RC 18442 (80763) 10/26/92
Computer Science 13 pages

Research Report

A Virtual Multiprocessor Implemented by an Encapsulated Cluster of Loosly Coupled Computers

IBM
RESEARCH LIBRARY

C.R. Attanasio and S.E. Smith

IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598

'92 DEC -4 P4:35

UNLIMITED DISTRIBUTION

NOTICE

This report will be distribuled outside of IBM up to one year after the IBM publication date.

IBM Research Division
Almaden • T.J. Watson • Tokyo • Zurich

A Virtual Multiprocessor Implemented By an Encapsulated Cluster of Loosely Coupled Computers

April 29, 1992

C. R. Attanasio
S. E. Smith

Advanced RISC Systems
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598

Abstract

The advantages of clustering workstations to provide expanded capability and availability are well known in the literature. Previous efforts have focussed on providing a single system image at each node of the cluster. In this work we provide the complement view, that the cluster appears as a single host to other hosts not part of the cluster. A message switch at the network connection routes messages transparently to the nodes of the cluster.

Introduction

Goals

Horizontal Growth

The amount of computing power available to the end user of a workstation has increased dramatically with the advent of RISC technology. Still, demand for ever-increasing cpu power persists; existing operating systems and applications are making computers ever more attractive to ever more users.

Hardware technology provides approximately 100% increase in computing power every two years. To provide greater capacity in one unit, many manufacturers offer tightly-coupled multiprocessors, multiple processors accessing the same real main storage and i/o configuration.

There are limits to the advantages of these two approaches. At some point the current rate of hardware speed increase may lessen. Tightly-coupled multi-processor versions of modern, pipelined and cached processors are difficult to design and implement, and become more so as the number of processors increases. This causes product cycle time to lengthen, a severe handicap in today's fast-moving workstation marketplace. Furthermore, the requirement to support multi-processing often reduces the speed of an architecture's implementation over what would be attainable by a uni-processor. A multi-processing operating system, if not available, must be produced. If one is available, it is likely that there is an intrinsic performance penalty because of m-p capability, as there is in hardware.

System Administration

A perceived advantage of workstations is that users can own their own machines. However, not all users want to or are able to maintain their own workstations. The administrative burden of maintaining large numbers of individual workstations which are physically distributed in offices, laboratories, etc., is considerable. As the sophistication of the software required by these workstations grows, the work required to keep these workstations operating correctly increases.

There is a requirement for being able to "batch" system maintenance and administration, i.e., to administer many machines with a single work effort.

System Technology Enhancements

System technology is improving in all areas. Introducing enhanced technology into a widespread environment is possible only when existing interfaces can be met. For central processors the difficulty is contained by widely accepted communication interfaces. However, advanced communication technology between machines is constrained by having to use software protocols which were defined when communication technology was more primitive.

There is a requirement to be able to apply advanced communication technology to a set of machines in such a way that the interface to the internet still functions. Examples of such communication technologies are advanced optical links and more closely coupled, e.g., bus-connected main storage units.

Overview of the Encapsulated Cluster

We have embarked on an activity to allow a set of loosely-coupled workstations to appear as a single host in the internet. We call our design the ***encapsulated cluster***. The individual nodes of the ***encapsulated cluster*** are never separately visible outside the cluster in the internet. The entire cluster is known by one name only. All communication with the cluster from outside is logically to and from a single host. With this fundamental approach we address each of the above-mentioned goals.

The ***encapsulated cluster*** can be viewed as a large-way multiprocessor, each of whose processors pays no performance penalty over a uniprocessor. Horizontal growth occurs with no changes outside the cluster. The enlarged cluster still appears as a single host in the internet.

Similarly, the nodes of the ***encapsulated cluster*** would most likely reside physically in close proximity, and would be under control of a single administrator. Furthermore, there is no change to routing tables, hostname resolution, etc., as processors are added to the cluster.

