides a number of services that use a custom configuration.

3.3 ISS Configuration File

The ISS configuration file is used to specify all options for configuring the ISS component. When you start the `issd`, give the name of the configuration file as the first parameter. Use the following command as the root user to start ISS:

```
issd -c filename
```

Sample configuration files are provided in the directory /usr/lpp/ind/iss.

The ISS configuration file is an ASCII file containing a series of definitions. Each definition must be at the start of a line and must start with a keyword which is followed by possible further keywords and then a value. In this section, we discuss some of these configuration keywords. For a detailed description of the ISS configuration file, and all possible keywords, refer to the manual described on “Interactive Network Dispatcher User’s Guide” on page 39.

When you configure the ISS file, you must first define the cell and its attributes, then list the nodes that will be part of the cell, the resource types of the cell with their parameters, the services to be provided, and the Observers you will work with.

3.3.1 Cell and Its Attributes

These options control the overall behavior of the ISS monitor and should be set once per configuration file.The `cell` keyword specifies the symbolic name of the ISS cell. It must appear at the top of the file. You must specify the name of the cell. You can have *local* or *global* cells. You need to define one and only one local cell per ISS configuration file. A local cell is the one that the node will be a member of. You define a global cell as a separate, remote cell that you want your node to communicate with to implement *ping triangulation*. For more related information about ping triangulation, see the manual described in “Interactive Network Dispatcher User’s Guide” on page 39.

The syntax is:

`Cell` *Name* [`Global` | `Local`]

Then you have to specify the following cell's attributes:

- **Authkey**

  This key is used to provide an authentication during the ISS internal traffic among the nodes of the cell. If this keyword is not specified in the configuration file, a default key will be used. A node can have its own `Authkey`, which has to be specified after the node is defined.

- **LogLevel**

  You can have five different loglevels, each one providing more information than the previous one:

  [`None` | `Error` | `Info` | `Trace` | `Debug`]

- **PortNumber**

  This option specifies the port used for ISS internal traffic. Use the syntax:

  ```
  PortNumber [number]
  ```

- **HeartbeatInterval**

  ISS will periodically contact all servers that provide a specific service to ensure that they are still "alive". ISS does this using the equivalent of the `ping` command. This does not verify that the service is running on the servers, but at least ensures that the servers can be contacted at the IP level. ISS maintains a ranking that lists the servers in their current priority order according to the configured load measurement metric. If the currently top-ranked server fails to respond, the next server will be selected. This check is referred to as a heartbeat. The time between heartbeats is determined by the `HeartbeatInterval` setting. The syntax is:

  ```
  HeartbeatInterval [seconds]
  ```

  The default is 10 seconds.

- **HeartbeatsPerUpdate**

  The `HeartbeatsPerUpdate` value is multiplied by the value of the `HeartbeatInterval` to give a time called the update interval. The update interval is used to determine:

  - How frequently the load measurement metric will be checked to determine if the ranking has changed
  - How frequently the `named` daemon will be updated when the `NameServer` Observer is running
  - How frequently load profiles will be sent to Network Dispatcher if the `Dispatcher` Observer is running

  The syntax of this keyword is:

`HeartbeatsPerUpdate count`

Where `count` is a non-negative integer.

The update interval is calculated as:

Update interval = `HeartbeatInterval` * `HeartbeatsPerUpdate`

Note that the `HeartbeatsPerUpdate` will be ignored in the situation where the top ranked server fails to respond to a heartbeat. The server will immediately be removed from its top ranked position. If `NameServer` Observer has been configured, the name server will be updated immediately.

3.3.2 Node

After defining the cell, you must list all the nodes that will be part of it using the `Node` keyword. The syntax is:

`Node` *[hostname | IP address] priority*

The *priority* field is a positive integer that can mean two things. In fact, in this release of ISS, you can decide to have a node running the `issd` in monitor mode, but you can set up other nodes that will take over if the monitor fails. The priority number establishes the order that will be followed in taking over the monitor. If you use only the `Node` keyword followed by the name of the server, you are declaring that machine can run as `an issd` monitor based on its priority number. If you don't want that machine to run as an `issd` monitor, you have to declare this using the `NotMonitor` keyword. The syntax is:

`Node` *[hostname | IP address] priority* `NotMonitor`

In this case the priority field is only a numeric identifier for that node.

There is another keyword referred to as a specific node, that is the `InternalNet` keyword. If your machine has more than one network adapter and you want to define an internal network for ISS network traffic among the nodes of the cell, you can do it using this keyword. The syntax is:

`Node` *[hostname1 | IP address1] priority* `InternalNet` *[hostname2 | IP address2*]

ISS will use the `IP address2` to make the `issd` communicate with each other, but it will use the `IP address1` to provide the services to the clients.

Node definition

The hostname of an AIX machine must be set up. It can be done by using smitty. ISS needs to recognize this hostname or the corresponding IP address. If your machine has more than one network interface (each one with its own DNS name and corresponding IP address) and you do not want to use the one corresponding to your main hostname to provide the service, you should either change your hostname to the one corresponding to the network interface you want to use or maybe define an internal net.

We worked with SP nodes with their hostnames corresponding to the Ethernet interfaces in the DNS configuration. But we used the token-ring interfaces to provide ISS services. Therefore, to make ISS recognize the machine in which it was running we needed to configure an Ethernet internal network.

3.3.3 ResourceType

The `ResourceType` keyword must be followed by the symbolic name of the resource you are defining. It specifies the criteria you have decided to use for selecting the Best server in a service. You can have many different resource types in the ISS configuration file and decide to use one or more of them for each service depending on your own needs. The syntax is:

```
ResourceTypes name1 name2
```

Every resource type is characterized by several parameters that define the metric that will be used to measure the load among the servers. The metric keywords are `Policy`, `MetricNormalization` and `MetricLimits`.

- **Metric**

  The keyword `Metric` in the configuration file specifies the type of measurement ISS will use to provide a specific service. The syntax is:

  ```
  Metric [ Internal | External ] string
  ```

  where the `string` field indicates a command or a program that gives a numeric output. We'll speak more about metrics later in this chapter.

3.3.4 Service

The `Service` keyword must be followed by the symbolic name of the service you want to provide and the service DNS name if you are only using ISS. The cluster IP address and the port number are also used if you are using ISS in conjunction with Network Dispatcher. The syntax is:

```
Service name DNS name [ cluster address ] [ port number ]
```

After defining the service, you need to use the `NodeList` keyword, which must be followed by the hostnames or IP addresses of the servers that will provide the service. Then you declare which previously defined `ResourceTypes` ISS will use for selecting the top-ranking server by listing these `ResourceTypes` after the `ResourceList` keyword.

For each service, you can specify an `Overflow` node that will take over the service if none of the servers in the `NodeList` is available. The syntax is:

```
Overflow  DNS name.
```

3.3.5 Observers

This keyword defines which type of Observer you are using: `NameServer`, `ISSNameserver` and/or `Dispatcher`.

3.4 Metrics

ISS balances the workload over a group of servers providing one or more services. It allows the system administrator to choose one or more measurement metrics for each service. A metric defines how ISS will measure the load on the servers for each service. This measurement should be appropriate to the particular service. For example, if the service provided is CPU-intensive, the metric could be defined as the percentage of time that the CPU is busy. If the service is disk- or I/O-intensive, the metric could be based on the amount of time that the disks are busy or the time spent waiting for I/O to complete.

3.4.1 RoundRobin

When the `SelectionMethod` is set to `RoundRobin`, ISS does not measure the load on the servers that belong to a certain service. It simply cycles through the available servers in the order they appear on the keyword `NodeList`. The update interval indicates the period of time during which a server will get all incoming requests before switching to the next server on the list. This is the simplest measurement type available. Its main advantage is that it does not put any overhead on the servers since they do not have to provide any feedback on load.

This measurement algorithm can be used with any ISS Observer: `NameServer`, `ISSNameserver` or `Dispatcher`. Figure 6 shows this configuration. In this

example, each server will rotate every 20 seconds, and the ISS will check the availability of the servers every 10 seconds.

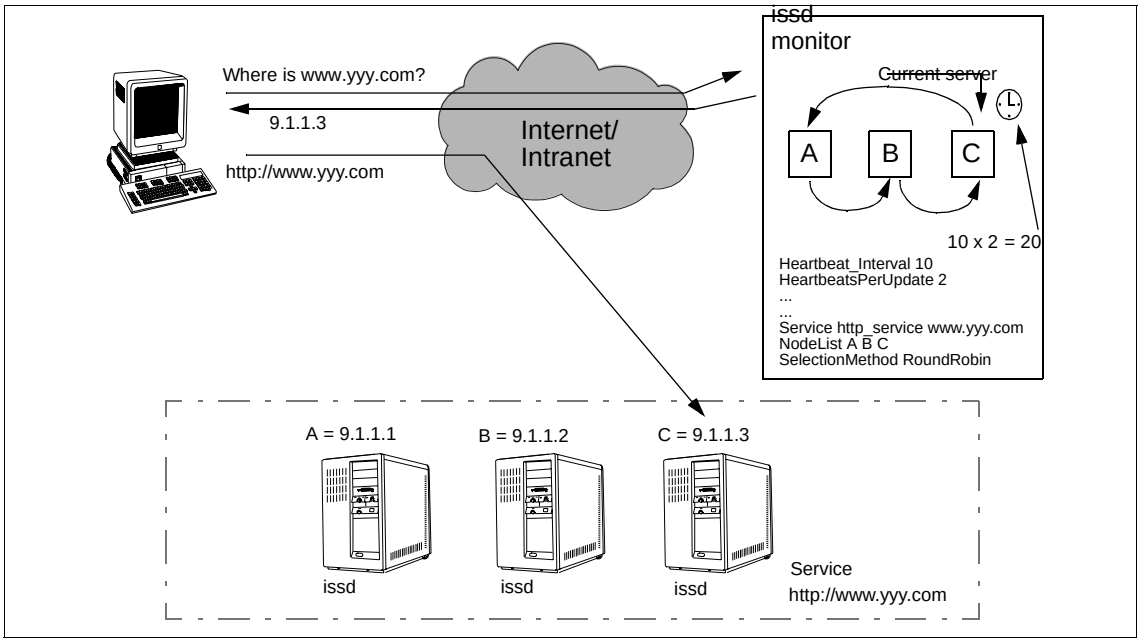


Figure 6. ISS working with a RoundRobin SelectionMethod

This metric would be useful when the access pattern of your clients to your service follows an uniform profile in terms of frequency and duration of access. In this configuration, ISS will never notice if a particular server is getting very busy and will continue sending requests to this server. This may lead to an unbalanced load on your servers.

Be aware that even if the `RoundRobin SelectionMethod` does not calculate any load on the servers, the `issd` must run on every node of the service.

Round-Robin Configuration

To make a round-robin configuration work, you should define in the ISS file at least one `ResourceType` and include it in the `Service` that will use the `RoundRobin SelectionMethod`. The `issd` monitor will not use that `ResourceType`, but it needs it to completely identify the `Service`. For example:

```
Service www_service  www.yyy.com
NodeList  server1 server2  server3
ResourceList CPULoad
SelectionMethod  RoundRobin
```

3.4.2 Internal Metric

The syntax of this option in the configuration file is:

`Metric Internal` *string*

If you specify the `Internal` metric, you can use a method provided directly by the system and the *string* field may be one of the following:

- `CpuLoad`, which measures the percentage of CPU is being utilized
- `FreeMem`, which returns the amount of free physical memory as a percentage

  `LoadLeveler`, which must work in conjunction with an existing and running LoadLeveler configuration

  LoadLeveler is a job-management system that allows users to run more jobs in less time by matching their processing needs to available resources. Loadleveler schedules and manages jobs that you submit to one or more machines under its control. It accepts the jobs and reviews the job requirements and compares them with the resources of the machines under its control to determine which is the best machine to run the job. For more related information about LoadLeveler and how to make ISS work with it, see:

  Using and Administering LoadLeveler, SC23-3989

  Also see the User's Guide described in "Interactive Network Dispatcher User's Guide" on page 39.

The `SelectionMethod` is `Best`.

3.4.3 External Metric

When you decide to use an `External` metric, you define how to measure the load on your servers. This metric is anything that can be executed on the servers and returns a numeric value as a result.

The syntax is:

```
Metric External  string
```

The `string` field defines the command or program that when executed returns a measure of the load on the server. For example, you can run a metric based on the number of processes running on the server by using the following entry:

```
Metric External  ps -ef | wc -l
```

Using a TCP/IP connection, the `issd` running in monitor mode communicates with the `issd` running in agent mode on each server. The port used for this connection is defined by the `PortNumber` keyword at the top of the configuration file. The `issd` agent will run the command provided by the `issd` running in monitor mode and will report back the result and any change in the load.

The `Policy` keyword indicates whether the metric function specified is to minimized or maximized—a high value indicates high load or low load. The syntax is:

```
Policy  [ Max | Min ]
```

This keyword has to be set to correspond with the function you use. In our previous example, we were counting the number of processes; so we had to favor the smallest values, and the policy had to be set to `Min`. But if you use something like:

```
Metric External vmstat 1 2 | awk ' NR == 5 { print $16 } '
```

which returns the percentage of CPU idle time on the machine, you must use a `Max` policy.

The `MetricLimits` keyword defines the limits for the server to continue to provide the service. There is a maximum value that defines the level beyond which the metric should not go, and a minimum value that defines the level down to which the metric should be taken before the server is returned to ranking. In the previous example, based on the number of processes running on the machines, you can decide that a server should provide the service

until there are not more than 250 processes running. If there are more than 250 processes, it will be excluded from the service until the number of the processes goes down to less than 200.

You can obtain this with the syntax:

```
MetricLimits 200 250.
```

Be aware that in this case we have the `Policy` keyword set to `Min`. There will be an inversion in the meaning of the values if the `Policy` has been set to `Max`.

Figure 7 shows an example of ISS running an `External` metric to provide a service. It has been defined as an overflow node, too.

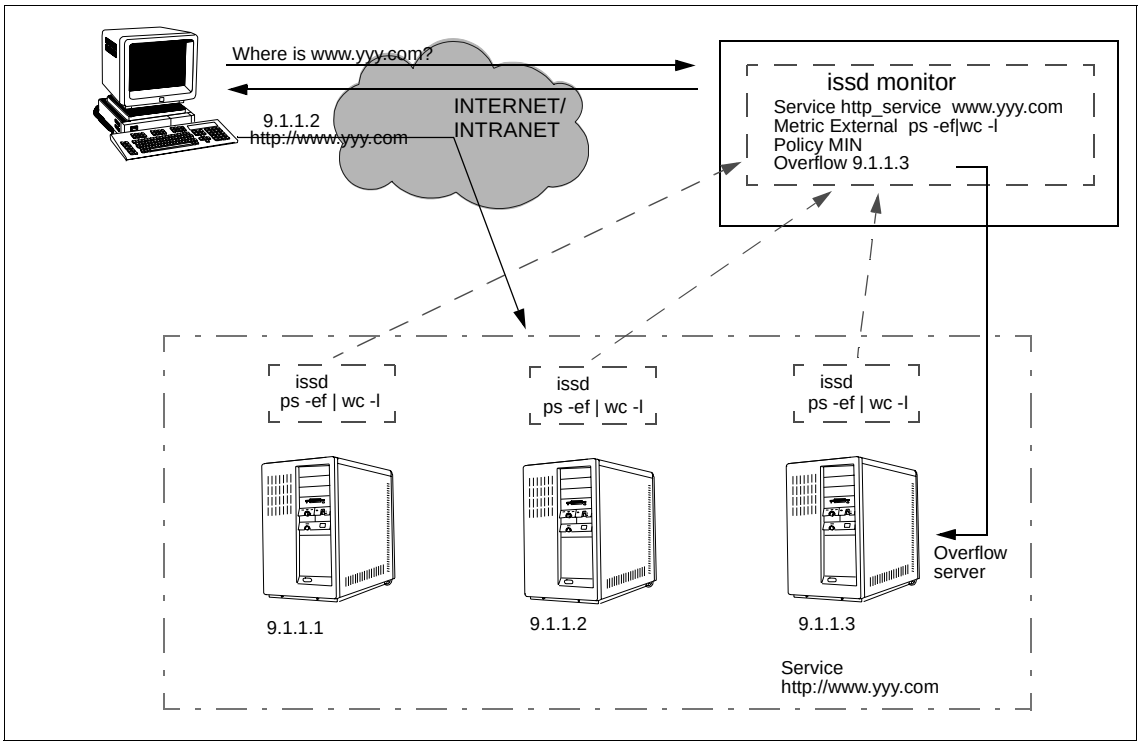


Figure 7. ISS Using an External Metric

3.4.4 Writing Your Own Metrics

The metrics you use with the `Metric External` option can be a mix of different parameters on the servers. You can make the measurement of the load on the server as complex as you need. However, if the metric function you are

planning to use is complex, it is recommended that you to encode it in a shell script or an executable program and make it available to all the servers of the service.

The resources most frequently used to determine the load on the servers are CPU, memory, paging space, I/O, and network usage. To measure CPU, you can use the output of commands like `vmstat` or `sar`. When using `vmstat`, the first line of the report contains statistics for the time since system start-up. Subsequent lines contain statistics collected during the interval since the previous report. For example, you can use the following metric to obtain the percentage of CPU usage:

```
Metric External vmstat 1 2 | awk ' NR == 5 { print $14 + $15 } '
```

This will add the 14 and 15 parameters of `vmstat` which are the CPU percentage for system and user.

Some suggestions of what can be used to obtain custom metric values, either using command line/shell scripts or using executable programs, are shown in Table 2.

Table 2. Possible Sources for External Metric Values

Item to be measured	Command Line	Program

CPU usage	`vmstat`
	`sar`
	`ps`
		special device /dev/kmem
		PTX API

Memory Usage	`vmstat`
	`sar`
	`svmon`
		PTX API

Disk activity	`iostat` `sar`	PTX API
Network traffic	`netstat`	PTX API
NFS performance	`nfsstat`	PTX API

Chapter 4. Installation and Operation

For your convenience, this chapter presents condensed guidelines based on the information found in the manual mentioned in 4.1, “Interactive Network Dispatcher User’s Guide” on page 39.

4.1 Interactive Network Dispatcher User’s Guide

The *Interactive Network Dispatcher User’s Guide*, GC31-8496, explains how to plan, install, configure, use, and troubleshoot the IBM Interactive Network Dispatcher for AIX, Microsoft Windows NT and Sun Solaris.

The most current version of this book is available in HTML and PDF formats on the Interactive Network Dispatcher Website. To access the on-line book, go to the following URL:

`http://www.ics.raleigh.ibm.com/netdispatch/downloads.htm`

4.2 Installing for AIX Installing for AIX

Before you install Interactive Network Dispatcher, you should have a design of the network as well the installation plan. The *Interactive Network Dispatcher User’s Guide* gives a broad idea on how to design your network based on your existing environment.

The hardware and software requirements are:

- A IBM RS/6000-based machine
- IBM AIX Version 4.1, modification level 5 or later, or AIX Version 4.2, modification level 1 or later
- Approximately 15 MB of available disk space for installation
- A CD-ROM drive if installing from CD-ROM

There are two options for installing this product. One is having your own CD, and the other is downloading from the Internet (if it is available). If you are downloading an evaluation copy of the product from the Web site, use the installation instructions at the Web site, whose URL is:.

`http://www.ics.raleigh.ibm.com/netdispatch/downloads.htm`

© Copyright IBM Corp. 1997

Interactive Network Dispatcher consists of two components to be installed.

- intnd.iss contains the ISS component.
- intnd.nd contains the Network Dispatcher component.

After receiving the images of Interactive Network Dispatcher, log in as *root*. Both, the command line or SMIT can be used. The following sequence presents the installation performed by SMIT.

1. Enter:

   ```
   smitty install_latest
   ```

2. From the install Software at latest Level screen:
   - If you purchased the product and have it on distribution media, press **F4** for the input device/ directory for software option and select the appropriate device.
   - If you downloaded the product, enter the name of the directory in which you extracted the packages.
3. From the Install Software Products at latest Level updated screen, press **F4** for the Software to install option.
4. When the product is successfully installed, you will see `Command: OK` in the COMMAND STATUS screen.
5. Press **F10** to exit SMIT.

Now you need to verify whether the product is installed on your machine by using the command below.

```
lslpp -h | grep intnd
```

If you have installed the entire product, this command returns the following:

intnd.iss.rte

intnd.msg.en_US.iss

intnd.msg.en_US.nd

intnd.nd.rte

intnd.ps.en_US

4.3 Configuring Network Dispatcher

This section explains briefly how to configure the Network Dispatcher function. For more conceptual information, see Chapter 2, “Interactive Network Dispatcher Concepts” on page 9, and Chapter 3, “Interactive

Network Dispatcher in Detail" on page 17. Before proceeding with the configuration, make sure the Network Dispatcher machine and all the Internet machines are properly connected to the network.

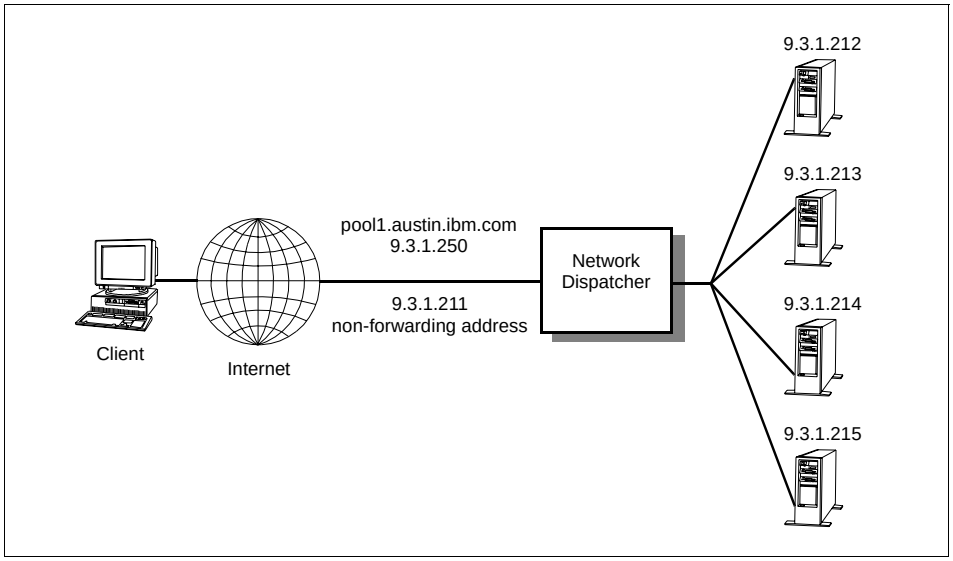


Figure 8. Sample Network with Network Dispatcher and a Group of Servers

Remember

1. You need to set up the network interface every time you start the Network Dispatcher. So before entering the `ndserver` command, check your network interface status by using this command.

   ```
   netstat -ni
   ```

   If you cannot find your network interface alias, you should set up the network interface again.

2. Once you set up your network interface and loopback device on the TCP servers, you can run the same configuration file for the Network Dispatcher machine and TCP servers machines.

The following steps present how to start Network Dispatcher and create the initial configuration.

1. You need to start the Network Dispatcher server.

   ```
   ndserver start
   ```

2. You need to start the Executor.

   ```
   ndcontrol executor start
   ```

3. You need to define the nonforwarding address.

   ```
   ndcontrol executor set nfa IP_address
   ```

4. You need to define a cluster and set cluster options.

   ```
   ndcontrol cluster add cluster
   ```

5. You need to define ports and set port options.

   ```
   ndcontrol port add cluster:port
   ```

6. You need to define TCP server machines.

   ```
   ndcontrol server add cluster:port:server
   ```

7. You may start the Manager component (optional)

   ```
   ndcontrol manager start
   ```

8. You may start the Advisor components (optional)

   `ndcontrol advisor start` *name port*

The Network Dispatcher components (Manager, Executor and Advisors) can be stopped by using `ndcontrol` at the command line.

- Stopping the Manager also stops all the Advisors. To stop it, use:

  ```
  ndcontrol manager stop
  ```

- To stop a specific Advisor (HTTP, FTP, and so forth), use:

  `ndcontrol advisor stop` *name port*

- To stop the Executor, use:

  ```
  ndcontrol executor stop
  ```

To stop the entire Network Dispatcher function, use:

```
ndserver stop
```

4.4 Configuring the Interactive Session Support

This section briefly explains how to configure the ISS function on Interactive Network Dispatcher with a step-by-step sample. Additional information about ISS configuring can be obtained in Chapter 2, “Interactive Network Dispatcher Concepts” on page 9.

Follow this procedure for editing your ISS configuration file:

1. Define each cell:

   Create a cell you want to manage. First, you need to put in the keyword `Cell` and include a name and a destination of `Global` or `Local`.

   Example:

   ```
   Cell              IND             Local
   ```

2. Define your cell attributes. The following attributes may be specified:

   `HeartbeatsPerUpdate 3`

   The name server updates once every three heartbeats.

   `HeartbeatInterval 10`

   Every 10 seconds, the heartbeat is initiated.

   So with these two parameters, you can determine an update interval.

   The update interval is 30 seconds, meaning the name server is updated every 30 seconds.

   `PortNumber 17800`

   You can use any number except well-known port numbers and reserved port numbers.

   `LogLevel Info`

   There are some other choices for this keyword. When you set it to `info`, it provides all error data provided by the `Error` parameter, plus information on significant events.

   `HeartbeatsToNetFail 5`

   After five heartbeats, this node is regarded as unreachable and is not be recommended.

   `HeatbeatsToNodefail 7`

   After seven heartbeats, this node is considered to have failed completely.

   `Authkey`

3. Define individual node data in the cell:

   For example:

   ```
   Node      lf5575a      001
   Node      lf5575b      002
   Node      lf5575c      003
   Node      lf5575d      099
   NotMonitor
   ```

Nodes lf5575a, lf5575b, and lf5575c act as monitors, but node lf5575d does not act as a monitor. The `Node` keyword needs to be specified with the IP address or a DNS name and ID number.

4. Specify the `ResourceTypes` for the cell:

   For example:

   ```
   ResourceType       CPU
   ```

   You can decide your `ResouceType` depending on your system; it can be CPU availability, CPU idle, disk usage, or memory availability. But you need to configure only one service at a time. “ISS Configuration File” on page 28, explains how to define the metric depending on the `ResourceType`.

   Here we configure the metric with CPU availability.

   ```
   Metric                  Internal     CPULoad
   MetricNormalization            0     100
   MetricLimits                  80     95
   Policy                               Min
   ```

5. You need to specify each of the services to be provided:

   For example:

   ```
   Service                 WWWService www.lf5575.austin.ibm.com 80
   ResourceList            CPU
   Nodelist                lf5575a, f5575b, lf5575c
   SelectionMethod         Best
   Overflow                lf5575d.ind.austin.ibm.com
   ```

   The keyword `Service` needs to specify with a name, DNS name and optionally, a service port number. When you use the `RoundRobin` metric, you need to specify `RoundRobin` on the `SelectionMethod` keyword.

6. You need to specify each of the observers:

   ISS has three observers (`ISSNameserver`, `Nameserver`, `Dispatcher`). It depends on design of your servers on your network. The following example is for the `Nameserver`, which changes the DNS name server according to the ISS monitor.

   ```
   NameServer                      f01n01t 53
   ServiceList                     service
   NamedData                       /etc/named.data
   NamedRev                        /etc/named.rev.9.3.1
   ```

For starting ISS, use the following command:

```
issd -c [your configuration file]
```

Run the ISS configuration file on the every machine defined in the cell.

For stopping ISS, use the following command:

```
kill -TERM iss_pid
```

Use the `grep` command to obtain the ISS process ID:

```
ps -ef | grep issd
```

Chapter 5. WWW Server Scenario

The most well-known and widely used service today, the WWW, can be considered the most famous TCP service at present. Each day, millions of people search the World Wide Web for information and commerce. Some sites can even reach 10 million hits per day. This means these sites must have an adequate structure to provide a reasonable response time for people accessing it.

This chapter discusses some tests performed using ISS and Network Dispatcher configured to provide a WWW service.

5.1 Testing Environment Considerations

Each different TCP/IP service demands different resources from the machine, which results in different loads. So it is very difficult to characterize the load for all kinds of service using only one metric. It is necessary to individually study the loads submitted to machines for every service to define individual metrics.

Even for each different service, it is not easy to reach a consensus about what the average load for a machine is. So, a load measurement should be made for each specific situation. Since this is not always possible, people try to study load using some simulations of workload. These simulations are not always precise, but give a reasonable idea of the behavior of the machine, allowing the study of load balancing and scalability. ISS and Network Dispatcher allow you to customize your load-balancing environment.

For the tests, a 5-node RS/6000 SP was used. Four nodes–two wide (nodes 3 and 7) and two thin (nodes 5 and 6)–were the WWW application servers, and one wide node (node 1) was the ISS and Network Dispatcher server. More information about the test configuration files can be found in the Chapter 10, “Configuration Files - Samples” on page 93.

During the tests, some graphics showing network traffic and CPU load in all nodes were collected using Performance Toolbox for AIX (instruments named *f01n01/CPU*, *f01n03/CPU*, *f01n05/CPU*, *f01n06/CPU*, *f01n07/CPU* for CPU load and *f01n01*, *f01n03/TCP*, *f01n05/TCP*, *f01n06/TCP* and *f01n07/TCP* for network load). Some of these graphics are shown in this chapter.

© Copyright IBM Corp. 1997

5.2 Load Generated by WWW Servers

The load generated by WWW servers depends on which types of pages are accessed by clients. There are two types of pages:

- Static pages: These pages are ready at the server, meaning the server only gets the file from local storage (disk, network-shared file systems, and so forth) and sends it to the client. This generates only a small amount of CPU load. In this case, the network is usually the bottleneck. There are some studies that show that medium-sized servers can hold a large number of hits per day with static pages.
- Dynamic pages: These pages are generated by a program that is executed when the request for the page arrives. It can access databases, other network servers and so forth. The amount of CPU load generated by dynamic pages depends on the application that will be executed. Simple applications generate less CPU load than complex applications that access several databases and perform calculations. Usually, sites with a great number of dynamically generated pages have CPU bottlenecks instead of network bottlenecks.

When using only static pages, the main concern is network speed. When using only dynamic pages, the main concern is CPU power. Since the set of pages clients access is usually composed of both types, care should be taken in sizing CPU power and network bandwidth.

5.3 How the Tests Were Performed

The Web server used was the IBM Internet Connection Secure Server Version 4.2. All the files served were local for each machine. They were replicated for all the machines.

To generate the load, two other RS/6000 machines were used. A modified version of the WebStone Version 2.0 benchmark was used. The modification was made because in the standard benchmark, the client program resolves the server name just once at the beginning of the execution, not every time a request for a page is made. A public version of WebStone Version 2.0 benchmark can be downloaded from:

`http://www.sgi.com/Products/WebFORCE/WebStone/`

5.4 The Tests and Their Results

This session describes the tests and shows the PTX graphics that explain the behavior of three different configurations: the traditional round-robin, a balanced distribution using ISS, and a balanced distribution using Network Dispatcher.

5.4.1 Round-Robin Configuration

In the tests using only ISS, we chose the `NameServer` Observer because there was already a name server running on the node selected to run the `issd` in monitor mode. The effect of running the `ISSNameserver` Observer instead of the `NameServer` would be a slightly smaller CPU load due to the nonexistence of the `named` daemon.

Using the `RoundRobin` metric, CPU load and network traffic are ignored. If you have an homogenous and very well-known request profile and the requests are quickly over, you can use the `RoundRobin` metric. But it is not very good if you have random-sized requests (as usually is the case, especially in WWW scenarios). The main advantage of the `RoundRobin` metric is that it is very simple to configure.

In round-robin mode, the load on servers is ignored; each server gets all the requests according to the following rule:

number of the servers in the pool * heartbeat interval * heartbeats per update seconds for heartbeat interval * heartbeats per update seconds

If that interval is not long enough, the server will get all the requests again even if it has not finished serving the previous requests, especially when the load is basically composed of dynamic pages.

Figure 9 on page 50 shows an example of the CPU load obtained by configuring a `NameServer` Observer in a round-robin configuration for dynamic WWW pages. The node *f01n01* was configured to be the ISS and DNS server, and the other ones *(f01n03, f01n05, f01n06 and f01n07)* were configured as WWW servers. You can see that `issd` on node *f01n01* selects every server regardless the real load of the machines. The `HeartbeatInterval` was set to 30 seconds.

The ISS server node is not heavily loaded. You can see a slightly higher CPU load in this machine only when it updates the DNS files. The network bandwidth utilization in this case is also very low.

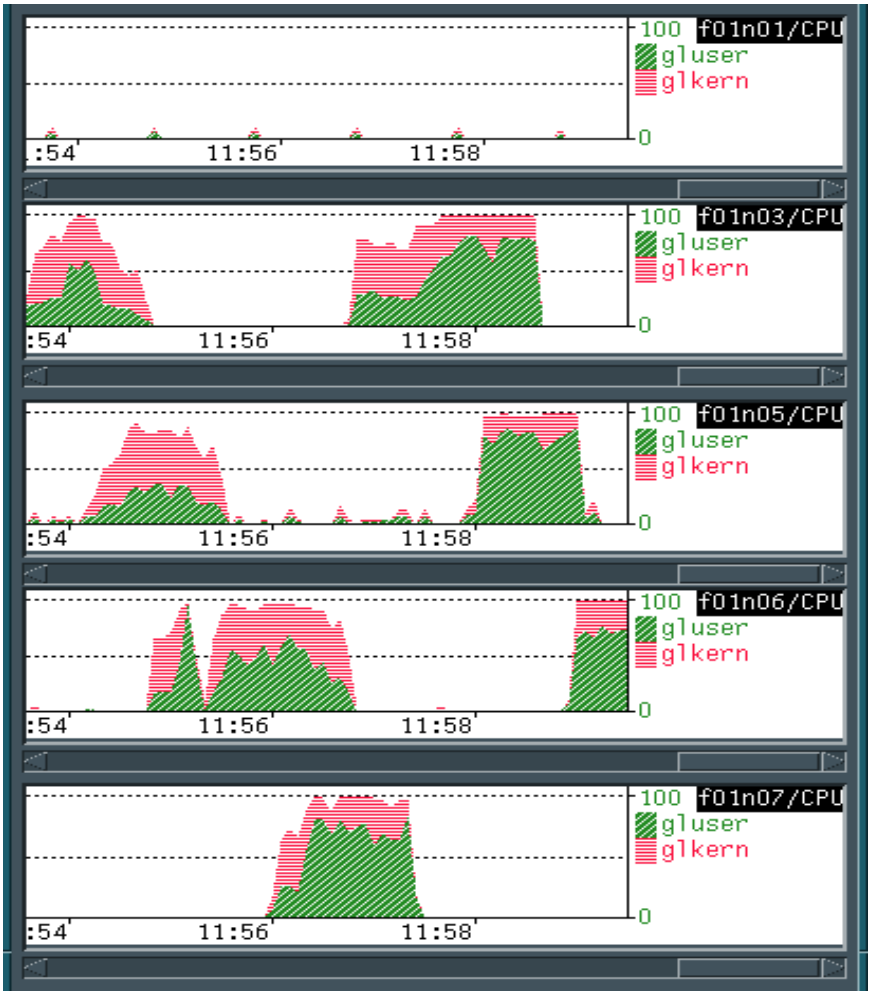


Figure 9. CPU Load - NameServer Observer: Round-Robin for Dynamic Pages

In the case of static pages, CPU utilization is very low, but network traffic is not. Network traffic using static-only pages can be seen in Figure 10 on page 51.

We chose an update interval of 60 seconds, and after every 60 seconds, the selected server changed, but it continued to send TCP packets because there were connections still open.

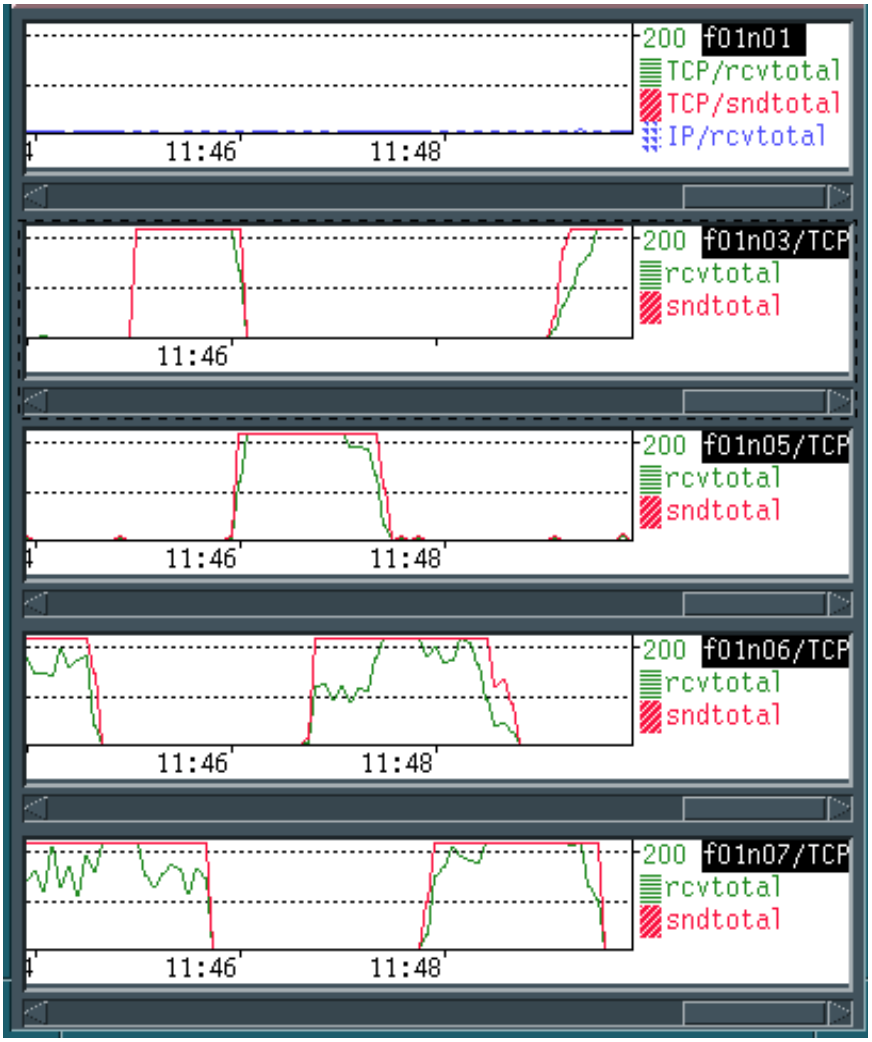


Figure 10. Network Traffic - NameServer Observer: Round-Robin for Static Pages

5.4.2 ISS with Best SelectionMethod

You can configure ISS to work only with a very simple round-robin configuration or to work in a better mode using a `Best SelectionMethod`. In this way, you decide on a specific `ResourceType` that `issd` will use to select the most appropriate server to provide a service. You can decide to balance the load based on CPU load, network traffic, disk usage, and so forth. It depends on your environment and on the service(s) you are providing.

For environments where the greatest part of the access is to dynamic pages, a CPU idle time metric can be used. For static pages, network traffic can be used. For applications that access databases, disk activity time is used.

For example, if you have a Website that provides many dynamic pages, you could use a `ResourceType` that monitors the CPU load:

```
ResourceType External  vmstat 1 2 | tail -1 | awk '{ print $16 } '
```

This `ResourceType` calculates the CPU idle time. The `Policy` must be set to `Max` to inform the `issd` monitor that the least loaded servers will be those with the greatest value for the metric.

You will obtain better results using this metric than a simple round-robin configuration. In fact, when you use a specific `ResourceType`, the `issd` selects the most appropriate server each time, while looking at the real load of each server. When a server is too heavily loaded, it is not chosen by ISS until either its load decreases or every other server is as loaded (or more loaded) as it is. An example of CPU load using dynamic pages and an idle CPU time `ResourceType` is shown in Figure 11 on page 53. Notice that the load on the ISS server (node 1) is very low but higher than in the round-robin configuration because the machine receives the load information packets from the servers and then calculates which one is the best. For the same test, the network traffic is shown in Figure 12 on page 54.

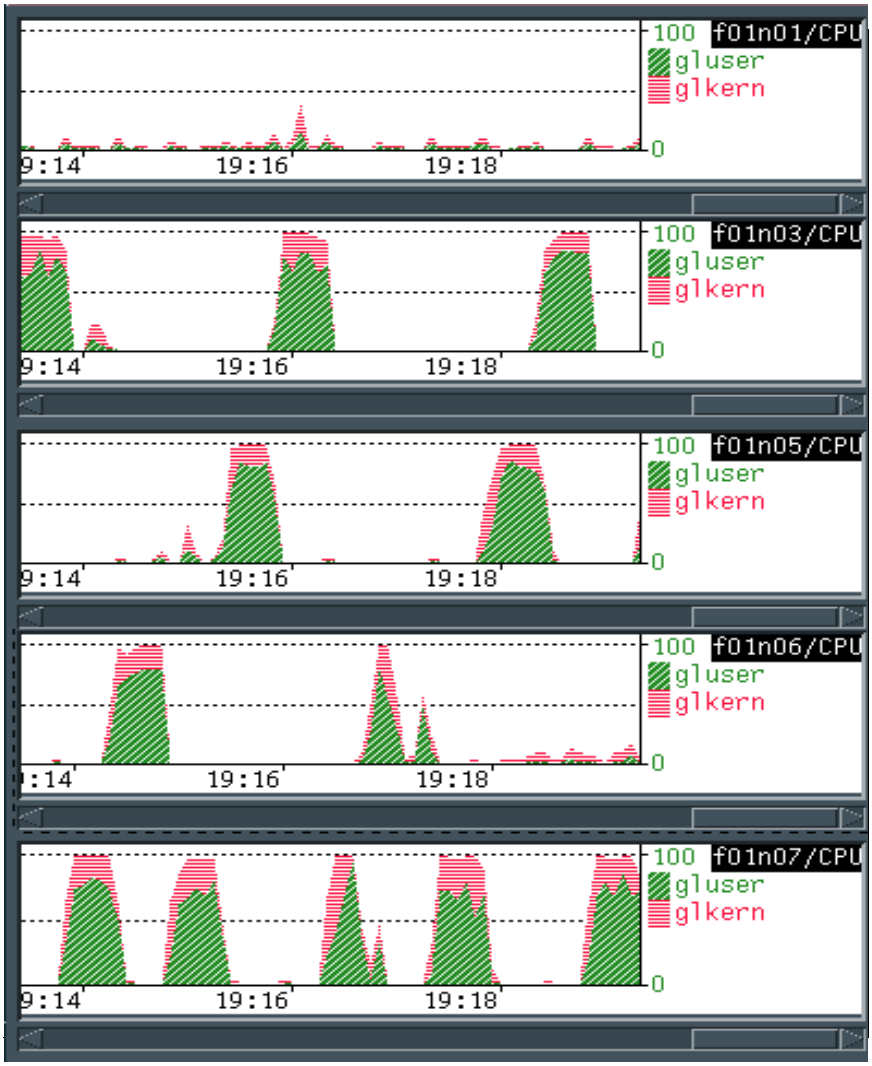


Figure 11. CPU Load - ISS with Idle CPU ResourceType for Dynamic Pages

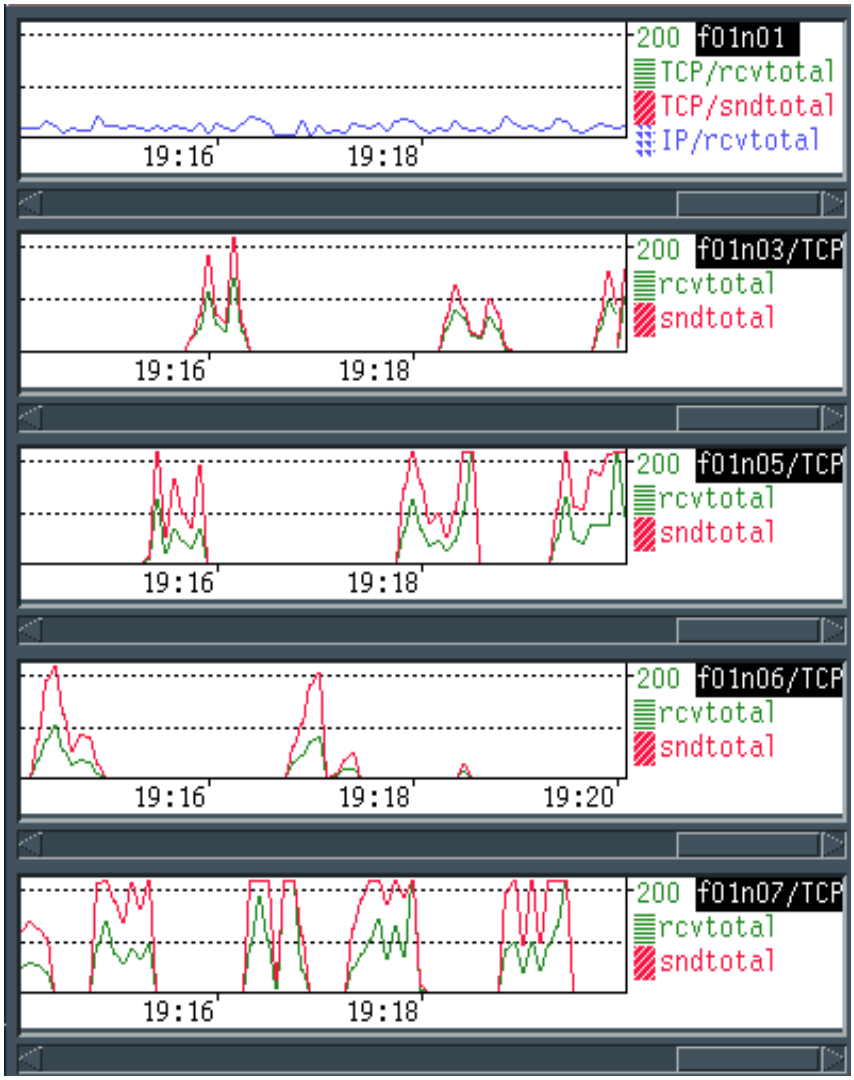


Figure 12. Network Traffic - ISS with Idle CPU ResourceType for Dynamic Pages

5.4.3 Network Dispatcher

To test Network Dispatcher, we used the same environment used for ISS and Node1 (*f01n01*) of SP2 as the Network Dispatcher machine.