Since the internal nodes of the ***encapsulated cluster*** are not known outside the cluster, communication among them can make use of advanced interconnect technologies and non-standard protocols. Communication with the internet, of course, is based on standard interfaces and protocols, executed by one (or more) cluster nodes which manage communication for the cluster with the internet. We call such a node the "gateway" for the ***encapsulated cluster***, remembering that it is not a gateway in the IP sense, since communicators outside the cluster do not have IP addresses for nodes inside.

This approach is in contrast to TCP/IP subnetting and "transparent gateway"(1), which are techniques to allow physically distinct networks to appear logically as one. These techniques do not conceal the existence or the addresses of the nodes on the distinct networks.

Other Work

The advantages of clustering computers are well-known, and many examples of such efforts have been reported (3,4,5,7,8,9,10,11). The degree to which the cluster appears to be a single machine varies in these efforts, which concentrate on the view of the cluster from within.

DUNIX (7) is a restructured UNIX kernel which makes several computers appear as a single machine. The "upper kernel" is entered on system calls from applications. At this level there is an explicit call to the "switch" component, which routes the call, on the

basis of the object referred to, to the proper machine to complete processing of the system call, in the "lower kernel".

This work differs from ours in two ways:

- DUNIX creates a single system image for a group of machines as seen from within; no discussion, beyond that the group is connected to the Internet, is given about single system image as seen by other machines not in the group.
- the "upper kernel" explicitly calls the "switch" routines; their switch is actually a conventional RPC mechanism.

Amoeba (8) is explicitly a distributed system. It provides UNIX capability only via emulation; the base operating system is new and not UNIX. The management of the distributed system is based on object management, rather than transparent message routing, as in our work.

x-kernel (9) is an operating system kernel designed to support the implementation of network protocols. It relies on others to provide full computing environments.

Sprite (10) is an explicitly distributed environment; it provides mechanisms for process migration and remote procedure call, and implements a distributed file system.

V (11) is a new, distributed operating system in which the multiple machines are largely but not totally transparent to the applications, as the reference describes. The "V inter-kernel gateway" is designed to interface between two V-System clusters; the remote relationship between clients and servers is supported only for processes in V-systems.

The works referenced here all provide some degree of single system image by writing new kernels (in the case of Amoeba, a totally new operating system). In Amoeba and V, the workstations by which users access the system are internal to the system. They run the system's modified kernel, and communicate with servers inside the system using new software and protocols. In the encapsulated cluster, user workstations are external to the cluster, and use conventional software and protocols (rlogin, ftp, NFS) to access the cluster, as if it were another host on the internet. This paper describes techniques for allowing an existing, industrial grade kernel, to be used in a way that many machines appear as one to the internet, where existing operating systems execute *unmodified.*

Existing techniques for managing distributed systems, such as LOCUS (3), TCF (4) and DCE (5) create various simulations of "single system image" for file name space and process name space. These are provided at levels higher than IP address and port number. The hosts participating in these environments retain their individual identity throughout the internet. When hosts are added to or taken out of these environments, network-wide facilities for name resolution and message routing are affected. In our work, outside the cluster there is only one hostname, and therefore additions to the cluster do not cause any changes beyond the cluster.

Further, one or more non-gateway nodes of the cluster might be executing non-standard control programs, for example, to provide a special function such as database management. So long as such nodes maintain communication with their peers inside the cluster,

their functions can be made available to the internet via application-level communication mediated by the gateway.

IP Background

Our initial implementation treats IP messages using UDP or TCP; such messages are addressed to a port at an IP host address. UDP is a datagram or connection-less protocol. The transport mechanism does not guarantee delivery of messages; only the IP address and port number identify the receiving process. TCP is a connection-based protocol; message delivery and sequencing is guaranteed. The connection, i.e., the four-tuple--source IP-address, source port, target IP-address and target port--identify the sending and receiving processes. A port at a host may be part of more than one connection.