Dynamic Pages

Using dynamic pages, all the CPU load graphics were similar. The distribution is smoother than using ISS alone. With Network Dispatcher, the load is better distributed among the servers because of the way Network Dispatcher routes IP connections instead of using DNS to distribute the load.

In Figure 13 on page 56, notice the difference between the configuration that uses only ISS and the one which uses Network Dispatcher. Every server is loaded all the time by simulating a heavier load. In this case, we have a better distribution of all the resources. The Network Dispatcher machine does not work so much in terms of CPU, but there is a lot of IP traffic because every client request goes to this machine before it is routed to the best server.

We set the proportions to 50 50 0 0 and the smoothing value to 1. With a higher smoothing value, there are fewer peaks, and the load distribution is smoother.

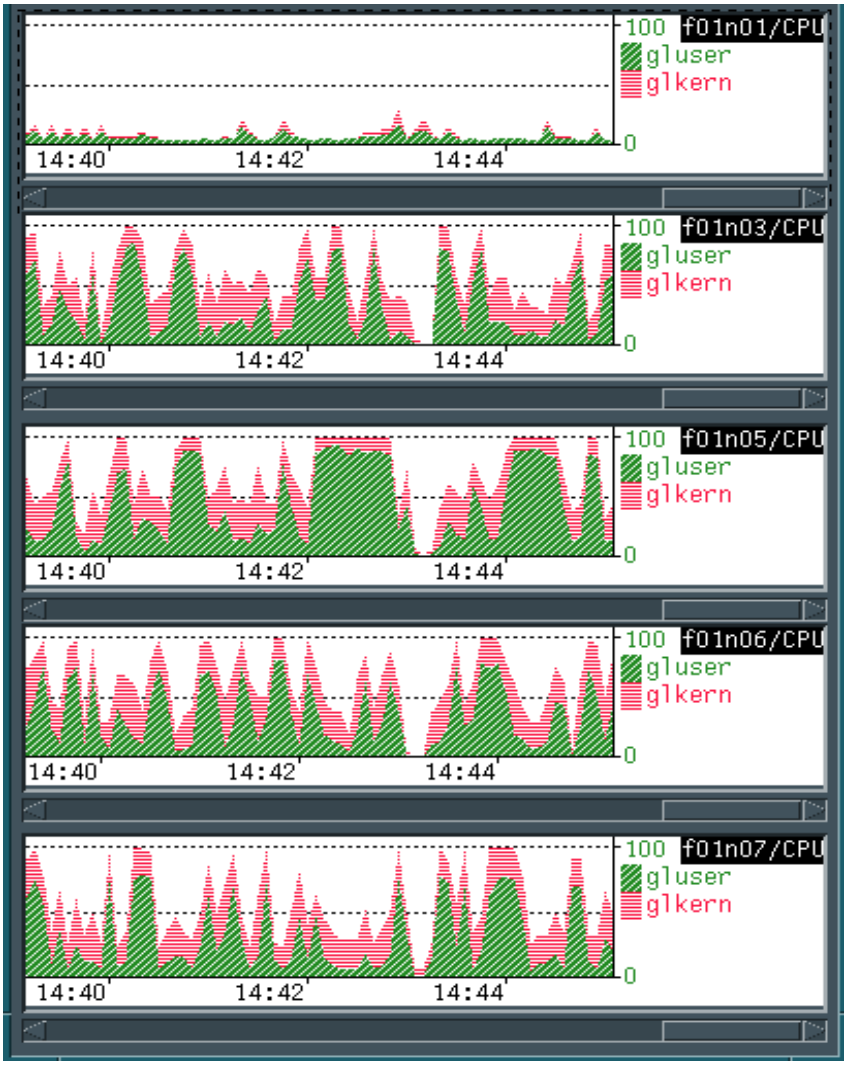


Figure 13. CPU Load Using Network Dispatcher for Dynamic WWW Pages

Figure 14 on page 57 shows the network traffic obtained with the same Network Dispatcher configuration. The number of total received IP packets iin the Network Dispatcher machine is very high because it works at IP level.

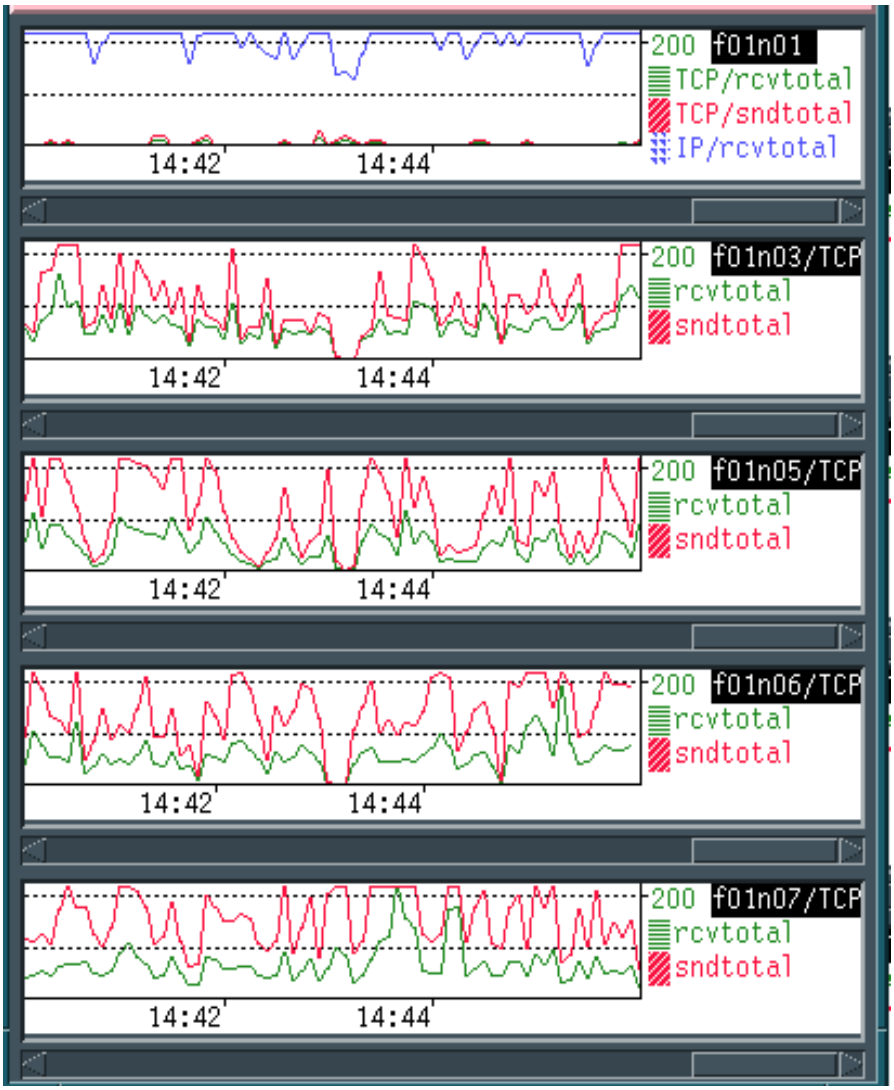


Figure 14. Network Traffic Using Network Dispatcher for Dynamic WWW Pages

Static Pages

With static pages, CPU load was very low. So, the main concern should be network traffic. If you run the Network Dispatcher advisors and/or ISS, you will have slightly higher network traffic due to the server information that the

advisors collect for the Manager and due to the communications between `issd` agents and the `issd` monitor and between the `issd` monitor and Manager, if they do not run in the same machine.

Network traffic distribution was smoother with both round-robin and best configurations than it was with ISS only. The client requests simulated were variable; that's why, in some short periods of time, the Network Dispatcher didn't receive any requests, and the server did not have any load. An example of the network traffic graphic is found in Figure 15 on page 58.

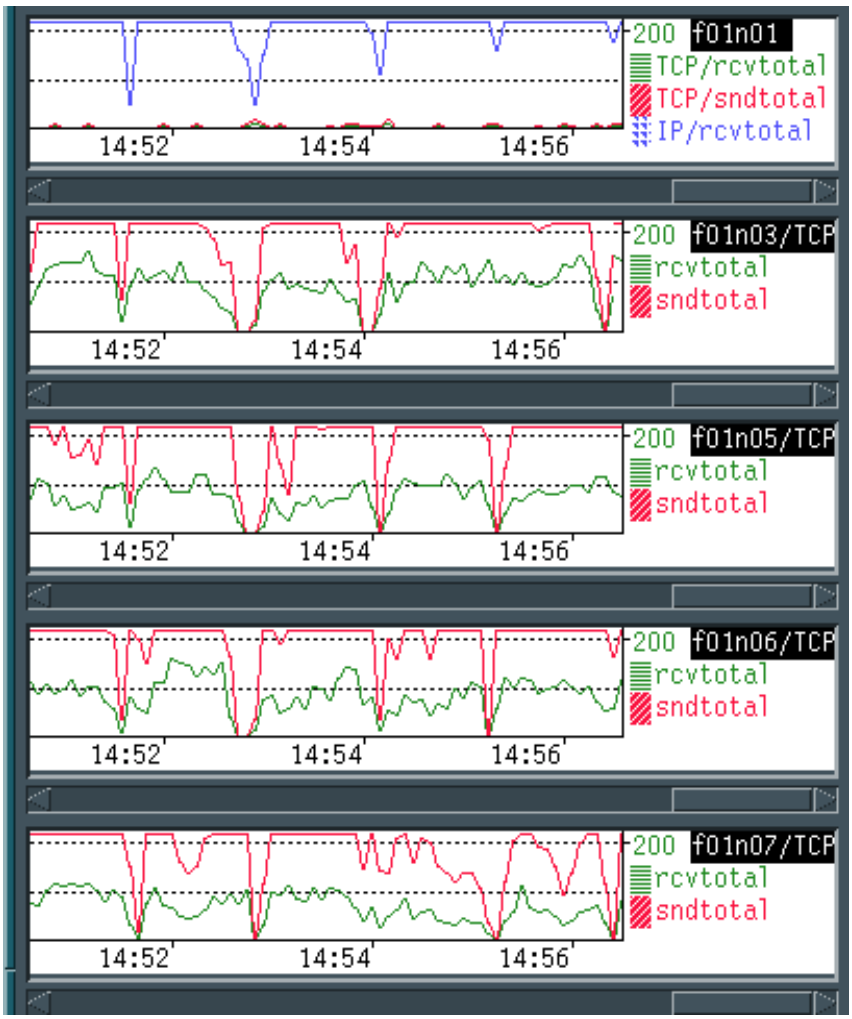


Figure 15. Network Traffic Using Network Dispatcher for Static WWW Pages

5.5 Conclusion

The `RoundRobin` metric is suitable only if load is well known and stable. It can load even a heavily loaded server by not considering actual load. The only advantage is that it is easy to configure.

The possibility of defining a `ResourceType` is suitable for every kind of application. Depending on your environment and on the service you want to provide, you can set up your own `ResourceType`. While CPU idle time is good for dynamic pages, network traffic is good only for static pages. Of course, measuring the CPU load you must remember that `issd` loads the CPU a little bit more.

Network Dispatcher uses results in a smooth distribution of CPU load for dynamic pages and of network traffic for static pages. Network traffic is higher when using both ISS and Network Dispatcher than when using ISS alone. This is the best way to distribute load among servers.

Chapter 6. FTP Server Scenario

FTP stands for File Transfer Protocol, which is defined as a protocol for file transfer between hosts on the TCP/IP network or on the Internet. The primary function of FTP is defined as transferring files efficiently and reliably among hosts, thus promoting the sharing of files. FTP is a convenient way to use remote file storage capabilities, shielding the user from variations in file storage systems among hosts. FTP is one of the most commonly used protocols on the Internet. FTP is a very useful and valuable Internet service; it is used programs, large databases, video, audio, and images. One of the most common types of servers available on the Internet are the FTP servers, which are servers with a repository of files for public access over the Internet. They are known also as *anonymous FTP* servers.

This chapter discusses some tests done using ISS and Network Dispatcher configured to provide FTP services.

6.1 Testing Environment Considerations

For the tests, a 5-node RS/6000 SP was used. Four nodes–two wide (nodes 3 and 7) and two thin (nodes 5 and 6)–were the FTP servers, and one wide node (node 1) was the ISS and Network Dispatcher server. More information about the test configuration files can be found in Chapter 10, “Configuration Files - Samples” on page 93.

During the tests, some graphics showing network traffic and CPU load in all nodes were collected using Performance Toolbox for AIX (instruments named *f01n01/CPU*, *f01n03/CPU*, *f01n05/CPU*, *f01n06/CPU*, *f01n07/CPU* for CPU load and *f01n01*, *f01n03/TCP*, *f01n05/TCP*, *f01n06/TCP* and *f01n07/TCP* for network load). Some of these graphics are shown in this chapter.

6.2 Load Generated by FTP Servers

FTP is a client/server application. On the client side, the user runs the `ftp` command, which opens a control connection with the FTP server. The FTP server on UNIX machines are implemented by a daemon called `ftpd`. The `ftp` command is a user interface; the subcommands the user types are translated into standard FTP commands and sent through the control connection using the Telnet protocol. The FTP commands specify the parameters for the data connection (data port, transfer mode, representation type, and structure) and the nature of the file system operation (store, retrieve, append, delete, and so on). To transfer data, the FTP processes use a different port, known as the

© Copyright IBM Corp. 1997

data port. On the server side, the standard port for control connection is 21, and for data, the port is 20. Figure 16 illustrates how FTP works.

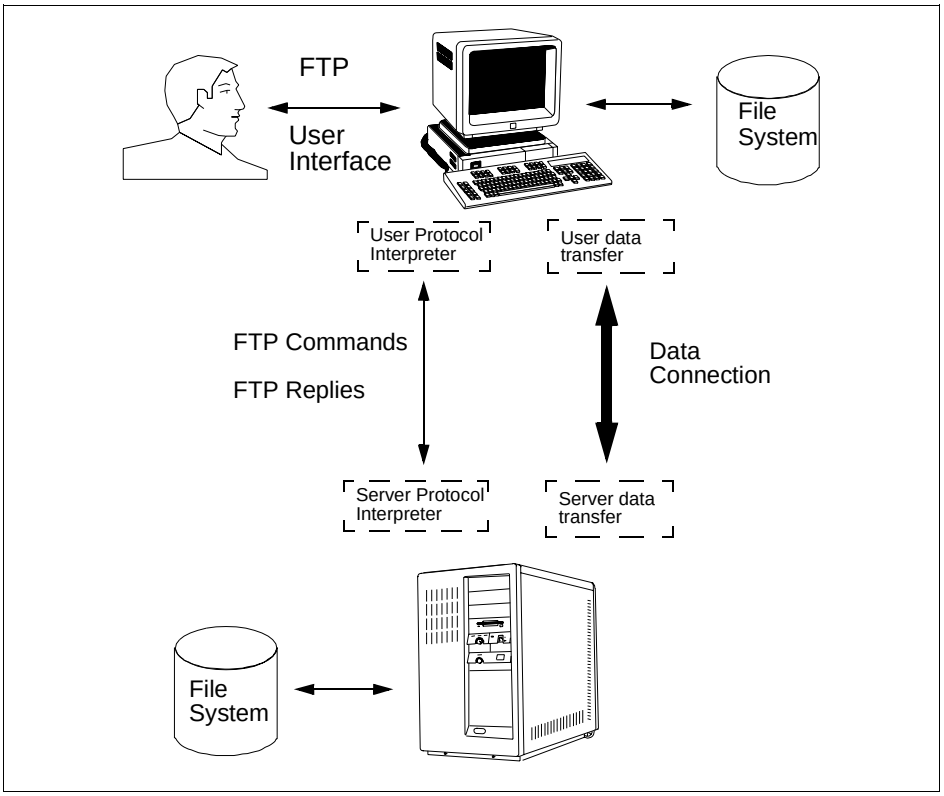


Figure 16. Interaction of the Client and Servers with File Transfer Protocol (FTP)

The FTP processes are not CPU intensive; the main load an `ftp` application puts on the server is network traffic. In order to transfer a file from the server and to the client, two TCP/IP connections are needed; so more packets are sent than with the HTTP protocol. Control commands just generate small packets. For example, the commands generated by the FTP user interface subcommands, `cd`, `pwd`, `bin`, and `prompt`, use only the control connection. But commands like `ls`, `get`, and `put` open a data connection to do a bulk transfer of data.

6.3 How the Tests Were Performed

To test this environment, we used the FTP server (`ftpd` daemon) including AIX and the `ftp` command as the user interface. All machines were connected through a token-ring network at 16 Mbps. In order to simulate various users accessing the server, we used a special utility called Expect. Expect is a program that "talks" to other interactive programs according to a script. Following the script, Expect knows what can be expected from a program and what the correct response should be. An interpreted language provides branching and high-level control structures to direct the dialogue. Expect is built on top of Tcl, which is an embedded scripting language. A public version of Expect can be downloaded from:

```
http://expect.nist.gov/
```

We wrote an Expect script to simulate a user accessing an FTP server using the `ftp` user interface. The script simply connects to the `ftp` server; it gives a user and a password, changes to a remote directory and selects a file to transfer (`get` command) at random. On the server, there were four different files with sizes of 10 KB, 50 KB, 500 KB, and 5 MB. The transfer time was measured. Because of the heavy load, some connections were timed out, in which case it was recorded in a log. The script executes a loop and repeats the same process for 10 minutes. It waits a random amount of time before starting the next `ftp` command (1 or 2 seconds). The script exits after 10 minutes, but waits for the current connection to finish or time out. We used four machines, each running 30 of these processes for a maximum of 120 concurrent FTP connections at all times. The load we generated, even pseudo-random, maintained a constant flow of TCP traffic through the servers, which is not always the case in a production environment, but it helps to determine the best configuration for ISS and Network Dispatcher with FTP servers.

6.4 The Tests and Their Results

This session describes the tests and shows the PTX graphics that explain the behavior of three different configurations; the traditional round-robin, a balanced distribution using ISS, and a balanced distribution using Network Dispatcher.

6.4.1 Testing with ISS

As with the WWW tests, we configured ISS in the `NameServer` mode, which sends a signal (SIGHUP) to the `named` daemon on each update, but as we said

before, the ISS machine did not have any CPU performance problems. Node 1 on the SP (*f01n01*) was configured as the name server and ISS monitor; the other four nodes (*f01n03, f01n05, f01n06 and f01n07*) were used as FTP servers.

6.4.2 Round-Robin Configuration

Round-robin could be a totally random way to distribute the load as it never knows when a server is too loaded or how many requests it is already serving. Many FTP sites use a random method to distribute the load; giving to the user a list of sites where he/she can download its files and which let him/her decide at "random". This may sound simple, but it has been proven to be a practical way to distribute the load. The ISS with round-robin will generate a slightly better distribution without bothering the user.

For the test, we set the heartbeat interval to 30 seconds and the heartbeats per update to 2 seconds. In this way a new server was chosen every 60 seconds. Since the load was constant, the monitor showed an evenly distributed load of traffic on all servers, with some peaks at certain times. These peaks were expected since ISS does not recognize when a server is heavily loaded. The results we obtained using ISS in our test is summarized in the Table 3.

Table 3. ISS Round-Robin FTP Simulation Results

File Size	Num.of Files	Total Bytes	Total Time	Avg KB/sec
10 KB	148	1515520	4.58	447.75
50 KB	125	64000000	113.84	594.79
500 KB	127	6502400	14.30	512.42
5 MB	135	42024960	72.98	625.26
Totals	535	108192880	205.70	545.05

The test ran for 13 minutes; so the total actual transfer rate for the cluster was approximately 1.43 MB/s. In this test, we set the time-out to 500 seconds, and we got 31 connections that timed out. ISS does a good job of distributing the load over the servers, but as we said before, this is not the typical load on an FTP server.

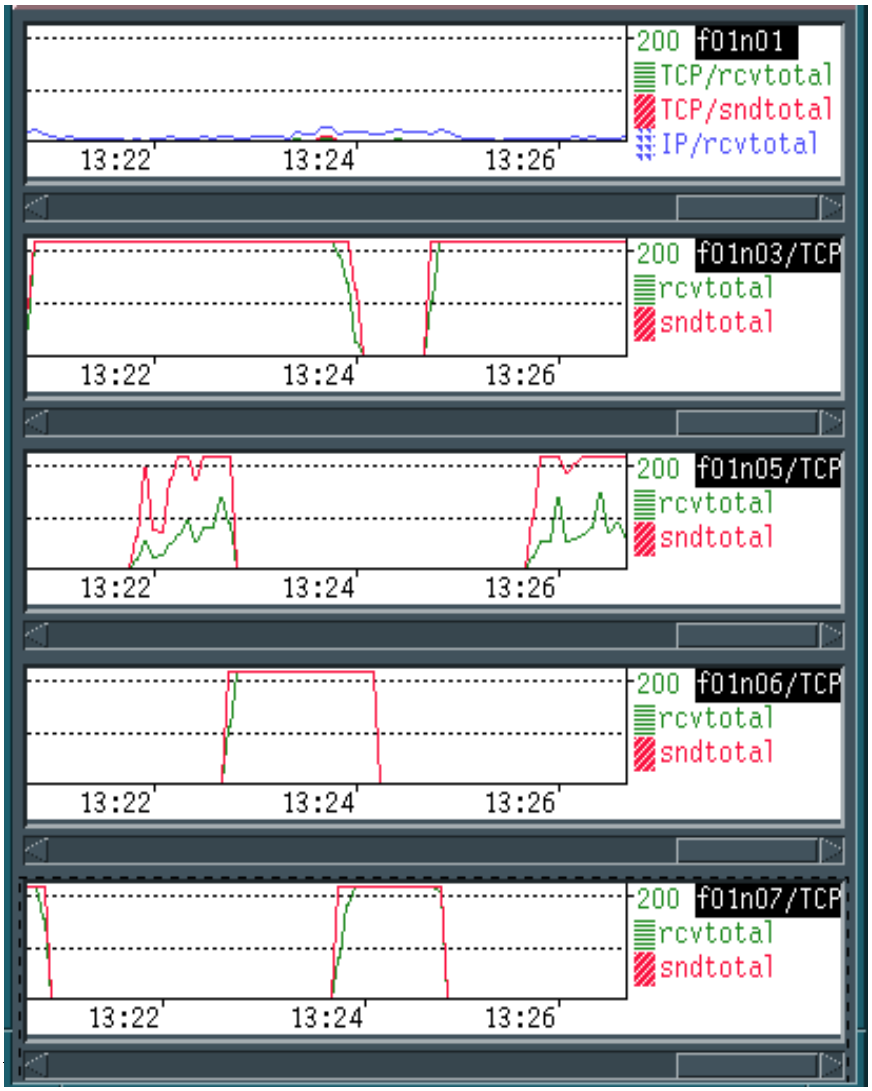


Figure 17. Network Traffic- ISS with the Round-Robin Configuration

Notice that the traffic is evenly distributed among all the servers most of the time. As you can see, there is one heavily loaded server, but others are getting loaded in turn. This is the normal behavior you would expect when using ISS in round-robin.

Other problems you may find with ISS is the response time to the users, since sometimes they will connect to a heavily loaded server and other times to a lightly loaded one. The response time they get varies greatly.

6.4.3 ISS with Best SelectionMethod

To have a better distribution of the load, you can add some intelligence to the server selection process. Because FTP load is over network traffic, we chose a metric that takes into account IP traffic over the token-ring interface. We took the output from the `netstat` command over the interface tr0, and from this we selected the input and output packets and output the sum as the metric value. On the ISS configuration file, we had the following line:

```
ResourceType External netstat -I tr0 | awk ' NR == 4 { print $1+$3; exit } '
```

We set the heartbeat interval to 3 and the heartbeats per update to 2 seconds; so every 3 seconds, the ISS monitor received input from the ISS agent, and every 6 seconds it updated the `named` daemon (if necessary). The overhead of the ISS agents on the nodes for both on traffic and CPU was minimal. The Figure 18 shows some peaks on node 1 (*f01n01*) that corresponds the `issd` daemon CPU usage.

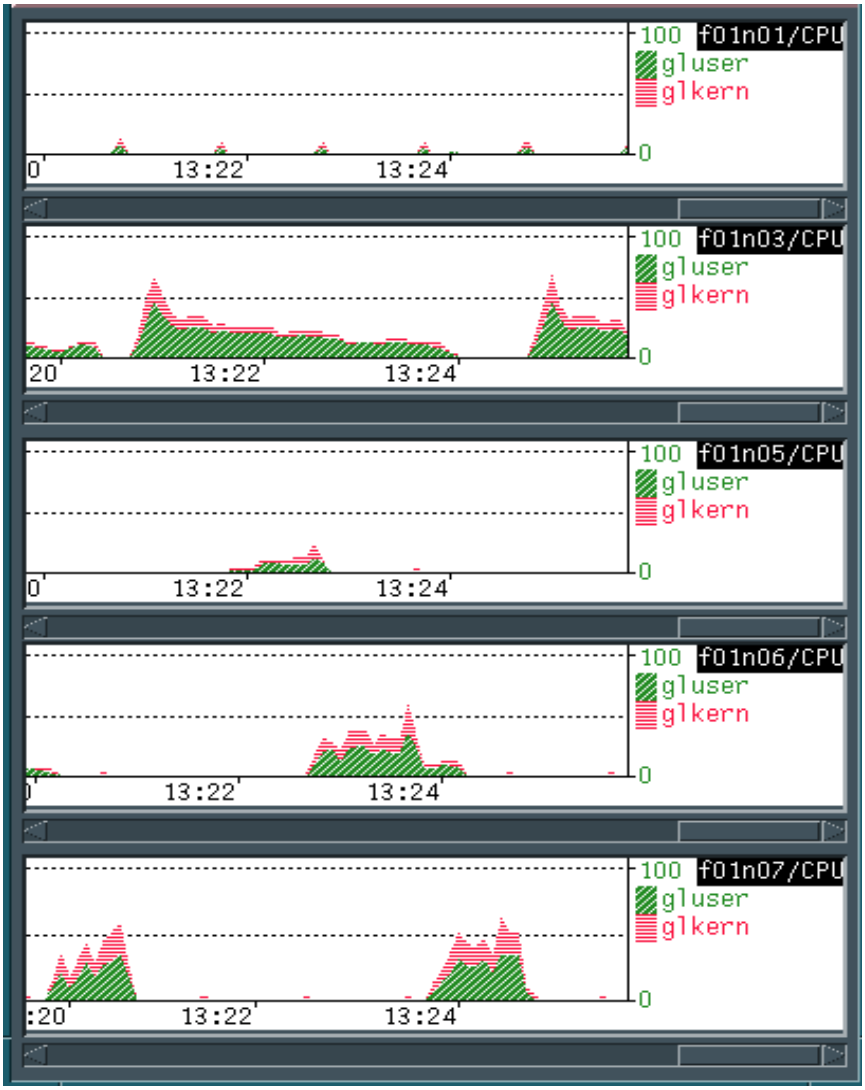


Figure 18. CPU Load - ISS with the Round-Robin Configuration

The test ran for 13 minutes. The actual transfer rate for the cluster was approximately 1.48 MB/s, which is just slightly better than the result we got with `RoundRobin`, but in this case we just got five connection time-outs, which was a significant improvement over the round-robin configuration. The distribution of work was also slightly better than with the round-robin configuration.

Another advantage that this metric gave us over the ISS in round-robin was the average response time (time to transfer a file). Using ISS with a custom metric to balance load, the clients got approximately equal response time on each request, meaning that there are not many changes from the response time of each subsequent request. We could also see that no client got a response time worse than 350 seconds as opposed to the round-robin configuration, where some clients got a response time of over 400 seconds.

6.4.4 Network Dispatcher

FTP has special consideration when used with Network Dispatcher. As we explained before, FTP uses two connections to make a transfer. The first connection is the control connection, where the user specifies the parameters for the data connection. When a transfer is going to take place, the client side opens a new process that "listens" to a specified data port for a connection from the server. The server initiates the data connection and makes the data transfer in accordance with the specified parameters.

Since the server is the one that initiates this connection, the connection is not known by Network Dispatcher. All the traffic going from the client to the server in the cluster must pass through the Network Dispatcher. The only way to ensure that this connection is going to reach the same server is though the affinity between the client and that server implemented by Network Dispatcher. In this way, all further FTP traffic (destination ports 20 and 21) from the client machine will be routed to the same server after the first connection is established. This means that if a second FTP process is initiated by any other users while the first user is still connected, it will be routed to the same server.

To define the FTP ports to the Network Dispatcher, you must enter the following commands:

```
CLUSTER=pool2.sp.itsc.austin.ibm.com
ndcontrol port add $CLUSTER:21
ndcontrol port add $CLUSTER:20
```

Some FTP clients support passive FTP, in which case the server data transfer process is placed in a "passive" state instead of the normal, active state. The server data transfer process then waits or listens for a connection rather than initiating a connection on the data port. If you are going to use FTP clients with the passive option, you need to enter the `pftp` option on the control port definition:

```
CLUSTER=pool2.sp.itsc.austin.ibm.com
ndcontrol port add $CLUSTER:21 pftp
ndcontrol port add $CLUSTER:20
```

The proportions for the network dispatcher were set to 50-50-0-0. However, since we only had four clients to test, all connections from one client went to the same server. In our case, four servers received all the connections. This can be seen in Figure 19 on page 69.

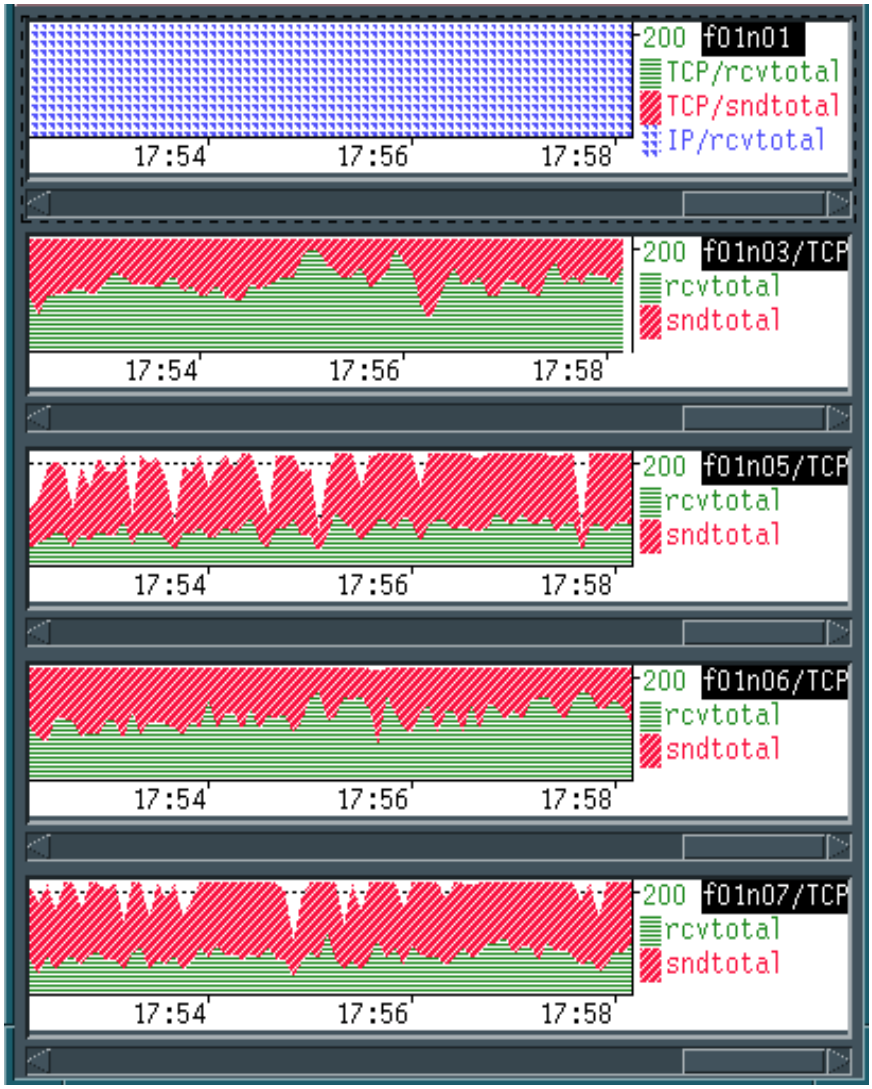


Figure 19. Network Traffic - Network Dispatcher - FTP Simulation of Load

In Figure 19, we can also see that FTP generates many packets, all of which are routed through node1 (*f01n01* in the graphic). The total number of packets generated by FTP are usually more than the ones generated with HTTP because of the two connections; so more network traffic load is generated on the network dispatcher node.

6.5 Conclusion

The `RoundRobin` metric is adequate to distribute the load over your FTP servers. Most of the time, the load will be evenly distributed, but the clients sometimes will experience bad response time.

ISS using a custom metric seems to be a good configuration option for this case; the overhead on CPU time and network traffic is insignificant. Your servers will have a well distributed load, and the clients will get a response time proportional to the load of all the cluster.

The Network Dispatcher would be a good option to use for balancing FTP traffic in most situations, except when the number of client machines accessing your FTP pool of server is small. In this case, all traffic will be routed to the same server until all active FTP connections are closed. This may not be a problem when your clients are coming from the Internet, where the odds of receiving many connections from the same client are very small, but it may be a problem on an Intranet, where there may be just a small number of server machines hosting multiple users.

Chapter 7. Other Server Scenarios

Although the most common Internet servers are WWW and FTP, there are other very important scenarios where Interactive Network Dispatcher can be useful. This chapter addresses some of these scenarios

7.1 Telnet Server Scenario

The purpose of the Telnet protocol is to provide a fairly general, bi-directional, eight-bit, byte-oriented communications facility. It is used for users to log in a host on the network. It uses a TCP connection to transmit its data. Although most of the people use it just to emulate a remote terminal, its primary goal is to allow a standard method of interfacing terminal devices and terminal-oriented processes to each other. Conceptually, the protocol may also be used for terminal-terminal communication ("linking") and process-process communication (distributed computation).

In order to have a pool of servers giving access to several users, the environment each server provides to the users must be indistinguishable from the others. They must provide the same password and same data to the users. This can be accomplished by using several types of distributed file systems and user-management systems, such as NFS/NIS and DFS/DCE. Depending on your application, you can also use a distributed database management system (DBMS) like DB2, Oracle or Sybase.

The load generated on the server by the Telnet clients is application-dependent. The users connected to the server over a virtual terminal and then execute whatever command or processes they have access to. The users may start a database application, a spreadsheet, and so forth. There is not a generic load that will be generated on the server, but some loads may be more common than others. In general, user applications over a Telnet session will consume CPU, memory, and I/O most of the time, but not network bandwidth.

With ISS for AIX Release 1.0, we performed a test on Telnet. Based on the results, we reached the following conclusion.

The load on the servers from requests coming from Telnet clients will always depend on the applications and user profiles. The best way to balance this load is to use a metric that reflects the load of your application. A good option is to use ISS with a custom metric. The option that gives more granularity on directing the request is Network Dispatcher with feedback from the ISS agents.

© Copyright IBM Corp. 1997

7.2 Domino Server Scenario

Lotus Notes Workstation and Lotus Domino Server provide a powerful client/server environment for workgroups; simply stated, they are groupware. Lotus Notes allows people to work together regardless of the software or hardware platforms they are using and regardless of any organizational or geographical boundaries. It enables organizations to communicate, collaborate, and coordinate key business processes. Lotus Domino is a server technology that transforms Lotus Notes into an interactive Web application server, thus allowing any Web client to participate securely in Notes applications.

Bridging the open networking environment of Internet standards and protocols with the powerful application development facilities of Notes, Domino provides businesses and organizations with the ability to rapidly develop a broad range of business applications for the Internet and intranet. The key elements of Lotus Notes are its ability to distribute object store, its powerful messaging system, and its application development environment. In its basic form, Lotus Notes is a document database that has the ability to distribute information throughout the enterprise through replication.

A Domino server can be accessed through a Lotus Notes Workstation or through a Web browser. When configured in a cluster of multiple servers, Domino Advanced Server has its own load balancing feature for Notes Workstation clients. You can specify a server availability threshold on the notes.ini file, and when the availability index falls below this threshold, clients are redirected to another server. This option is only valid for Notes Workstations. For load balancing of requests coming from a Web browser, you can use ISS and Network Dispatcher.

Even if the documents we accessed may be considered "static" in content for a standard benchmark, these were really dynamics pages for Domino. This is because each HTTP request from a client activates a thread in Domino which has to open the database, access the particular document, and translate it to HTML format. Domino generates custom HTML pages from the object store on-the-fly based on time, user identity, user preference, client type, and data input.

Domino’s object store differs from a file store in that it separates form from content (page layout and page data); form and content are joined when a document/page is requested. At the time of the request, Domino may also integrate information from other data sources such as a relational database. So in the benchmarks, we did not make a distinction between static and dynamic pages; for Domino everything is dynamic. Also because of the way

Domino handles requests, the most constrained resource is CPU most of the time.

With ISS for AIX Release 1.0, we performed a test on Domino. Based on the results, we reached the following conclusion.

As we said before, the Domino servers will constrain the CPU of the servers most of the time; so the best load balancing solution will be one which takes this resource into consideration. ISS with a continuous custom metric is a good option to balance the load of a Domino server, but the best option for load balancing of a Domino server is Network Dispatcher with feedback from the ISS monitor using a continuous metric. Although ISS in `RoundRobin` shows good results, this may not be always the case. In the environment we tested, the load was always uniformly generated by the WebStone clients. In a real environment, this would not always be true.

Chapter 8. High Availability Server Scenario

The high availability configuration detects and recovers from network and server failures. Interactive Network Dispatcher is smart enough to determine that a server or a network is down. In case of failure, clients lose only the current connections, but they can immediately establish a new connection to the remaining servers with no problem.

This chapter covers concepts and the results of some tests performed using highly available configurations with ISS and Network Dispatcher.

8.1 ISS High Availability

The new release of Interactive Network Dispatcher now eliminates the ISS machine running in monitor mode as a single point of failure. The `issd` process running in monitor mode can be in a different machine than the one which works as a DNS name server (if you are using either the `NameServer` or the `ISSNameserver` Observer). When you declare the nodes that will be part of the ISS cell, you can define them either with the `Node` or with the `Node NotMonitor` keyword.

A `Node` machine is a node in which the `issd` process can run in monitor mode. You define for each `Node` a priority number. The `Node` machine can be a server of one or more services, can simply work as an ISS monitor, or it can wait in stand-by mode. If the first node fails, the next in the priority list takes over as `issd` monitor. As soon as the failed node is operational again, it will become the monitor because of its higher priority number.

A node `Node NotMonitor` machine is a server of one or more services of the ISS cell, and the `issd` process runs only in agent mode. For a more complete explanation of the keyword used to define a server that is part of a cell, see “ISS Configuration File” on page 28.

8.2 Failure of a DNS Server

High availability is not guaranteed if ISS is configured for working with a `NameServer` or `ISSNameserver` Observer and the DNS machine fails. If clients point only to that DNS machine, configured to be either a `NameServer` or an `ISSNameserver`, ISS can no longer provide any service when fails. A solution could be to set up a particular configuration providing the client with a substitute name server.

© Copyright IBM Corp. 1997

This option makes use of a feature of the TCP/IP name resolution system. This feature provides the ability to configure a client or a site name server to use multiple name servers. In a UNIX client system, this is implemented by simply listing multiple servers in the /etc/resolv.conf file. The name servers will be used in the order they are listed. The client will contact the first listed server to resolve a name or address. If the first server does not respond, the client will then contact the second server. This can continue to further servers if required.

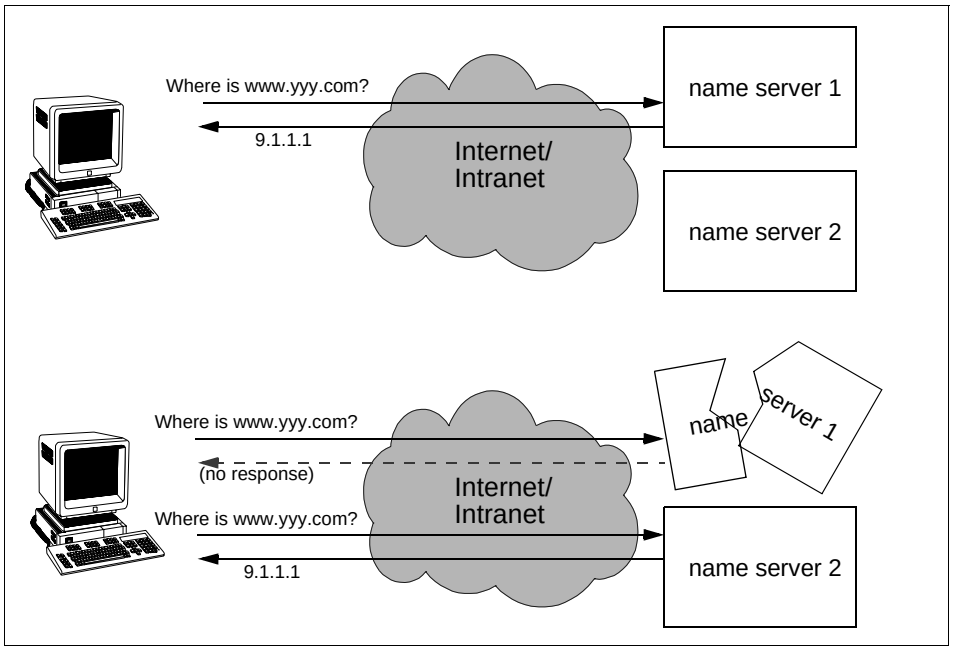


Figure 20. Configuring a Client to Use Multiple Name Servers

It is important to realize that the second name server will only be queried if the first server is not able to answer the request. However, multiple clients can be configured so that some clients list the name servers in the opposite order to spread the load. Each name server would then act as a backup for the other. This is a very common configuration in large networks.

To use this type of configuration with ISS, you will need two (or more) ISS systems. Each one should be configured as a DNS server for the ISS-served domain, either using the `named` daemon with the `NameServer` Observer or with the `ISSNameserver` Observer. Each server should contain the necessary ISS and DNS configuration files, which can be part of the same ISS cell.

An example of this configuration is shown in Figure 21.

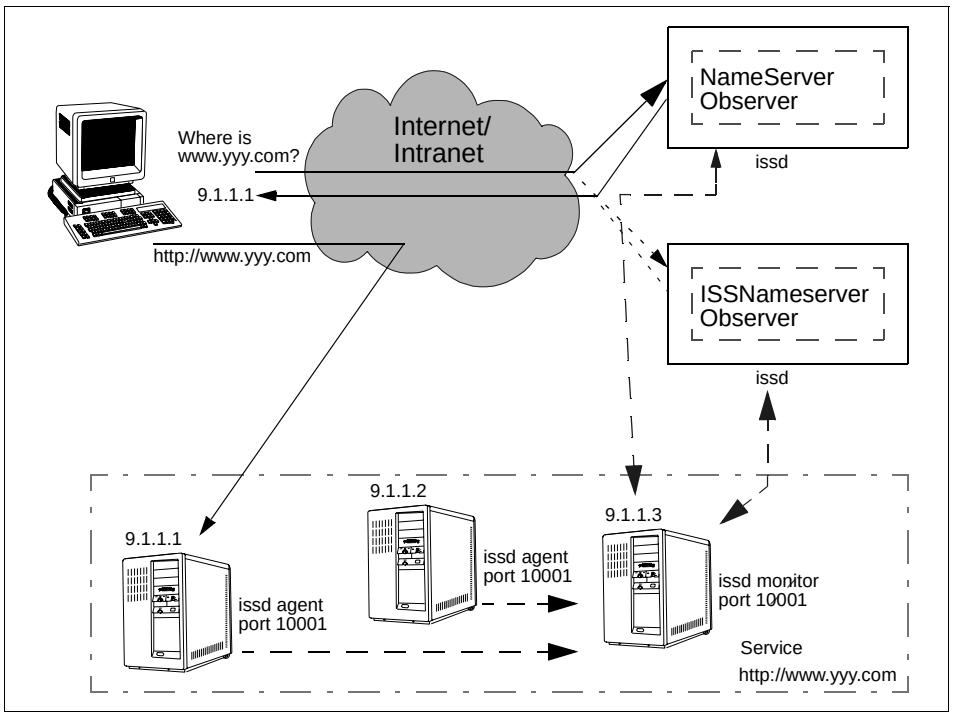


Figure 21. Example of Multiple ISS Servers in a High-Availability Configuration

This technique cannot be used with Network Dispatcher. It requires that each Network Dispatcher server system be configured with the cluster address aliased to the network adapter. Doing this on two systems in not permitted in TCP/IP because this would mean having the same IP address defined on two different systems on a single network.

8.2.1 One Possible Solution

Configure the systems for this environment as follows:

1. If you have a DNS forwarding requests to the cell name servers, you should ensure that the DNS server for the site has a name server entry listing each of the ISS systems as name servers for the cell domain.

   If you have clients sending DNS requests directly to the ISS systems, ensure that all of the ISS systems are listed in the clients' /etc/resolv.conf or equivalent files.

2. On the ISS systems, you should install ISS for AIX and store the ISS configuration files on a local disk or on some shared disk technology that will still be available in the event that the other ISS system fails.
3. If the `NameServer` Observer is being used, you should configure DNS on each system, again storing the files locally. Start the `named` daemon. If you use a `RoundRobin` metric type, no additional configuration is required.
4. If you use the `LoadLeveler` metric type, you must install the LoadLeveler product on each system, and ensure that they are configured as LoadLeveler servers capable of scheduling jobs.

8.2.2 Pros and Cons

The main advantage of this solution is that it is extremely simple to configure both clients and servers using this method. Also, no additional software or configuration and management skills are required.

The main disadvantage of this solution is that it does not work for Network Dispatcher because of the duplication of IP addresses on Network DIspatcher systems that this would require.

8.3 Network Dispatcher High Availability

This release of Interactive Network Dispatcher comes with a new high availability feature. You can now configure a backup machine that takes over the primary machine without using the HACMP/6000 product.

The environment involves a *primary* machine that works normally as a Network Dispatcher, balancing the load among the servers of its clusters. You configure a *backup* machine, too, that stands in stand-by mode unless the primary fails. The two machines are synchronized, and only the primary routes packets while the backup one is continually updated. The two machines establish communication to monitor the status of each other, referred to as a *heartbeat*, using a port that you can choose. If the primary fails, the backup detects this failure and begins to take over the routing packets. When the primary machine is operational again, you can either decide that it again automatically becomes the primary machine, or leave it in stand-by mode and act manually if you want it to become the primary again.

Figure 22 on page 79 shows an example of the Network Dispatcher high availability feature. The two machines are synchronized. There is one cluster for the HTTP service on port 80. Every server has an alias for the cluster IP address on the loopback interface. The standby machine has the same configuration file as the primary one (apart from the nonforwarding address)

and is continually updated. It does not route packets. It has an IP cluster alias to the loopback interface, and the primary Network Dispatcher has an IP cluster alias to a network interface (tr0, en0, etc.).

When a client sends a request to the HTTP service, the Network Dispatcher selects the best server that responds directly to the client.

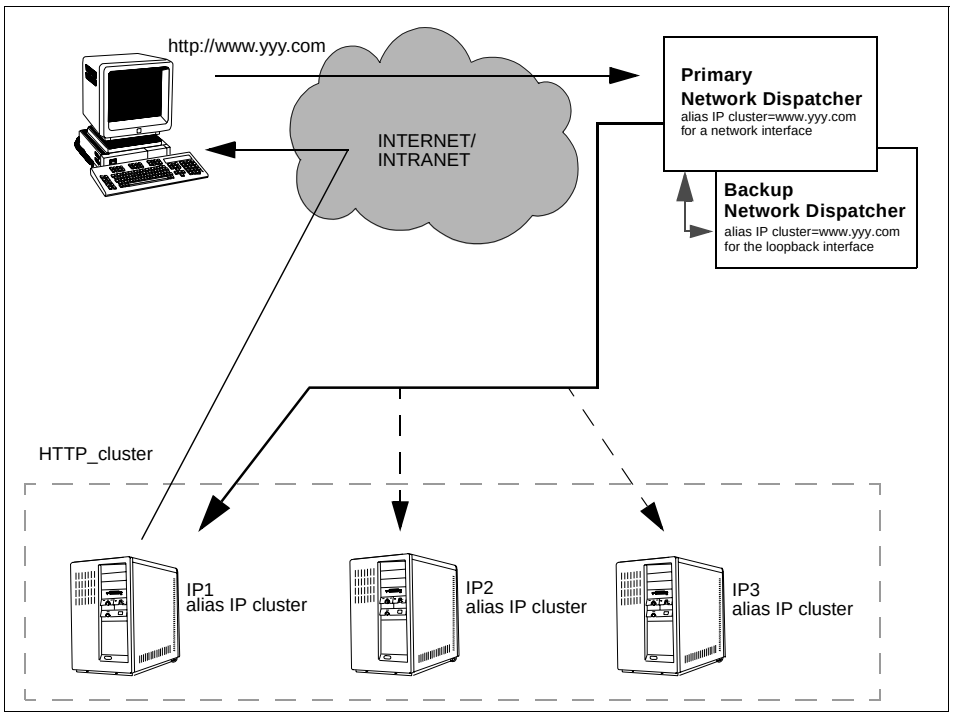


Figure 22. Network Dispatcher High-Availability Feature

8.3.1 Configuring Network Dispatcher High Availability

Both the *primary* and the *backup* machines must have their own nonforwarding address for remote administration and configuration. In the primary machine, for every address, there is a corresponding alias for a network interface (tr0, en0, and so forth), but the clusters' addresses in the backup machine must be aliased to their `loopback` interface.

All the processes (Executor, Manager and Advisors, if you are planning to use them) must run in both machines. All the necessary configuration files regarding cluster, port and server information must reside in both machines. Then you let the machines know about the availability configuration with the command:

`ndcontrol highavailability heartbeat add` *sourceaddress destinationaddress*

Of course, the *sourceaddress* and *destinationaddress* (IP addresses or DNS names of the primary and backup machine) will be reversed on the primary and backup machines.

Then you have to declare for each machine whether it will be the primary or backup by using the following command:

`ndcontrol highavailability backup add primary` [`auto` | `manual`] *portnum*

on the *primary* machine, and the command:

`ndcontrol highavailability backup add backup` [`auto` | `manual`] *portnum*

on the *backup* machine. The *portnum* field indicates the port over which the two machines will communicate. If you set the `auto` option, as soon as the primary machine becomes operational again after the takeover, it will become the active one automatically. If you set the `manual` option, the backup machine that takes over after the failure of the primary should remain active and continues to route packets even if the primary becomes operational again.

Auto Option

If, after a takeover event, the `auto` option does not work well, you should again run the commands:

`ndcontrol highavailability backup add primary auto` *`portnumber`*

`ndcontrol highavailability backup add backup auto` *`portnumber`*

on the *primary* and *backup* machines.

The `manual` option has the advantage that you can run the following command:

`ndcontrol highavailability takeover`

whenever you want to make the standby machine become the active machine.

Manual Option

During our tests, when the active machine failed, the backup took over immediately without waiting for a manual intervention. If we wanted the backup machine to go on routing the IP packets even after the primary was operational again, we had to run the takeover command.

If you want to change some configuration parameters, you must run the `ndcontrol` command first in the backup machine and then in the primary.

To delete a high availability configuration, you need to stop the Executor on one machine; then delete all the high availability definitions on the other one.

Like HACMP/6000, you need some scripts to make the high availability configuration work. These reside in the /usr/lpp/ind/nd/bin directory and are named:

- goActive script

  This script deletes every loopback alias and adds device aliases. It will be executed when a Network Dispatcher machine planned to be the primary machine in a high availability configuration goes into active state and begins routing packets.

- goStandby script

  This script deletes every device alias and adds loopback aliases. It will be executed when a Network Dispatcher machine planned to be the backup machine in a high availability configuration goes into stand-by state without routing packets.

- goInOp script

  This script deletes all device and loopback aliases. This script is executed when a Network Dispatcher Executor is stopped and before it is started for the first time.

8.4 Testing Environment

To test the high-availability feature, we used a 5-nodes RS/6000 SP. Node f01n03t was the primary Network Dispatcher; node f01n01t was the backup machine, and the other three nodes were configured as servers listening on port 80 for an HTTP cluster.

Configuration Hint

One of the default ports used by Network Dispatcher (port 10003) is also used by one of the SP daemons. So, if you use the Network Dispatcher server as one SP node, do not forget to modify this port number substituting the 10003 with another free port in the ND_PORT line of the /usr/bin/ndcontrol and /usr/bin/ndserver files.

We changed the port from 10003 to 50003.

You can find the configuration files we used on Chapter 10, “Configuration Files - Samples” on page 93.

The environment is the same as the one described in Figure 22 on page 79. We started all the daemons in both the machines, wrote the goActive, goInOp and goStandby files and set up the `auto` option. We tested the environment with WebStone, simulating a load of 200 clients asking for dynamic WWW pages. Node *f01n03t* was operational until we issued the following command:

```
ifconfig tr0 down
```

to bring the token-ring interface down. As is shown in Figure 23 on page 83, node *f01n01t* took over packet-routing immediately. The PTX window of node *f01n03t* finished working as soon as the interface went down. That is why the time of the PTX window is different from the others. From the end-user point of view, nothing has changed. The network traffic in node *f01n01t* before the takeover was also due to the communication session that the two machines established in order to be synchronized all the time.

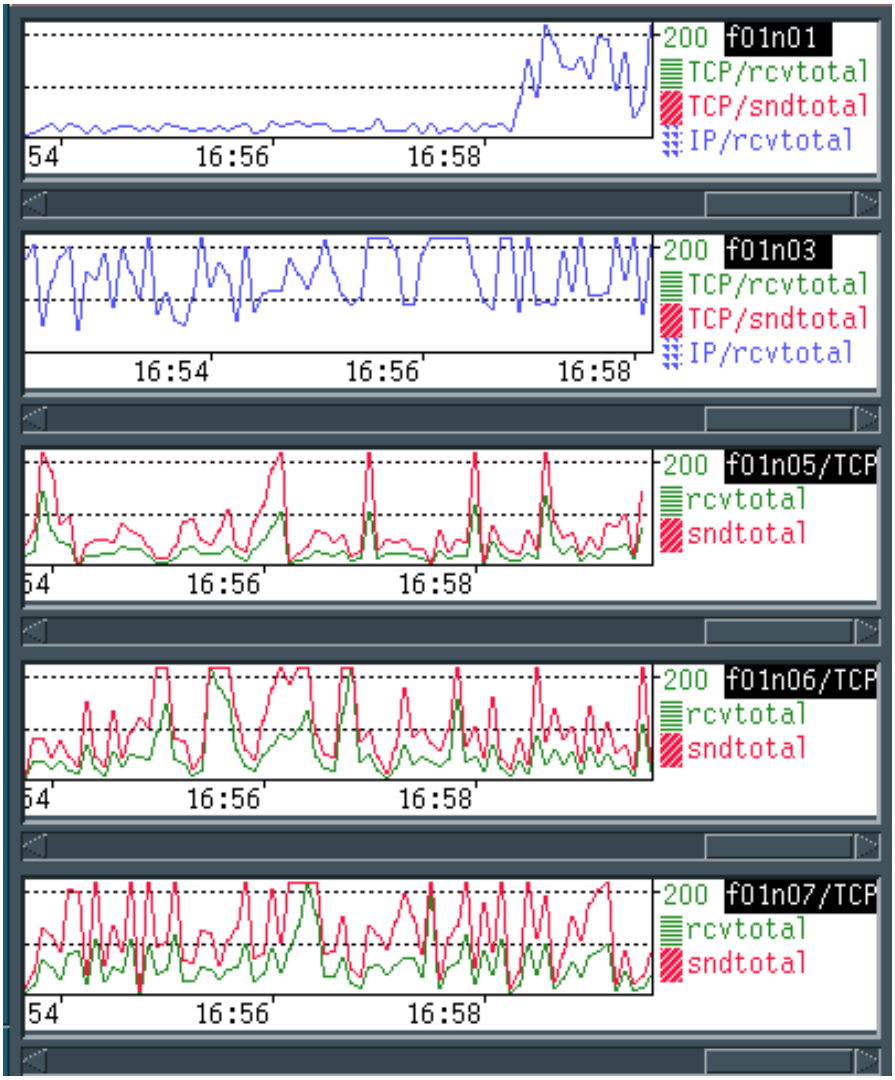


Figure 23. Network Traffic with a High-Availability Configuration

As soon as the primary machine failed, the loopback IP alias to the cluster address configured in the backup machine passed to the token-ring interface as written in the goActive file. When the primary machine became operational again, the alias returned to the loopback interface as written in the goStandby script.

Figure 24 on page 84 shows the CPU behavior for the same test. The CPU load on the servers is heavy because we used dynamic WWW pages. You can see the higher CPU load in node *f01n01t* as soon as it began actively routing packets.

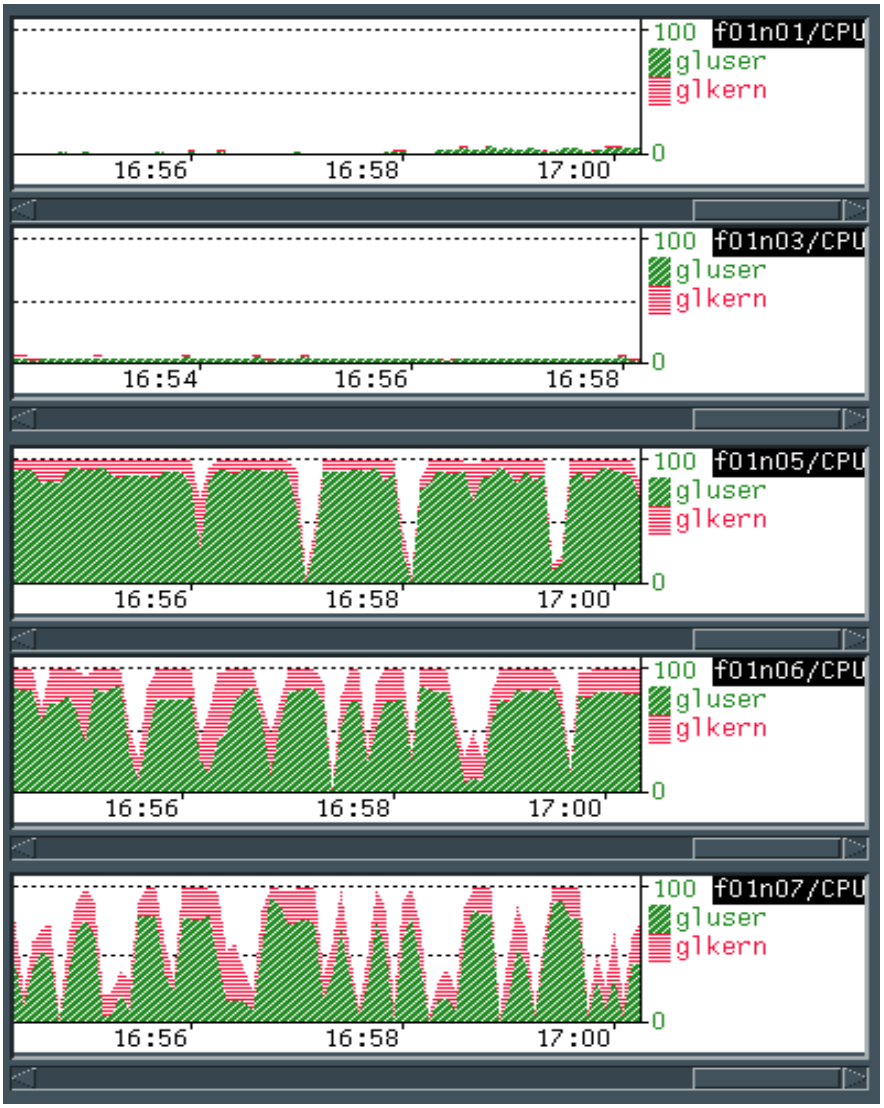


Figure 24. CPU Load with a High-Availability Configuration

8.5 Conclusion

The new high availability feature of Network Dispatcher has the great advantage that is much simpler to configure than HACMP/6000.

If you decide to use the `auto` strategy, you should make sure, after a takeover event, that when the primary machine becomes operational again, it becomes active, as well. If you decide to use the `manual` strategy and want the *backup* machine to remain active after a takeover event, and the *primary* to wait in standby mode, you should run the takeover command from the *backup* machine.

Chapter 9. Two-Tier Network Scenario

Sometimes having just one site to store and manage your information is simply not enough. The load over one site may be so high that it can not be handled. Besides the load, you may have other requirements for having more than one site, such as the need to have replicas of your information over different geographical areas for safety reasons. In case of a catastrophic event at one site, your sites in different locations can continue providing the service. This will also increase your general serviceability, allowing you to shut down a site for maintenance while the others are processing the incoming requests.

ISS in conjunction with Network Dispatcher allows you to create a two-tiered configuration for a cluster of servers located on a remote network. The first tier, or front-end tier, is composed of a machine that runs the `issd` monitor. This machine manages the front-ends of the second tier, which are Network Dispatcher machines. After receiving a request, the `issd` monitor machine will send back the IP address of the Network Dispatcher machine that will handle this particular request. The decision as to which IP address is sent back can be as simple as a round-robin selection, or it can depend on the feedback of `issd` agents running on the Network Dispatcher's machines that measure the load for each specific cluster. Network Dispatchers are the front-end of the second, or back-end, tier. They will route the request to the most appropriate server on that cluster at that moment.

The process is illustrated in Figure 25 on page 88. In this example, a client is trying to access an HTTP server with the name www.yyy.com. At step 1, the client lookup request is attended to by the `issd` monitor machine. If the `issd` daemon is not replacing the name server, the `named` daemon must be running in that machine. The `issd` daemon determines which Network Dispatcher cluster is the least-loaded at this moment, and at step 2, it sends this IP address back to the client. At step 3, the client sends the HTTP request to that IP address. Network Dispatcher receives the request and routes it (step 4) to the least-loaded server in the cluster at that moment. The server gets the request and responds directly to the client at step 5. The client has been redirected to the least-loaded server in the least-loaded cluster without ever noticing. Network Dispatcher and the cluster of machines it is managing need to be on the same network, but each of these clusters can be on any network around the world. The `issd` monitor machine can also be placed on any remote network.

© Copyright IBM Corp. 1997

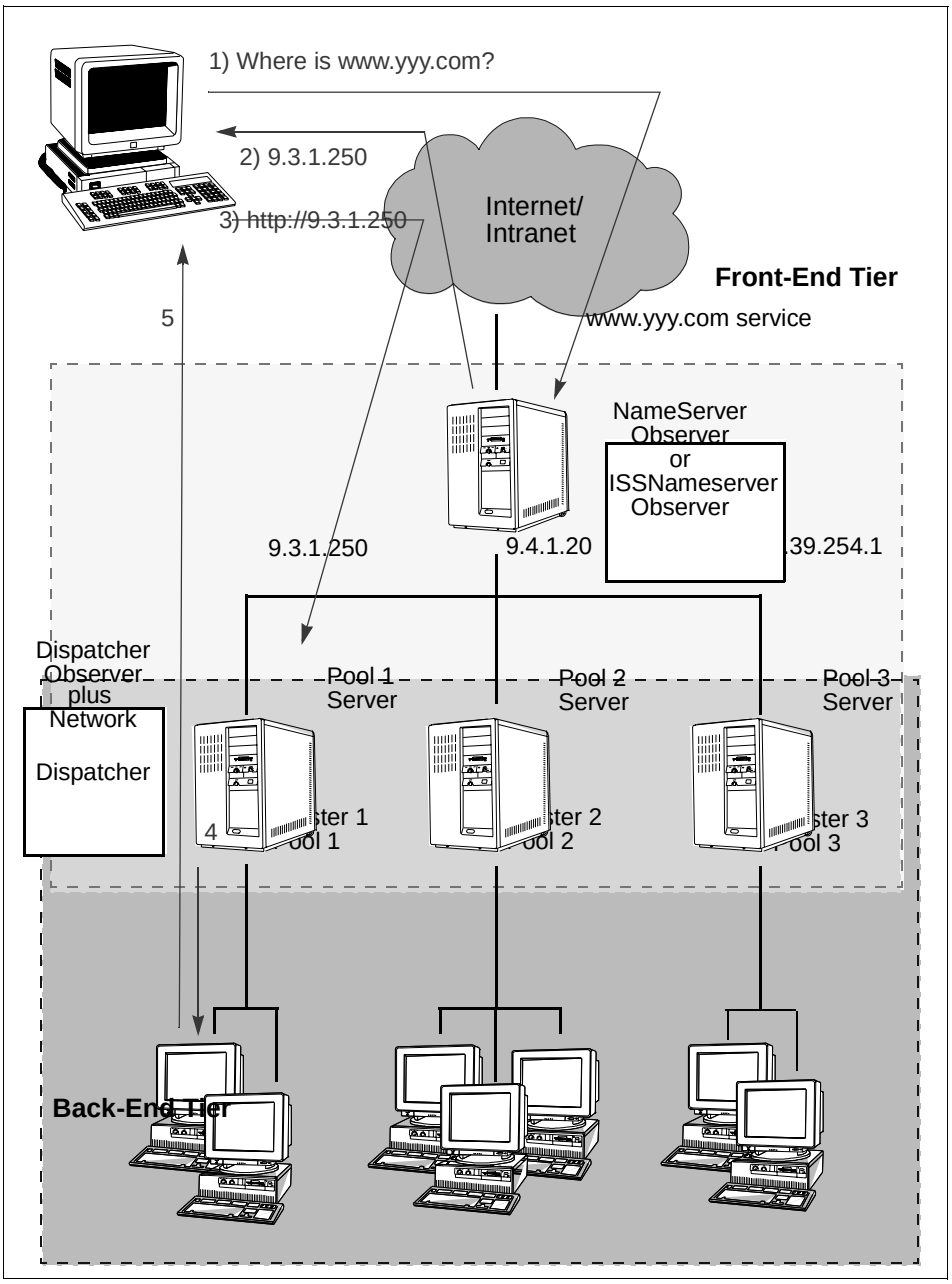


Figure 25. Redirecting Requests on a Two-Tier Configuration of Clusters

If you want a cluster to also use ISS to provide feedback to Network Dispatcher, you can define as an `issd` monitor one of the servers, the Network Dispatcher machine, or a different machine.

9.1 Front-End Tier Configuration

On a two-tier configuration, the `issd` running in monitor mode can have two functions. On the front-end tier, you must have it running. ISS is the first interface the clients get when sending a request into a two-tier network. ISS in the first tier has to be configured either to work with a `NameServer` Observer or with an `ISSNameserver` Observer. For the second tier, ISS can optionally be configured to work with a `Dispatcher` Observer to provide load feedback to Network Dispatcher.

The ISS configuration for a two-tier network is similar to the normal configuration. Assuming that you also want to use ISS to provide feedback from the servers to the Network Dispatcher, you should define several services, each one corresponding to the IP address of one cluster. Then you define the service provided by the `issd` monitor machine of the first tier. To define the set of Network Dispatcher machines to which this service is going to forward the request, use the `NodeList` keyword followed by the IP addresses of the clusters that you are planning to use in the back-end tier. There are no requirements for machines to be within the same Internet subnet or in the same DNS domain; all they need is a valid IP address accessible from the client and from the name server machine. For example, to define the service shown in Figure 25 on page 88, you can have the following lines on the ISS configuration file:

- To define, for example, one cluster HTTP service:

  ```
  Service http_dispatcher  pool1 9.3.1.250  80
  ```

  This service will be used by the `Dispatcher` Observer of this cluster to provide feedback regarding the load of the servers.

- To define the front-end tier `issd` monitor:

  ```
  Service  front_tier  www.yyy.com
  NodeList  9.3.1.250 9.4.1.20  9.39.254.1
  ```

- To define a `Dispatcher` Observer:

  ```
  Dispatcher  9.3.1.250  10004
  ServiceList http_dispatcher
  ```

  The `10004` port is the default port used by the `issd` agents and `issd` monitor to communicate. You can choose another one.

Another required keyword is `HeartbeatInterval`. It defines the interval of time in seconds in which ISS checks the cluster currently ranked at the top. To indicate the number of heart beat intervals, ISS waits before determining. If an update to the `named` daemon is required, use the `HeartbeatsPerUpdate` keyword.

If you use a `RoundRobin SelectionMethod`, the requests will be rotated among the clusters of the top-tier service. The time interval in which a cluster receives all the request is determined by:

`HeartbeatsPerUpdate * HeartbeatInterval.`

Because `RoundRobin` does not measure the load on the clusters, the distribution of work will not be equal most of the time.

The `ResourceType` keyword gives the system administrator greater flexibility in balancing the load between the clusters. In this metric, you specify a command string or a shell script that will return a value indicating how loaded a cluster is.

Figure 26 on page 91 shows the configuration of an ISS working with a `NameServer` Observer. Here the `issd` monitor receives input from the `issd` agents running on the Network Dispatcher machines. The `issd` monitor machine can be the Network Dispatcher machine, a server or another machine outside the cluster.

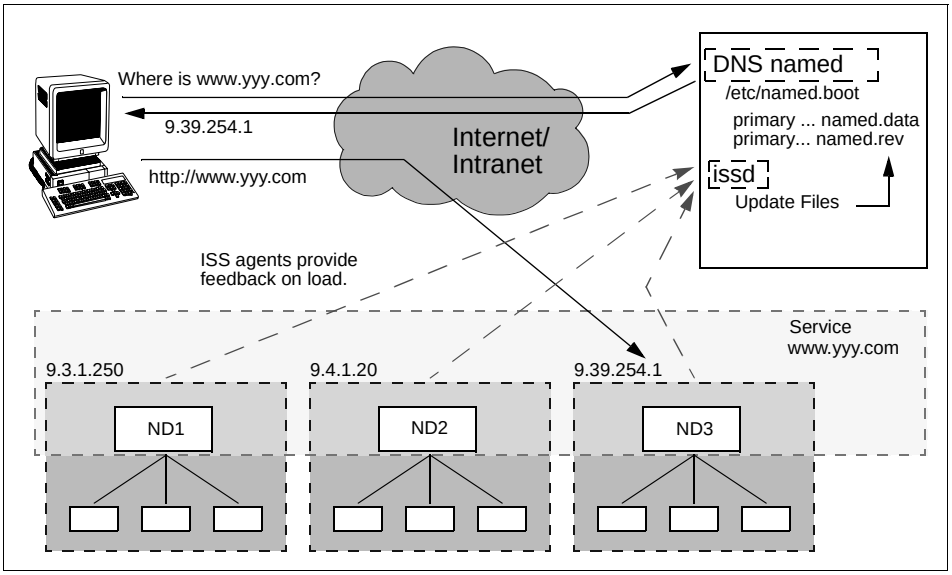


Figure 26. ISS Front-Tier Configuration

9.2 Back-End Tier Configuration

The configuration of Network Dispatcher on the back-end tier does not change in respect to a normal Network Dispatcher configuration. The only difference you have is that the `issd` agent has to run on the Network Dispatcher machine.

A Network Dispatcher machine can manage several clusters belonging to one or more services on the front-end tier. Each server can belong to one or more clusters. The cluster can be defined in terms of IP address (for example www.yyy.com and www.test.com), services offered (HTTP, FTP, and so on) or both. ISS and Network Dispatcher on a two-tier network give you the possibility of having many configuration options, and you can choose the one that best suits your needs. For example, Figure 27 on page 92 shows a two-tier configuration in which ISS manages two front-end services, each one with two Network Dispatchers. Service1 offers HTTP and FTP services for service1.www.com, and Service2 offers just FTP service for service2.www.com. ND1 belongs to Service1 and has two clusters, one for HTTP and the other for FTP. ND2 belongs to both services and has two clusters, one for HTTP and FTP in Service1 and one for FTP in Service2. ND3 has just one FTP cluster. To configure every Network Dispatcher machine, you have to follow the same steps used to configure Network Dispatcher in a

normal configuration. You must also be sure to define the same ports on the clusters that belong to the same service.

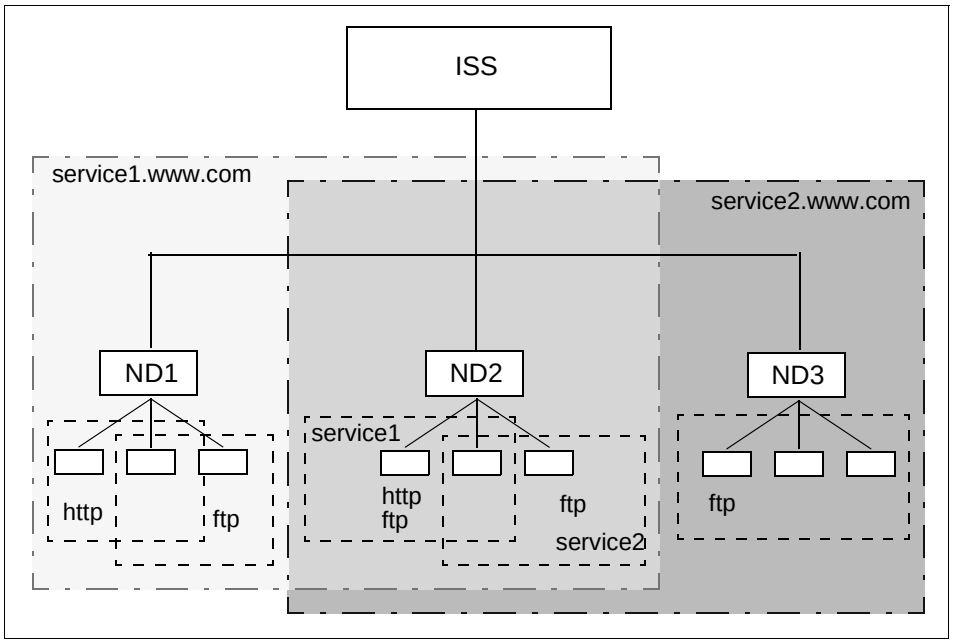


Figure 27. Different Configuration Option for Network Dispatcher and ISS

Network Dispatcher can get feedback from ISS based on a defined `ResourceType`. For example, good feedback for HTTP is CPU load; good feedback for FTP is network traffic. You have to decide which one is best for these front-end services when both services are offered.

Chapter 10. Configuration Files - Samples

This chapter presents the configuration files used for our testing, along with the parameters utilized.

10.1 The Test Environment

A 5-node RS/6000 SP was used. It had three wide nodes–nodes 1,3 and 7–and two thin nodes–nodes 5 and 6. Each node had three network interfaces:

- Token-ring: network IP address 9.3.1.0 (from 241 to 247, skipping unused node addresses) mask 255.255.255.0, connected to a private token-ring network (itsc subdomain), interface name f01n0Xt, where X is the node number (1, 3, 5, 6, and 7).
- Ethernet: SP Ethernet Network, network address 9.3.16.0 mask 255.255.255.0, interface name f01n0X.
- Switch tb2 interface (Hi-Performance Switch), network address 9.3.19.0 mask 255.255.255.0), interface name f01n0Xs.

The host name for the nodes was the name associated with the SP Ethernet adapter (f01n0X). The control workstation had two network interfaces: one token-ring (IP address 9.3.1.240 connected to a private token-ring network—itsc subdomain) and one Ethernet (SP Ethernet Network, IP address 9.3.16.240).

Figure 28 on page 94 shows the SP configuration, network topology and IP addresses used.

© Copyright IBM Corp. 1997

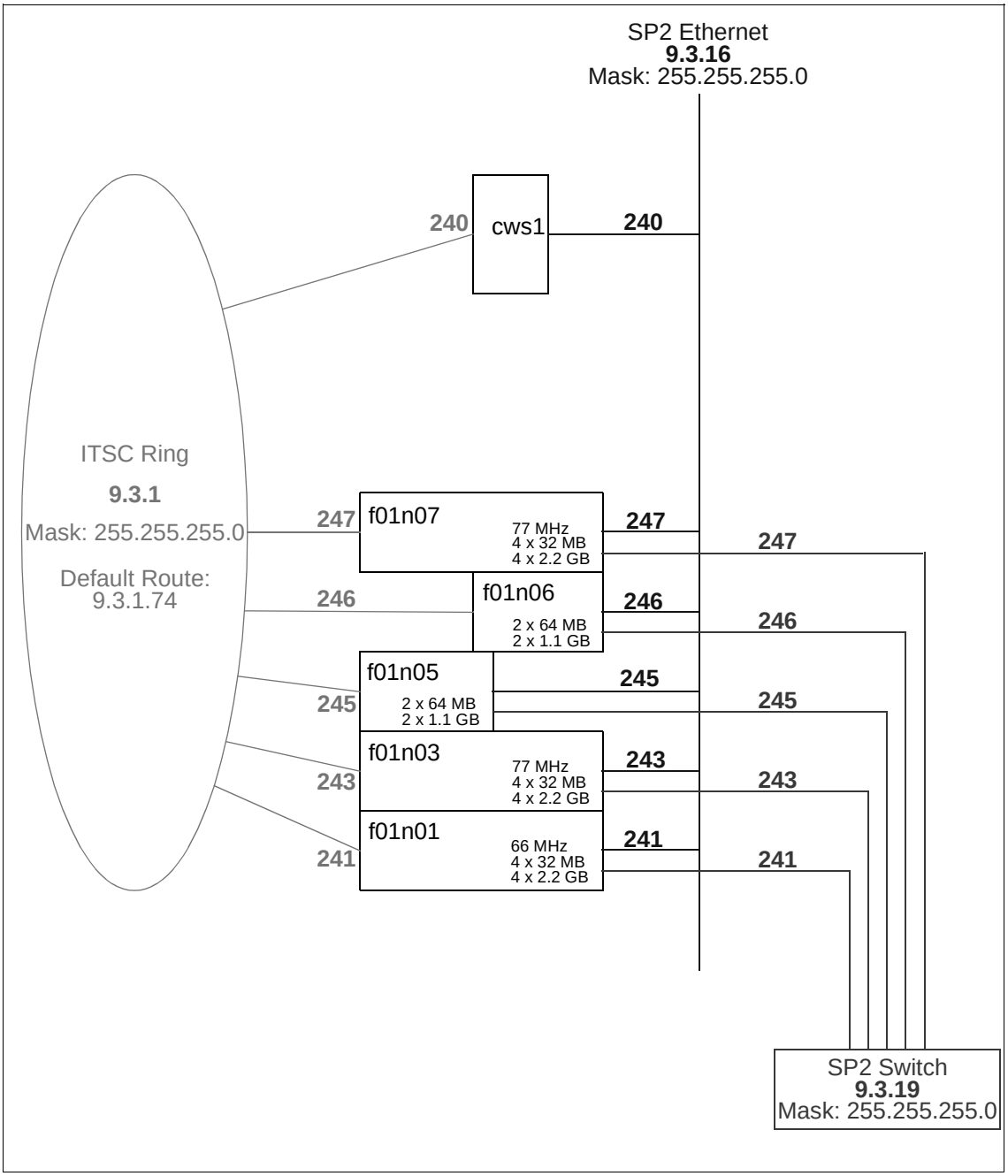


Figure 28. SP Configuration Used for Interactive Network Dispatcher Tests

A DNS domain `sp.itsc.austin.ibm.com` was used for the SP. Node 1 was the primary DNS server for this domain. The DNS configuration files used were:

- /etc/named.boot

```
;
; type          domain                  source file or host
;
domain          sp.itsc.austin.ibm.com
cache           /etc/named.cache

primary         sp.itsc.austin.ibm.com  /etc/named.data
primary         0.0.127.in-addr.arpa    /etc/named.rev.127.0.0
primary         16.3.9.in-addr.arpa     /etc/named.rev.9.3.16
primary         19.3.9.in-addr.arpa     /etc/named.rev.9.3.19
```

- /etc/named.cache

```
.                               9999999 IN    NS   castor.cdf.ibm.com.
castor.cdf.ibm.com.             9999999 IN    A    9.78.1.2
```

- /etc/named.data

```
; name server data file
; (also see /etc/named.boot)
;
; NAME          TTL     CLASS   TYPE    RDATA
;
; setting default domain to "sp.itsc.austin.ibm.com"
;
@               9999999 IN      SOA     f01n01.sp.itsc.austin.ibm.com.
root.f01n01.sp.itsc.austin.ibm.com. (
                                        992611          ; Serial
                                        3600            ; Refresh
                                        300             ; Retry
                                        3600000         ; Expire
                                        300 )           ; Minimum
                9999999 IN      NS      f01n01
;
; Control Workstation
;
cws1            9999999 IN      A       9.3.16.240
cw              9999999 IN      CNAME   cws1
;
; SP Ethernet Addresses
;
f01n01          9999999 IN      A       9.3.16.241
f01n03          9999999 IN      A       9.3.16.243
```

```
f01n05          9999999 IN      A       9.3.16.245
f01n06          9999999 IN      A       9.3.16.246
f01n07          9999999 IN      A       9.3.16.247
;
; Token Ring Addresses
;
cws1t           9999999 IN      A       9.3.1.240
f01n01t         9999999 IN      A       9.3.1.241
f01n03t         9999999 IN      A       9.3.1.243
f01n05t         9999999 IN      A       9.3.1.245
f01n06t         9999999 IN      A       9.3.1.246
f01n07t         9999999 IN      A       9.3.1.247
;
; Switch Addresses
;
f01n01s         9999999 IN      A       9.3.19.241
f01n03s         9999999 IN      A       9.3.19.243
f01n05s         9999999 IN      A       9.3.19.245
f01n06s         9999999 IN      A       9.3.19.246
f01n07s         9999999 IN      A       9.3.19.247
;
; Pool Addresses
;
pool1.sp.itsc.austin.ibm.com. IN      A       9.3.1.243
pool2.sp.itsc.austin.ibm.com. IN      A       9.3.1.250
;
; Alias for the nodes
;
n01             999999 IN       CNAME   f01n01
n03             999999 IN       CNAME   f01n03
n05             999999 IN       CNAME   f01n05
n06             999999 IN       CNAME   f01n06
n07             999999 IN       CNAME   f01n07
n01s            999999 IN       CNAME   f01n01s
n03s            999999 IN       CNAME   f01n03s
n05s            999999 IN       CNAME   f01n05s
n06s            999999 IN       CNAME   f01n06s
n07s            999999 IN       CNAME   f01n07s
n01t            999999 IN       CNAME   f01n01t
n03t            999999 IN       CNAME   f01n03t
n05t            999999 IN       CNAME   f01n05t
n06t            999999 IN       CNAME   f01n06t
n07t            999999 IN       CNAME   f01n07t
```

- /etc/named.rev.127.0.0

```
; setting default domain to ... sp.itsc.austin.ibm.com
@               9999999 IN      SOA     f01n01.sp.itsc.austin.ibm.com.
```

```
root.f01n01.sp.itsc.austin.ibm.com. (
                                        970226          ; Serial
                                        3600            ; Refresh
                                        300             ; Retry
                                        3600000         ; Expire
                                        86400 )         ; Minimum
               9999999 IN      NS      f01n01.sp.itsc.austin.ibm.com.
1       IN PTR loopback.sp.itsc.austin.ibm.com.
```

- /etc/named.rev.9.3.16

```
; setting default domain to ... sp.itsc.austin.ibm.com
@               9999999 IN      SOA     f01n01.sp.itsc.austin.ibm.com.
root.f01n01.sp.itsc.austin.ibm.com. (
                                        970226          ; Serial
                                        3600            ; Refresh
                                        300             ; Retry
                                        3600000         ; Expire
                                        300 )           ; Minimum
               9999999 IN      NS      f01n01.sp.itsc.austin.ibm.com.
240     IN PTR cws1.sp.itsc.austin.ibm.com.
241     IN PTR f01n01.sp.itsc.austin.ibm.com.
243     IN PTR f01n03.sp.itsc.austin.ibm.com.
245     IN PTR f01n05.sp.itsc.austin.ibm.com.
246     IN PTR f01n06.sp.itsc.austin.ibm.com.
247     IN PTR f01n07.sp.itsc.austin.ibm.com.
```

- /etc/named.rev.9.3.19

```
; setting default domain to ... sp.itsc.austin.ibm.com
@               9999999 IN      SOA     f01n01.sp.itsc.austin.ibm.com.
root.f01n01.sp.itsc.austin.ibm.com. (
                                        970226          ; Serial
                                        3600            ; Refresh
                                        300             ; Retry
                                        3600000         ; Expire
                                        300 )           ; Minimum
               9999999 IN      NS      f01n01.sp.itsc.austin.ibm.com.
241     IN PTR f01n01s.sp.itsc.austin.ibm.com.
243     IN PTR f01n03s.sp.itsc.austin.ibm.com.
245     IN PTR f01n05s.sp.itsc.austin.ibm.com.
246     IN PTR f01n06s.sp.itsc.austin.ibm.com.
247     IN PTR f01n07s.sp.itsc.austin.ibm.com.
```

10.2 Network Dispatcher Configuration

As you saw in Figure 28 on page 94, the tests were performed with a 5-node RS/6000 SP machine. Node 1 was configured as the Network Dispatcher

server. All the others were part of the Network Dispatcher pool. The IP address used for the pool was 9.3.1.250, and its name was `pool2.sp.itsc.austin.ibm.com`.

An alias for the token-ring interface of node 1 using the pool IP address was created by the command:

```
ifconfig tr0 alias 9.3.1.250 netmask 255.255.255.0
```

An alias for the loopback interface for all other nodes was created by the command:

```
ifconfig lo0 alias 9.31.250 netmask 255.255.255.0
```

****Configuration Hint****

One of the default ports used by Network Dispatcher (port 10003) is used by one of the SP daemons. So, if you use the Network Dispatcher server as one SP node, do not forget to change this port number by setting (and exporting) the environment variable `ND_MGR_COMMANDPORT` before starting Network Dispatcher execution.

- The following configuration file was used to perform the test in Chapter 5, “WWW Server Scenario” on page 47 and in Chapter 6, “FTP Server Scenario” on page 61.

```
#!/bin/ksh
#
# Make sure the root user is the one executing this script.
#

iam=’whoami’
if [ "$iam" != "root"]
  then
    echo "You must login as root to run this script"
    exit 2
fi

# First we must run the ndserver.
# NOTE: We include the ndserver start.

ndserver start

# First we must start Network Dispatcher

ndcontrol executor start
```

```
#
# The next step in configuring Network Dispatcher is to set the
# NFA or non-forwarding address and to set your CLUSTER address(es).
#
# The non-forwarding address is used to remotely access Network Dispatcher
# machine for whatever administration or configuration purposes. This is
# required because Network Dispatcher will forward requests sent to the
# CLUSTER address.
#
# The CLUSTER address is the hostname (or IP address) to which remote
# clients will connect.
# Anywhere in this file, you may use hostnames and IP addresses
# interchangeably.

NFA=9.3.1.241
CLUSTER=pool2.sp.itsc.austin.ibm.com

echo "Loading the non-forwarding address"
ndcontrol executor set nfa $NFA

# The next step in configuring Network Dispatcher is to create a
# CLUSTER. Network Dispatcher will route the requests sent to the
# CLUSTER address to the corresponding TCP server machines defined
# for that CLUSTER. You can configure and serve multiple CLUSTER
# addresses using Network Dispatcher. Use a similar configuration
# for CLUSTER2, CLUSTER3, etc.

echo "Loading first CLUSTER address "
ndcontrol cluster add $CLUSTER

echo "Creating ports for CLUSTER: $CLUSTER"

ndcontrol port add $CLUSTER:80
ndcontrol port add $CLUSTER:20
ndcontrol port add $CLUSTER:21

# The next step is to add each of the TCP server machines to the ports in
# this CLUSTER.

SERVER1=f01n03t.sp.itsc.austin.ibm.com
SERVER2=f01n05t.sp.itsc.austin.ibm.com
SERVER3=f01n06t.sp.itsc.austin.ibm.com
SERVER4=f01n07t.sp.itsc.austin.ibm.com

echo "Adding TCP server machines"

#
```

```
# server1
#

ndcontrol server add $CLUSTER:80:$SERVER1
ndcontrol server add $CLUSTER:20:$SERVER1
ndcontrol server add $CLUSTER:21:$SERVER1

#
# server2
#

ndcontrol server add $CLUSTER:80:$SERVER2
ndcontrol server add $CLUSTER:20:$SERVER2
ndcontrol server add $CLUSTER:21:$SERVER2

#
# server3
#

ndcontrol server add $CLUSTER:80:$SERVER3
ndcontrol server add $CLUSTER:20:$SERVER3
ndcontrol server add $CLUSTER:21:$SERVER3

#
# server4
#

ndcontrol server add $CLUSTER:80:$SERVER4
ndcontrol server add $CLUSTER:20:$SERVER4
ndcontrol server add $CLUSTER:21:$SERVER4

#
# Now we will start the load-balancing components of Network Dispatcher.
# The main load-balancing component is called the Manager. The Manager
# will update the weight of each of the TCP server machines based on
# three policies:
#   1. The number of active connections on each TCP server
#   2. The number of new connections for each TCP server
#   3. Input from TCP server Advisors
#   4. Input from the system level Advisor (iss)
# Each of these policies can be given a weight or importance.

echo "Starting the manager..."
ndcontrol manager start

# give some time for the manager to load
```

```
sleep 4

# The second load-balancing component of NetworkDispatcher is the Advisors.
# The Advisors will give the Manager more insight into the TCP server's
# ability to serve TCP requests (such as HTTP).Currently,NetworkDispatcher
# comes with an HTTP Advisor, an FTP Advisor, and an SSL Advisor.

echo "Starting the http advisor on port 80..."
ndcontrol advisor start http 80
echo "Starting the http advisor on port 21..."
ndcontrol advisor start http 21

# The last step is to set the proportions for each of the policies.
#
# These proportions should be proportioned to add up to 100
# This will give the number of active connections, new
# connections, the advisor and system input the same importance in the
# weight decision.
#
# NOTE: By default the proportions are 50 50 0 0
# Set some parameters for the manager

echo "Setting the proportions..."
ndcontrol manager proportions 50 50 0 0

echo "Setting the smoothing value for the manager..."
ndcontrol manager smoothing 2.5
```

10.3 ISS Configuration

In this section, the ISS configuration files for the `RoundRobin` and `Best SelectionMethod` are shown.

10.3.1 RoundRobin

We used the same SP environment for these tests. Node 1 is for the DNS name server, and it also acts as an ISS monitor and agent.

- The following configuration file was used to perform the test in Chapter 5, "WWW Server Scenario" on page 47 and in Chapter 6, "FTP Server Scenario" on page 61.

```
# This ISS configuration file illustrates how ISS can be used with a DNS
# name server to perform load balancing for one cluster of servers using
# the RoundRobin metric.
#
# Basic parameters of AustinTx1 cell.
```

```
Cell                AustinTx1 local
Authkey             036454DC 65739CB2 23C68DF1 129C4D61
LogLevel            Debug
HeartbeatInterval   30
HeartbeatsPerUpdate 2
PortNumber          11364

# Nodes in AustinTx1 cell

# The node monitor1 has ISS monitor priority 1 and will run the issd
# process in ISS monitor mode. The node monitor2 has monitor priority 2
# and can run as a backup ISS monitor when monitor1 is down. The other
# nodes are declared NotMonitor. The issd processes on these nodes will
# run as ISS agents or perform other special ISS functions. They cannot
# assume the role of an ISS monitor.

Node        f01n01t  1
InternalNet f01n01
Node        f01n03t  2
NotMonitor
InternalNet f01n03
Node        f01n05t  3
NotMonitor
InternalNet f01n05
Node        f01n06t  11
NotMonitor
InternalNet f01n06
Node        f01n07t  12
NotMonitor
InternalNet f01n07

# Definition of "ProcessCount" ResourceType

# The metric associated with ProcessCount is assigned a range of (0-250).
# If "ps -fe | wc -l" returns a value greater than 250, the metric is
# assigned the upper limit of 250. When the value of this metric on a node
# is greater than 225, the node is removed from active participation
# in the Service that it is associated with. The node is returned to active
# participation when the metric has a value less than or equal to 200.

ResourceType        ProcessCount
Metric              External   ps -ef | wc -l
MetricNormalization 0   250
MetricLimits        200 225
Policy              Min
```

```
# Definition of Services available in AustinTx1 cell.

# The first service has the DNS name pool1.sp.itsc.austin.ibm.com
# associated with it. Also associated with service are the nodes:f01n03t,
# f01n05t,f01n06t,f01n07t. Since the observer is NameServer, ISS will work
# in conjunction with the DNS name server running on the node f01n01t to
# resolve the DNS name pool1.sp.itsc.austin.ibm.com to the IP address of
# either f01n03t,f01n05t,f01n06t,f01n07t. Since the SelectionMethod is
# RoundRobin, this mapping of the name pool1 to one of the four IP
# addresses of the name pool1 to one of the three IP addresses is done using
# a simple round-robin algorithm. A new address is selected
# once every 60 seconds (HeartbeatInterval*HeartbeatsPerUpdate) and once
# every 20 seconds (HeartbeatInterval) ISS checks that the server whose
# address is currently functional at its ICMP layer. If at
# any time there are no servers available, the name pool1 will be removed
# from the DNS files and the alarm triggered, which in this case makes and
# entry into a file called /tmp/issalarm.log.
#

Service              service  pool1.sp.itsc.austin.ibm.com
NodeList             f01n03t
                     f01n05t
                     f01n06t
                     f01n07t
ResourceList         ProcessCount
SelectionMethod      RoundRobin
Alarm  echo "pool1 ran out of servers at ‘date‘" >> /tmp/issalarm.log

# Defining a NameServer Observer

# The NameServer observer running on the node f01n01t subscribes all four
# services defined in this file. ISS will work with the DNS name server
# running on f01n01t to load balance the cluster of servers.

NameServer           f01n01t   53
ServiceList          service
NamedData            /etc/named.data
NamedRev             /etc/named.rev.9.3.1
```

10.3.2 Best - SelectionMethod

For the `Best SelectMethod` tests, we used the same SP environment we accessed before. Node 1 is for the DNS name server, and it also acts as an ISS monitor and agent.

- The following configuration files was used to perform the test in Chapter 5, “WWW Server Scenario” on page 47.

```
# This ISS configuration file illustrates how ISS can be used with a DNS
# name server to perform load balancing for three clusters of servers. A
# different load balancing method (RoundRobin, Custom, and LoadLeveler) is
# used for each cluster.

# Basic parameters of AustinTx1 cell.

Cell                AustinTx1 local
Authkey             036454DC 65739CB2 23C68DF1 129C4D61
LogLevel            Debug
HeartbeatInterval   15
HeartbeatsPerUpdate 2
PortNumber          11364

# Nodes in AustinTx1 cell

# The node monitor1 has ISS monitor priority 1 and will run the issd
# process in ISS monitor mode. The node monitor2 has monitor priority 2
# and can run as a backup ISS monitor when monitor1 is down. The other
# nodes are declared NotMonitor. The issd processes on these nodes will
# run as ISS agents or perform other special ISS functions. They can not
# assume the role of an ISS monitor.

Node        f01n01t  1
InternalNet f01n01
Node        f01n03t  2
NotMonitor
InternalNet f01n03
Node        f01n05t  3
NotMonitor
InternalNet f01n05
Node        f01n06t  11
NotMonitor
InternalNet f01n06
Node        f01n07t  12
NotMonitor
InternalNet f01n07

# Definition of "CPUIdle3" ResourceType

# On AIX, "vmstat 3 2 | tail -1 | awk '{print $16}'" returns the percentage
# of CPU Idle time during a three-second sampling period. This value should
# be maximized when you want to identify the least loaded server.
# (On Solaris, use: "vmstat 3 2 | tail -1 | awk '{print $22}'").
# When the value of this metric on a node is less than 20, the node
# is removed from active participation in the Service that it is associated
# with. The node is returned to active participation when the metric has a
```

```
# value greater than or equal to 60.

ResourceType          CPUIdle
Metric                External   vmstat 3 2 | tail -1 | awk '{print $16}'
MetricNormalization   0   100
MetricLimits          60  30
Policy                Max

# Definition of Services available in AustinTx1 cell.

# The first service, has the DNS name pool1.sp.itsc.austin.ibm.com
# associated with it. Also associated with service are the nodes:f01n03t,
# f01n05t,f01n06t,f01n07t. Since the observer is NameServer, ISS will work
# in conjunction with the DNS name server running on the node f01n01t to
# resolve the DNS name pool1.sp.itsc.austin.ibm.com to the IP address of
# either f01n03t,f01n05t,f01n06t,f01n07t.Since the SelectionMethod is
# RoundRobin,this mapping is done using a simple round-robin algorithm.
# A new address is selected once every 60 seconds
# (HeartbeatInterval*HeartbeatsPerUpdate) and once
# every 20 seconds (HeartbeatInterval) ISS checks that the server whose
# address is currently functional at its ICMP layer. If at
# any time there are no servers available, the name pool1 will be removed
# from the DNS files and the alarm triggered, which, in this case, makes
# an entry into a file called /tmp/issalarm.log.
#

Service               service  pool1.sp.itsc.austin.ibm.com
NodeList              f01n07t
                      f01n06t
                      f01n05t
                      f01n03t
ResourceList          CPUIdle
SelectionMethod       Best
Alarm  echo "pool1 ran out of servers at ‘date‘" >> /tmp/issalarm.log

# Defining a NameServer Observer
#
# The NameServer observer running on the node f01n01t subscribes all four
# services defined in this file. ISS will work with the DNS name server
# running on f01n01t to load balance the cluster of servers.

NameServer            f01n01t   53
ServiceList           service
NamedData             /etc/named.data
NamedRev              /etc/named.rev.9.3.1
```

10.4 High Availability Configuration

This section shows the configuration script of the high availability function. The result of this test is in Chapter 8, “High Availability Server Scenario” on page 75.

```
#!/bin/ksh
#
# This is the Sample configuration file for Interactive Network Dispatcher
# for High Availability.

#
# Make sure the root user is the one executing this script.
#

iam=‘whoami‘

if [ "$iam" != "root" ]
  then
   echo "You must login as root to run this script"
   exit 2
fi

# First we must run the Ndserver

ndserver start

# We must start Interactive Network Dispatcher

ndcontrol executor start

# The next step in configuring Interactive Network Dispatcher is to set the
# NFA (non-forwarding address) and to set the cluster address(es).
#
# The non-forwarding address is used to remotely access the Interactive
# Network Dispatcher machine for administration or configuration purposes.
# This address is required since Interactive Network Dispatcher will
# forward packets to the cluster address.
#
# The CLUSTER address is the hostname (or IP address) to which remote
# clients will connect.
#
# Anywhere in this file, you may use hostnames and IP addresses
# interchangeably.
# NFA=[non-forwarding address]
# CLUSTER=[your clustername]
```

```
NFA=9.3.1.241
CLUSTER=9.3.1.250

echo "Loading the non-forwarding address"
ndcontrol executor set nfa $NFA

# The next step in configuring Interactive Network Dispatcher is to create
# a CLUSTER. Interactive Network Dispatcher will route requests sent to the
# CLUSTER address to the corresponding server machines defined to that
# CLUSTER. You may configure and serve multiple CLUSTER addresses using
# Interactive Network Dispatcher.
# Use a similar configuration for CLUSTER2, CLUSTER3, etc.

echo "Loading first CLUSTER address "
ndcontrol cluster add $CLUSTER

# Now we must define the ports this CLUSTER will use.
# The most common ports used are 80 for HTTP and 20 and 21 for FTP.
# Any request received by Interactive Network Dispatcher on one of these
# ports will be forwarded to the corresponding port of one of the server
# machines.

echo "Creating ports for CLUSTER: $CLUSTER"

ndcontrol port add $CLUSTER:20
ndcontrol port add $CLUSTER:21
ndcontrol port add $CLUSTER:80

# The last step is to add each of the server machines to the ports in
# this CLUSTER. Again, you can use either the hostname or the IP address of
# the server machines.

SERVER1=f01n05t
SERVER2=f01n06t
SERVER3=f01n07t

echo "Adding server machines"

#
# f01n05t
#

ndcontrol server add $CLUSTER:20:$SERVER1
ndcontrol server add $CLUSTER:21:$SERVER1
ndcontrol server add $CLUSTER:80:$SERVER1

#
```

```
# f01n06t
#

ndcontrol server add $CLUSTER:20:$SERVER2
ndcontrol server add $CLUSTER:21:$SERVER2
ndcontrol server add $CLUSTER:80:$SERVER2

#
# f01n07t
#

ndcontrol server add $CLUSTER:20:$SERVER3
ndcontrol server add $CLUSTER:21:$SERVER3
ndcontrol server add $CLUSTER:80:$SERVER3

# We will now start the load balancing components of Interactive Network
# Dispatcher. The main load balancing component is called the manager. The
# The manager will update the weight of each of the server machines based
# on four policies
#   1. The number of active connections on each TCP server
#   2. The number of new connections for each TCP server
#   3. Input from TCP server advisors.
#   4. Input from the system level advisor (ISS).
# Each of these policies can be given a weight or importance.

echo "Starting the manager..."
ndcontrol manager start

# The second load balancing component of Interactive Network Dispatcher is
# the advisors.The advisors give the manager more insight into the server's
# ability to serve TCP requests (such as HTTP). Interactive Network
# Dispatcher has advisors for FTP, Telnet, SMTP, HTTP, POP3, NNTP and SSL.

echo "Starting the FTP advisor on port 21 ..."
ndcontrol advisor start ftp 21
echo "Starting the HTTP advisor on port 80 ..."
ndcontrol advisor start http 80

# The last step is to set the proportions for each of the policies.
# For example:
#   ndcontrol manager proportions 15 15 35 35
#
# These proportions must be proportioned to add up to 100.
# This will give both the number of active connections and new
# connections 15% input in the weighting process, while the advisor
# and system inputs will contribute 35% each to the weight decision.
#
```

```
# NOTE: By default the proportions are 50 50 0 0

echo "Setting the proportions..."
ndcontrol manager proportions 50 50 0 0
echo "Setting the smoothing value for the manager..."
ndcontrol manager smoothing 1.0

# The following commands are set to the default values.
# Use these commands to guide to change from the defaults.

#  ndcontrol manager loglevel    1
#  ndcontrol manager logsize     unlimited
#  ndcontrol manager sensitivity 5.000000
#  ndcontrol manager interval    2
#  ndcontrol manager refresh     2

#  ndcontrol advisor interval ftp  21  5
#  ndcontrol advisor loglevel ftp  21  1
#  ndcontrol advisor logsize  ftp  21  unlimited
#  ndcontrol advisor timeout  ftp  21  unlimited
#  ndcontrol advisor interval telnet 23 5
#  ndcontrol advisor loglevel telnet 23 1
#
# These proportions must be proportioned to add up to 100.
# This will give both the number of active connections and new
# connections 15% input in the weighting process, while the advisor
# and system inputs will contribute 35% each to the weight decision.
#
# NOTE: By default the proportions are 50 50 0 0

echo "Setting the proportions..."
ndcontrol manager proportions 50 50 0 0
echo "Setting the smoothing value for the manager..."
ndcontrol manager smoothing 1.0
```

Appendix A. Minimal Configuration

Sometimes it may be necessary to test or to demonstrate ISS and Network Dispatcher in a restricted environment, with very few machines. This appendix shows the minimal configuration needed to do that.

If you need further information, you may refer to Chapter 10, “Configuration Files - Samples” on page 93, and to Chapter 4, “Installation and Operation” on page 39.

A.1 ISS

For ISS, the minimal configuration depends on how you want ISS to balance the load on the servers. This session focuses on the use of ISS using the `RoundRobin` metric and custom metric configurations.

A.1.1 RoundRobin Metrics

When using ISS with the `RoundRobin` metric, it is possible to use just one machine with one network interface to implement an ISS server. Using this implementation, the ISS server machine will also be the pool. To do this, it is necessary to create several aliases for the existing network interfaces. These aliases will be the pool ISS will use. An example of this configuration is shown on Figure 29.

It is preferable that all of the alias addresses are on the same network as the main network interface address. If this is not the case, it is necessary to create routes pointing to the main network interface as the route for addresses on the other networks.

© Copyright IBM Corp. 1997

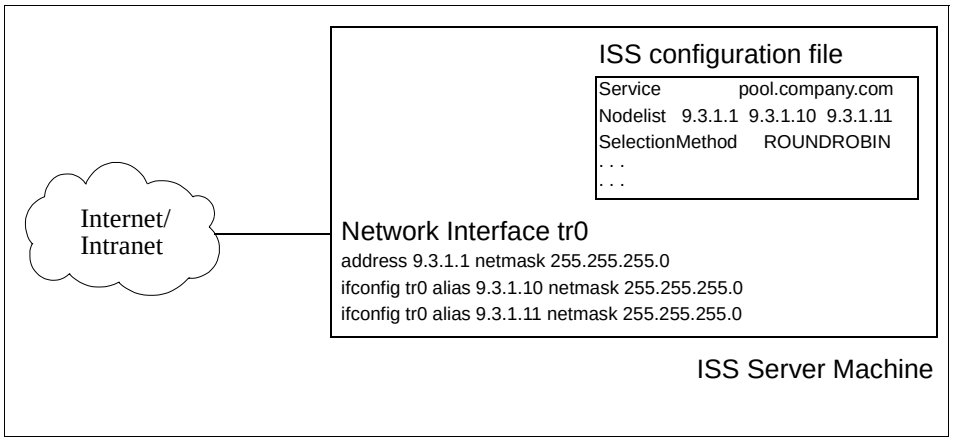


Figure 29. Example of Minimum Configuration for ISS in Round-Robin

A.1.2 Custom Metrics

For custom metrics, this one-machine configuration is not possible because ISS tries to start an instance of ISS for each machine in the pool. Since all the addresses of the pool would be in just one machine, the ISS (monitor) would try to start several instances of ISS on the same server, which is not possible.

So, the minimum configuration that can be used for ISS with a custom metric is a two-machine configuration, each with one network interface. One machine would be the ISS server and a part of the pool of two machines. This configuration was used for the basic tests with ISS. See 4.1, “Interactive Network Dispatcher User’s Guide” on page 39.

A.2 Network Dispatcher

For Network Dispatcher, the minimum configuration is also two servers, each of which has one network interface.

When we tried to test using just one machine with several network interface aliases, the execution of the configuration script was fine. But when we tried to access the pool address and the selected server was the alias address, the machine hung, and a reboot was necessary.

The configuration of these two machines with one network interface each was used for the basic tests with Network Dispatcher, and the document containing the configuration is described in 4.1, “Interactive Network Dispatcher User’s Guide” on page 39.

Appendix B. Domain Name System (DNS)

Every machine in a network needs a name and a address. Although 32-bit Internet addresses provide machines an efficient means of identifying the source and destination of datagrams sent across a network, users prefer meaningful, easily remembered names. TCP/IP provides a naming system that supports both flat and hierarchical network organizations.

This appendix explains some key elements to understand how DNS–the Domain Name System–works.

B.1 Naming

Naming in flat networks is very simple: Host names consist of a single set of characters and generally are administered locally. In flat TCP/IP networks, each machine on the network has a file (/etc/hosts) containing the name-to-Internet-address mapping information for every host on the network. As a TCP/IP network grows, the administrative duty of keeping each machine's naming file up-to-date grows. When TCP/IP networks become very large, as in the Internet, naming is divided hierarchically. Typically, the divisions follow the network's organization. In TCP/IP, hierarchical naming is known as the domain name system (DNS), and it uses the Domain protocol. The Domain protocol is implemented by the `named` daemon in TCP/IP.

As in naming for flat networks, the domain name hierarchy provides for the assignment of symbolic names to networks and hosts that are meaningful and easy for users to remember. However, instead of each machine on the network keeping a file containing the name-to-address mapping for all other hosts on the network, one or more hosts are selected to function as name servers. Name servers translate (resolve) symbolic names assigned to networks and hosts into the efficient Internet addresses used by machines. A name server has complete information about some part of the domain, referred to as a zone, and it has authority for its zone.

B.2 Naming Authority

In a flat network, all hosts in the network are administered by a central authority. This form of network requires that all hosts in the network have unique host names. In a large network, this requirement creates a large administrative burden on the central authority.

© Copyright IBM Corp. 1997

In a domain network, groups of hosts are administered separately within a tree-structured hierarchy of domains and subdomains. In this case, host names need to be unique only within the local domain, and only the root domain is administered by a central authority. This structure allows subdomains to be administered locally and reduces the burden on the central authority. For example, the root domain of the Internet consists of such domains as COM (commercial organizations), EDU (educational organizations), GOV (governmental organizations), and MIL (military groups). New top-level domains can only be added by the central authority. Naming at the second level is delegated to designated agents within the respective domains. An example of a domain structure is shown in Figure 30.

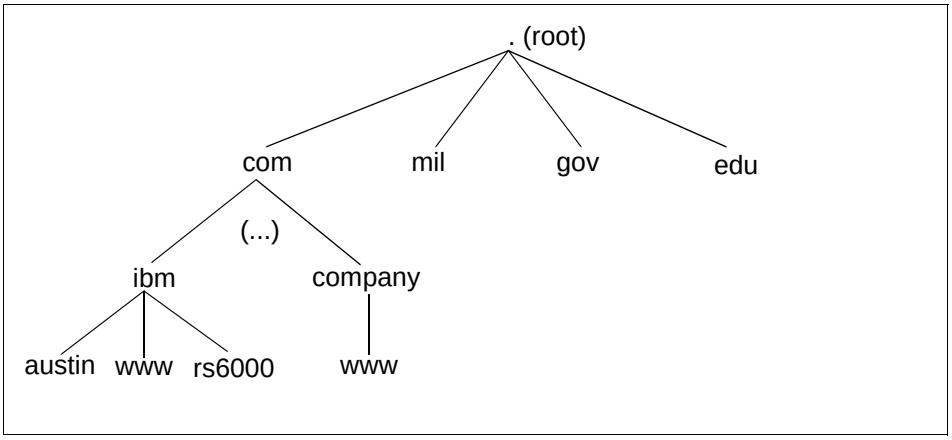


Figure 30. Example of Domain Structure

B.3 Naming Conventions

In the hierarchical domain name system, names consist of a sequence of case-insensitive subnames separated by periods, with no embedded blanks. The Domain protocol specifies that a local domain name must be less than 64 characters and that a host name must be less than 32 characters in length. The host name is given first, followed by a . (period), a series of local domain names separated by periods, and finally the root domain. A fully specified domain name for a host, including periods, must be less than 255 characters in length and in the following form:

host.subdomain1.[subdomain2 . . . subdomainN].rootdomain

Since host names must be unique within a domain, you can use an abbreviated name when sending messages to a host within the same domain.

For example, instead of sending a message to `smith.eng.lsu.edu`, a host in the `eng.lsu.edu` domain could send a message to `smith`. Additionally, each host can have several aliases that other hosts can use when sending messages.

B.4 Name Servers

In a flat name space, all names must be kept in the /etc/hosts file on each host on the network. If the network is very large, this can become a burden on the resources of each machine.

In a hierarchical network, certain hosts designated as name servers resolve names into Internet addresses for other hosts. This has two advantages over the flat name space: It keeps the resources of each host on the network from being tied up in resolving names, and it keeps the person who manages the system from having to maintain name resolution files on each machine on the network. The set of names managed by a single name server is known as its *zone of authority*.

Note: Although the host machine that performs the name-resolution function for a zone of authority is commonly referred to as a *name server host*, the process controlling the function, the `named` daemon, is the actual name server process.

To further reduce unnecessary network activity, all name servers cache (store for a period of time) name-to-address mappings. When a client asks a server to resolve a name, the server checks its cache first to see if the name has been resolved recently. Since domain and host names do change, each item remains in the cache for a limited length of time specified by the record's time to live (TTL). In this way, authorities can specify how long they expect the name resolution to be accurate.

Within any autonomous system, there can be multiple name servers. Typically, name servers are organized hierarchically and correspond to the network's organization. Each domain might have a name server responsible for all subdomains within the domain. Each subdomain name server communicates with the name server of the domain above it (called the *parent name server*), as well as with the name servers of other subdomains.

There are several types of name servers:

- Primary Name Server: loads its data from a file or disk and may delegate authority to other servers in its domain.
- Secondary Name Server: receives its information at system startup time for the given zone of authority from a primary name server, and then periodically asks the primary server to update its information. On expiration of the *refresh* value in the start of authority (SOA) resource record on a secondary name server, the secondary will reload the database from the primary if the serial number of the database on the primary is greater than the serial number in the current database on the secondary. To force a new zone transfer from the primary, simply remove the existing secondary databases and refresh the `named` daemon on the secondary name server.
- Caching-Only Server: indicates that the caching-only server responds to queries by asking other servers that have the authority to provide the information needed if a caching-only server does not have a name-to-address mapping in its cache.
- Forwarder or Client Server: forwards queries it cannot satisfy locally to a fixed list of forwarding servers. Slave forwarders (a forwarder which obtains information and passes it on to other clients, but which is not actually a server) will not interact with the primary name servers for the root domain and other domains. The queries to the forwarding servers are recursive. There may be one or more forwarding servers, which are tried sequentially until the list is exhausted. A client and forwarder configuration is typically used when you do not wish all the servers at a given site to interact with the rest of the Internet servers.
- Remote Server: runs all the network programs that use the name server without the name server process running on the local host. All queries are serviced by a name server that is running on another machine on the network.

One name server host can perform in different capacities for different zones of authority. For example, a single name-server host can be a primary name server for one zone and a secondary name server for another zone.

B.5 Planning for Domain Name Resolution

If you are part of a large network, you need to coordinate setting up your domain and name servers with their central authority.

Some hints in planning your own domain name resolution system:

- Plan ahead.
- Changing a name is much more difficult than setting up the initial one. Obtain consensus from your organization on network, gateway, name server, and host names before you set up your files.
- Set up redundant name servers.
- If you cannot set up redundant name servers, be sure to set up secondary and cache name servers so you have some type of backup.

B.6 Configuring Name Servers

In a hierarchical network, certain hosts are designated as name servers. These hosts resolve names into Internet addresses for other hosts. The `named` daemon controls the name server function and, therefore, must be run on a name server host.

Before you configure a name server, decide which type or types best fit the network it will serve. There are several types of name servers.

A primary name server actually stores the database containing name-to-address mapping information. It loads its data from a file or disk and may delegate authority to other servers in its domain. A secondary name server receives its information at system startup time for a given zone of authority from a primary name server, and then periodically asks the primary server to update its information. A caching-only server responds to requests to resolve names by querying other servers that have the authority to provide the information needed.

Keep in mind that a name server may function in different capacities for different *zones of authority*. For example, one name server host can be a primary name server for one zone and a secondary name server for another zone. If your system has the Network Information Service (NIS) application installed, NIS can also provide name server service.

There are several files involved in configuring name servers.

- boot: This file is read when the named daemon starts. The records in the boot file tell the `named` daemon which type of server it is, which domains it has authority over (its zones of authority), and where to get the data for initially setting up its database. The default name of this file is /etc/named.boot. However, you can change the name of this file by specifying the name and path of the file on the command line when the `named` daemon is started. When using the /etc/named.boot as the boot file, if it does not exist, a message is generated in syslog and `named` terminates.

However, if an alternative boot file is specified, and the alternative file does not exist, no error message will be generated and named will continue.

- cache: contains information about the local cache. The local cache file contains the names and addresses of the highest authority name servers in the network. The cache file uses the standard resource record format. The name of the cache file is set in the boot file.
- domain data: there are three domain data files, also referred to as the named *data* files. The named *local* file contains the address resolution information for local loopback. The named *data* file contains the address resolution data for all machines in the name server's zone of authority. The named *reverse data* file contains the reverse address-resolution information for all machines in the name server's zone of authority. The domain data files use the standard resource record format. Their names are user-definable and are set in the boot file. By convention, the names of these files generally include the name of the daemon (named), and in the extension, the type of file and name of the domain. For example, the name server for the domain abc might have the following files: named.abcdata, named.abcrev and named.abclocal. When modifying the named data files the serial number in the SOA resource record should be increased.
- resolv.conf: The presence of this file indicates to a host that it should go to a name server to resolve a name first. If the resolv.conf file does not exist, the host looks in the /etc/hosts file for name resolution. On a name server, the resolv.conf file must exist and can contain the local host's address, the loopback address (127.0.0.1), or be empty.

Time-to-live (TTL) is specified in resource records. If TTL is not specified in a record, the length of this time period defaults to the minimum field as defined in the start of authority (SOA) record for that zone. TTL is used when data is stored outside a zone (in a cache) to ensure that the data does not stay around indefinitely.

B.6.1 Configuring a Primary Name Server

The following steps describe how to configure a primary name server by editing a series of files and then using the System Management Interface Tool (SMIT) or the command line to start the named daemon.

1. Edit the /etc/named.boot file. If there is no named.boot file in the /etc directory, copy the /usr/samples/tcpip/named.boot sample file into the /etc directory and edit it. This file is read each time the named daemon starts. It tells the server which type of server it is, the zone for which it is responsible, and where to get its initial information.

- Specify the directory in which the named data files can be located (optional). Use this line if you want the named data files to use paths relative to this directory. For example:

  ```
  directory /usr/local/domain
  ```

- Define the name of the default domain for the name server. For example:

  ```
  domain abc.aus.century.com
  ```

- Optionally, specify the name of the cache file. For example:

  ```
  cache . /etc/named.ca
  ```

- Specify a name server type of primary, and define the names of the named hosts data file and named reverse hosts data file. For example:

  ```
  primary abc.aus.century.com    /etc/named.abcdata
  primary 201.9.192.in-addr.arpa /etc/named.abcrev
  ```

  Note: Include lines for each zone for which the name server is primary.

- Define the name of the named local file. For example:

  ```
  primary 0.0.127.in-addr.arpa /etc/named.local
  ```

2. Edit the /etc/named.ca file. This file contains the addresses of the servers that are authoritative or that are root name servers for the domain. For example:

   ```
   ; root name servers.
   .                          IN NS relay.century.com.
   relay.century.com. 3600000 IN A 129.114.1.2
   ```

   Note: All lines in this file must be in standard resource record format.

3. Edit the /etc/named.local file.

   - Specify the start of authority of the zone and the default time-to-live information. For example:

     ```
     @ IN SOA venus.abc.aus.century.com. gail.zeus.abc.aus.century.com.
             ( 1.1 ;serial
              3600 ;refresh
               600 ;retry
           3600000 ;expire
             86400);minimum
     ```

   - Specify the name server (NS) record. For example:

     ```
     IN NS venus.abc.aus.century.com.
     ```

   - Specify the pointer (PTR) record.

     ```
     1 IN PTR localhost.
     ```

Note: All lines in this file must be in standard resource record format.

4. Edit the /etc/named.data file. The /usr/samples/tcpip/hosts.awk file contains directions for creating the /etc/named.data file. Use the /usr/samples/tcpip/named.data sample file as an example when creating the /etc/named.data file.

   - Specify the start of authority of the zone and the default time-to-live information for the zone. This record designates the start of a zone. There should only be one start of authority record per zone. For example:

     ```
     @ IN SOA venus bob.robert.abc.aus.century.com.
             ( 1.1 ;serial
              3600 ;refresh
               600 ;retry
           3600000 ;expire
            86400) ;minimum
     ```

   - Include name-to-address resolution information on all hosts in the name server's zone of authority. For example:

     ```
     venus  IN A 192.9.201.1
     kronos IN A 192.9.201.2
     mars   IN A 192.9.201.3
     ```

   - Include name server records for all primary name servers in the zone. For example:

     ```
     IN NS venus.abc.century.com
     IN NS kronos.xyz.century.com
     ```

   - Include other types of entries, such as canonical name records and mail exchanger records as needed.

     Note: All lines in this file must be in standard resource record format.

5. Edit the /etc/named.rev file. The /usr/samples/tcpip/addrs.awk file contains directions for creating the /etc/named.rev file.

   - Specify the start of authority (SOA) of the zone and the default time-to-live information. This record designates the start of a zone. There should only be one start of authority (SOA) record per zone. For example:

     ```
     @ IN SOA venus.abc.aus.century.com. bob.robert.abc.aus.century.com.
             ( 1.1 ;serial
              3600 ;refresh
               600 ;retry
           3600000 ;expire
            86400) ;minimum
     ```

- Include address to name resolution information on all hosts to be in the name server's zone of authority.

  ```
  ;ABC.AUS.CENTURY.COM Hosts
  1 IN PTR venus.abc.aus.century.com.
  2 IN PTR kronos.abc.aus.century.com.
  3 IN PTR mars.abc.aus.century.com.
  ```

- Include other types of entries, such as name server records and canonical name records (optional).
- Note: All lines in this file must be in standard resource record format.

6. Create an /etc/resolv.conf file by issuing the following command:

   ```
   touch /etc/resolv.conf
   ```

   The presence of this file indicates that the host should use a name server, not the /etc/hosts file, for name resolution. This file must exist on a name server host and may contain either the local host's address, the loopback address (127.0.0.1) or be empty.

   Alternatively, the /etc/resolv.conf file may contain the following entry:

   ```
   nameserver 127.0.0.1
   ```

   The 127.0.0.1 address is the loopback address, which causes the host to access itself as the name server. The /etc/resolv.conf file may also contain an entry like the following:

   ```
   domain domainname
   ```

   In the previous example, `domainname` would be austin.century.com.

7. Perform one of the following steps:

   - Enable the `named` daemon using the following SMIT fastpath:

     ```
     smit stnamed
     ```

     This initializes the daemon with each system startup. Indicate whether you want to start the `named` daemon now, at the next system restart, or both.

   - Edit the /etc/rc.tcpip file. Un-comment the line for the `named` daemon by removing the comment (`#`) symbol from the following line:

     ```
     #start /etc/named "$src_running"
     ```

     This initializes the daemon with each system startup.

8. If you chose not to initialize the `named` daemon through SMIT, start the daemon for this session by issuing the following command:

   ```
   startsrc -s named
   ```

B.7 Delegating Domains to Other Name Servers

To delegate a subdomain to other name servers for normal DNS queries (hostname to IP address conversion), do the following:

1. Create all the data files for the subdomain in the primary name server for the new domain as explained in item B.6.1, “Configuring a Primary Name Server” on page 120. If it is desired for the clients to be able to resolve names outside their domain, don’t forget to include a cache and/or forwarders line in the named boot file.
2. Start the name server daemon on the primary name server for the delegated domain, and test it to see if everything is working correctly.
3. If necessary, set up a secondary name server.
4. Include in the named data file of the main name server for the domain a NS record indicating that queries for the subdomain should be handled by other name servers.

   For example, if your domain is company.com and you wish to create a new subdomain, dept.company.com, and the name server for this domain should be machine.dept.company.com, the following lines must be added to the named data file (/etc/named.data) on the main DNS server for company.com:

   ```
   dept.company.com.          9999999 IN  NS machine.dept.company.com.
   machine.dept.company.com. 9999999 IN  A   10.0.1.100
   ```

5. Refresh the named daemon to the changes in the data files become effective.

In the case of reverse DNS (IP address to hostname conversion), there are two cases:

- If the subdomain will be the owner of a whole range of IP addresses (for example, 9 or 9.179 or 9.179.84), the delegation should be done the same way as explained above. It is necessary just to set up all the files in the subdomain name server and to put an entry in the domain name server reverse data file pointing to the subdomain. In this case, the domain being delegated will be *.in-addr.arpa, where * are the first bytes of the IP address for the range in reverse order.

  Continuing the previous example of domain delegation, to delegate a reverse domain whose network IP address is 10.0.1.0 to a server named machine.dept.company.com, the entry in the reverse data file (/etc/named.rev) of the main domain server for company.com (which is the owner of all the 10 network) would be:

  ```
  1.0.10.in-addr.arpa.     9999999 IN NS machine.dept.company.com.
  ```

- If the subdomain will only have a subset of a range of IP addresses (for example, 9.179.184.100 to 9.179.184.200), the name server of the owner of the range must have the IP address-to-hostname entries for all the subdomains in its reverse name data files because you cannot delegate a part of a domain (or reverse domain) to other servers.

Appendix C. Hints and Tips

This appendix contains some hints and tips for ISS and Network Dispatcher configuration that were learned from experiences in writing this book.

C.1 ISS

- The `issd` must be able to resolve all the names contained in the ISS configuration file. To avoid having problems in case of external DNS failure, put all addresses (qualified and non-qualified) used in the ISS configuration file in the local /etc/hosts file.
- Do not forget to put the full, qualified pool name followed by a period (.) in the DNS named.data file when using `NameServer` Observer. Otherwise `issd` will not find the service name in that file.
- ISS always puts the full reverse DNS address when you use the NamedRev field in the ISS configuration file instead of putting only the part that was not mentioned in the named.boot file.
- When using `Dispatcher` Observer, specify the same server name or IP address in the InternalNet of the Network Dispatcher Node definition of the ISS configuration file as the one specified for the Network Dispatcher cluster.

  For example, if you have the following lines in the ISS configuration file:

  ```
  Node  9.1.1.1
  InternalNet  10.1.1.1
  ```

  the command to use to add this server to a Network Dispatcher cluster should be:

  ```
  ndcontrol server add <cluster_address>:<port>:10.1.1.1
  ```

  If you do not do this, Network Dispatcher will not use any information ISS sends.
- If your round-robin configuration does not work, you should define in the ISS file at least one `ResourceType`, and include it in the `Service` that will use the `RoundRobin SelectionMethod`. The `issd` monitor will not use that `ResourceType`, but it needs it to completely identify the Service. For example:

  ```
  Service www_service www.yyy.com
  NodeList  server1 server2 server3
  ResourceList CPULoad
  SelectionMethod  RoundRobin
  ```

© Copyright IBM Corp. 1997

- The hostname of an AIX machine must be set up through SMIT. ISS needs to recognize this hostname or the corresponding IP address. If your machine has more than one network interface (each one with its own DNS name and corresponding IP address) and you do not want to use the one corresponding to your main hostname to provide the service, you should either change your hostname to the one corresponding to the network interface you want to use or maybe define an internal net.
- When testing ISS, the SIGINT signal used to stop the `issd` did not work. We used the SIGTERM signal instead.
- For clean administration of your machines, do not forget to stop the `issd` monitor and agents first if you have to shut down your machines. Moreover, it is a good rule always to have a backup copy of your DNS configuration files.

C.2 Network Dispatcher

- The Network Dispatcher server must be able to resolve all the names contained in the configuration script. To avoid having problems in case of external DNS failure, put all addresses (qualified and non-qualified) used in the Network Dispatcher configuration script in the local */*etc/hosts file.
- It is **always** necessary to create an alias for the network interface that will receive the client requests for the Network Dispatcher server.
- It is **always** necessary to create an alias for the loopback interface for all the servers that will receive packets forwarded by the Network Dispatcher server.
- If you are using address mapping files, when you add a server to a cluster (using `ndcontrol server add` command), you must use the address of the back-end network interface and not the address of the public network interface.
- Network Dispatcher uses two default ports: 10003, used by `ndserver`, and 10004 to receive load feedback by ISS.

  To change the 10003 port, you have to change the ND_PORT value in the /usr/bin/ndserver and /usr/bin/ndcontrol files. To change the 10004 value, you have to enter the following command in the Network Dispatcher machine when you start the Manager:

  `ndcontrol manager start` *`logfile metric_port`*.

  If you specify a `metric_port`, you must specify a `logfile` name.

Furthermore, when you define the `Dispatcher` Observer in the ISS configuration file, you should indicate the port you have decided to use:

`Dispatcher` *`hostname metric_port`*

- One of the default ports used by Network Dispatcher (port 10003) is used by one of the SP daemons. So, if you use the Network Dispatcher server as one SP node, do not forget to change this port number by changing the ND_PORT value in the /usr/bin/ndserver and /usr/bin/ndcontrol files.

Appendix D. Special Notices

This publication is intended to help system administrators to implement the Interactive Network Dispatcher to balance the load of Internet servers. The information in this publication is not intended as the specification of any programming interfaces that are provided by Interactive Network Dispatcher product. See the PUBLICATIONS section of the IBM Programming Announcement for Interactive Network Dispatcher for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have

© Copyright IBM Corp. 1997

been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

The following document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX ®	Deep Blue
HACMP/6000	IBM®
LoadLeveler®	OS/2®
RS/6000	Scalable POWERparallel Systems®

The following terms are trademarks of other companies:

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Appendix E. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

E.1 International Technical Support Organization Publications

For information on ordering these ITSO publications, see “How To Get ITSO Redbooks” on page 135.

- *Olympic-Caliber Computing*, SG24-4279
- *High Availability on the RISC System/6000 Family,* SG24-4551
- *An HACMP Cookbook*, SG24-4553
- *HACMP/6000 Customization Examples,* SG24-4498
- *RS/6000 SP System Management: Easy, Lean and Mean*, GG24-2563
- *High Availability on RISC/6000 SP*, SG24-4742
- *SP Problem Determination Guide*, SG24-4778

E.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
Lotus Redbooks Collection	SBOF-6899	SK2T-8039
Tivoli Redbooks Collection	SBOF-6898	SK2T-8044
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SBOF-8700	SK2T-8043
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

© Copyright IBM Corp. 1997

E.3 Other Publications

These publications are also relevant as further information sources:

- *Interactive Network Dispatcher User's Guide - Version 1.2 for AIX, Windows NT, and Sun Solaris,* GC31-8496
- *Using and Administering Interactive Session Support for AIX, Release 1,* SC28-1548
- *Using and Administering LoadLeveler Release 3.0*, SC23-3989
- *HACMP/6000 Concepts and Facilities*, SC23-2699
- *HACMP Planning Guide*, SC23-2700
- *HACMP/6000 Installation Guide*, SC23-2701
- *Performance Toolbox for AIX Guide and Reference, Version 1.2 and 2,* SC23-2625
- *Performance Toolbox Parallel Extensions for AIX Guide and Reference*, SC23-3997
- *DNS and BIND,* Paul Albitz & Cricket Liu, O'Reilly & Associates, Inc., SR28-4970, ISBN 1-56592-236-0

How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at `http://www.redbooks.ibm.com`.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** – to order hardcopies in United States
- **GOPHER link to the Internet** – type `GOPHER WTSCPOK.ITSO.IBM.COM`
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get lists of redbooks:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Web Site on the World Wide Web**

  `http://w3.itso.ibm.com/redbooks`

- **IBM Direct Publications Catalog on the World Wide Web**

  `http://www.elink.ibmlink.ibm.com/pbl/pbl`

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **On-line** – send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

© Copyright IBM Corp. 1997

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **On-line Orders** (Do not send credit card information over the Internet) – send orders to:

	IBMMAIL	**Internet**
In United States	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** – send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** – send orders to:

United States (toll free)	1-800-445-9269
Canada	1-800-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 408 256 5422 (Outside USA)** – ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** – send note to `softwareshop@vnet.ibm.com`
- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to `announce@webster.ibmlink.ibm.com` with the keyword `subscribe` in the body of the note (leave the subject line blank).

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

☐ Invoice to customer number

☐ Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Abbreviations

ANSI	American National Standards Institute
ARP	Address Resolution Protocol
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
AFS	Andrew File System
AIX	Advanced Interactive eXecutive
APAR	Authorized Program Analysis Report
ATM	Asynchronous Transfer Mode
BOS	Base Operating System
CGI	Common Gateway Interface
CNAME	Canonical Name (DNS Configuration)
CPU	Central Processing Unit
DBMS	Database Management System
DCE	Distributed Computing Environment
DFS	Distributed File System
DNS	Domain Name Server
IN	Internet (DNS configuration)
FDDI	Fiber Distributed Data Interface
FTP	File Transfer Protocol
HA	High Availability
HACMP	High Availability Cluster MultiProcessing
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IBM	International Business Machines Corporation
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IND	Interactive Network Dispatcher
I/O	Input/Output
IP	Internet Protocol
ISP	Internet Service Provider
ISS	Interactive Session Support
ITSO	International Technical Support Organization
JFS	Journaled File System
LAN	Local Area Network
MAC	Medium Access Control
ND	Network Dispatcher
NFA	Non-Forwarding Address
NIS	Network Information System
NFS	Network File System
NS	Name Server (DNS Configuration)
NNTP	Network News Transfer Protocol
OLTP	On-Line Transaction Processing
OS/2	Operating System/2

© Copyright IBM Corp. 1997

POP	Post Office Protocol
PTF	Program Temporary Fix
PTPE	Performance Toolbox Parallel Extension
PTR	Pointer (DNS Configuration)
PTX	Performance Toolbox
RDBMS	Relational Database Management System
RS/6000	IBM RISC System/6000
RS/6000 SP	IBM RS/6000 Scalable POWERparallel Systems
SHTTP	Secure HyperText Transfer Protocol
SMIT	System Management Interface Tool
SMP	Symmetric MultiProcessor
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SOA	Start of Authority (DNS Configuration)
SP	IBM RS/6000 Scalable POWERparallel Systems
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TTL	Time-To-Live
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WAN	Wide Area Network
WWW	World Wide Web

Index

Symbols

/etc/hosts file 115, 120, 127, 128
/etc/named.boot file 119, 120
/etc/named.data file 124
/etc/named.rev file 124
/etc/resolv.conf file 76, 77, 123

A

Advisors 11, 17, 18, 79
Authkey keyword 29, 43, 102, 104

B

Best keyword 24, 27, 34, 51, 66, 103
bibliography 133

C

Cell keyword 13, 23, 28, 43, 102, 104
CLUSTER keyword 68, 99
Configuration
  High Availability 106
  ISS 111
  Minimal Configuration 111
  Name Server 119
  Network Dispatcher 112
Configuration Files 28, 93
  ISS 101
  Network Dispatcher 97
Custom Metrics 112

D

Dispatcher keyword 23, 24, 26, 32, 89, 127, 129
DNS
  See Domain Name System
Domain Name System 8, 10, 12, 49, 75, 95, 115
  High Availability 75
Domino 72
Dynamic Pages 48, 55

E

Executor 11, 17, 79, 81
Expect 63

F

File Transfer Protocol 2, 8, 61, 63
  Testing 61
FTP
  See File Transfer Protocol
ftpd daemon 61

G

graphics 47, 61, 63

H

Hardware Requirements 39
HeartbeatInterval keyword 29, 43, 90, 102, 104
HeartbeatsPerUpdate keyword 29, 43, 90, 102, 104
HeartbeatsToNetFail keyword 43
High Availability 6, 75
  Configuration 79, 106
  Network Dispatcher 78
  Testing 81
HTML
  See Hypertext Markup Language
HTTP
  See Hypertext Transport Protocol
Hypertext Markup Language 2, 8
Hypertext Transport Protocol 8

I

IND
  See Interactive Network Dispatcher
Interactive Network Dispatcher 4, 8, 9, 17
  Components 4
  Functions 9
  Why Do I Need it? 5
Interactive Session Support 4, 9, 12, 17, 23, 127
  Cell 13, 23, 28
  Configuration Files 28, 101
  High Availability 75
  issd daemon 23
  ISSNameserver keyword 25
  Methods 27
  NameServer keyword 24
  Observers 24
  Service 13, 23
InternalNet keyword 30, 102, 104, 127
Internet 1, 2

© Copyright IBM Corp. 1997

Dealing with the Growth 2
intnd.iss.rte fileset 40
intnd.msg.en_US.iss fileset 40
intnd.msg.en_US.nd fileset 40
intnd.nd.rte fileset 40
intnd.ps.en_US fileset 40
Intranet 1
ISS
See Interactive Session Support
issd daemon 23, 24, 25, 28, 30, 33, 44, 51, 58, 59, 75, 87, 90, 91, 127
ISSNameserver keyword 24, 25, 26, 32, 49, 75, 89

L

LoadLeveler keyword 78
LogLevel keyword 29, 43, 102, 104
loopback interface 79

M

Manager 11, 17, 79
Metric 31, 32, 112
External 35
Internal 34
Writing it 36
Metric keyword 35, 44, 102, 105
MetricLimits keyword 36, 44, 102
MetricNormalization keyword 44, 102, 105
Minimal Configuration 111

N

Name Server 117
Configuration 119
named daemon 15, 24, 25, 63, 66, 76, 87, 90, 115, 119, 120
NamedData keyword 44, 103, 105
NamedRev keyword 44, 103, 105
NameServer keyword 24, 25, 26, 32, 44, 49, 63, 75, 78, 89, 90, 103, 105, 127
ndcontrol command 19, 42, 68, 80, 81, 100, 106, 107, 127
ndserver command 19, 41
Network Dispatcher 4, 9, 10, 17, 54, 68, 78, 128
Advisors 11
Components 11, 17, 19
Configuration 97
Executor 11
High Availability 78
log file 20
manager 11
ndcontrol command 19
ndserver command 19
Proportions 21
Node keyword 30, 43, 75, 102, 127
NodeList keyword 32, 44, 89, 103, 105, 127
NotMonitor keyword 30, 43, 75, 102, 104

O

Observers 24, 32, 49, 127
Overflow keyword 32, 44

P

Pages
Dynamic 48, 55
Static 48, 57
ping triangulation 28
Policy keyword 35, 44, 102, 105
pool 13
port 22, 82, 98, 128
PortNumber keyword 29, 35, 43, 102, 104
Proportions 21, 22, 55, 69

R

ResourceList keyword 44, 103, 105, 127
ResourceType keyword 23, 24, 27, 31, 44, 51, 52, 59, 66, 90, 92, 102, 105, 127
Round-Robin 32, 33, 49, 51, 64, 87, 101, 111
RoundRobin keyword 24, 27, 49, 59, 70, 78, 111, 127
RS/6000 22, 47, 48, 61, 81, 93, 97

S

Samples
Configuration Files 93
Scenario
Domino 72
FTP 61
High Availability 75
Telnet 71
Two-Tier 87
WWW 47
SelectionMethod keyword 25, 32, 33, 34, 44, 51, 103, 105, 127
Service 13, 23, 31
Service keyword 32, 44, 89, 103, 105, 127

ServiceList keyword 44, 89, 103, 105
signal 63, 128
Software Requirements 39
Static Pages 48, 57

T

Telnet 71
Test Environment 47, 61, 93
Testing
    FTP 61
    WWW 47
Two-Tier 87
    Back-End Tier Configuration 91
    Front-End Tier Configuration 89
    ISS 87
    Network Dispatcher 87

W

Web server 48
WebStone 48
World Wide Web 1, 2, 4, 8, 9, 47
    Testing 47
WWW
    See World Wide Web

ITSO Redbook Evaluation

Load-Balancing Internet Servers
SG24-4993-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the on-line evaluation form found at http://www.redbooks.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@vnet.ibm.com

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

_____

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____

_____

_____

_____

_____

© Copyright IBM Corp. 1997



Printed in U.S.A.