There are more or less formal conventions for allocation and use of port numbers in various ranges:

- ports between 1 and 512 are used as "well-known" port numbers associated with services listed in the /etc/services file.
- "privileged" ports are those between 512 and 1024, and are available only to processes executing with superuser privilege.
- we call the range of port numbers between 1024 and 5000 the "dynamic" range; these ports are used when applications request a non-privileged, non-specific TCP or UDP port.
- ports above 5000 are designated "user" and are intended for user-written applications.
- programs may request a specific port number in the privileged range or the user range, or may ask for any available port number in the user range, at the kernel interface.
- the kernel interface allows more than one socket to be bound to a specific port number, by request, to accomodate TCP connections. If the application does not allow multiple connections to a given port, the kernel returns an "in-use" indication for a specific request for a previously allocated port.
- many applications are written to request a specific port number, but have logic to request another specific port number when receiving an "in-use" response. Indeed, this is the only way a non-specific privileged port can be obtained.

There are two recommended ways that clients may discover the number of the port at which a service may be requested. The file /etc/services contains a set of "well-known" port numbers and service pairs. Clients invoke the service by addressing the port number. One of these "well-known" services is "Portmapper", which is a program that remembers server programs and their associated ports which are not listed in /etc/services. At its initialization (or restart) time, a server program may obtain a non-specific port, and call Portmapper to register its identity and obtained port number. Clients at other hosts access the service by obtaining its port number from Portmapper.

Clients generally obtain non-specific ports to communicate with servers, establishing connections if TCP-based, or sending datagrams if UDP-based.

The Encapsulated Cluster

Fig. 1 illustrates the structure of the ***encapsulated cluster***. The internet represents all machines other than those in the cluster; all such machines know only the name/IP-address of the distinguished machine of the cluster, which we call the *cluster gateway*. Our figure shows only one machine of the cluster connected to the internet. Our general architecture allows for the cluster to be connected to more than one network, through one or more gateway machines. Each network would know only one machine of the cluster, and would interact with the entire cluster as if it were one machine. By adding networks, connectivity and bandwidth to the internet can be increased as necessary. Our prototype uses a fiber optic point-to-point switch as the communication medium between nodes of the cluster. Support software simulates a network interface between these nodes, allowing the use of standard IP communication, and therefore industry standard software between these nodes.

From outside, the cluster appears to be a single host with a single set of UDP and TCP ports. A ***message switch***, residing in the *cluster gateway*, processes messages coming into the cluster from outside, and going out to the network from inside the cluster.

All messages to the ***encapsulated cluster*** arrive with the ***cluster gateway*** address as target node; if the target port is one for which cluster-oriented processing is required, that is done after IP-level processing is completed, but before the message is passed to the protocol (TCP or UDP) layer. If not, processing for the message is completed locally in the gateway. This allows cluster features to be added incrementally.

"Cluster-oriented processing" for inbound messages falls into three categories:

- route a message to a cluster node because it is directed to a service which executes in more than one cluster node, i.e., has become "distributed" in the cluster. Examples are MOUNT, NFS, RLOGIN, RSH, REXEC, and TELNET.
- route a message to a cluster node which contains the only instance of the service requested. We have no examples at this time, but an example would be a special purpose server in the cluster, e.g., a database server.
- route a message to a cluster node because it is associated with a client in a cluster node requesting service from a host outside the cluster.

The ***message switch*** resides in the ***encapsulated cluster*** gateway and contains a set of tuples--cluster port number and protocol, cluster node, and an associated function for routing messages arriving at that port. When a message arrives at the ***cluster gateway***, the ***message switch*** is scanned for an entry for the target port number and protocol. If an entry is found, and the associated function is NULL, the message is routed to the associated cluster node. If an entry is found and the associated function is not NULL, it is called with the message as argument. The function, which is service-specific, reads the data of the message to determine the cluster node and port number to receive the message and the message is then forwarded.

The allocation of protocol port numbers within the cluster is controlled (described below) so that client processes requesting non-specific ports, obtain port numbers "be-

longing" to the node upon which they are executing. When the ***message switch*** sees an inbound message with one of these port numbers, it is able to forward the message to the associated node, without the need for a routing function.

In all cases, the target address of the forwarded message is changed from that of the *cluster gateway* to that of the internal cluster node which receives the message.

If the ***message switch*** does not find a routing function for a particular port, and the port is not associated with a node as a result of the port allocation strategy, then the containing message is processed locally in the gateway.

"Cluster-oriented processing" for messages outbound from the cluster consists of changing the source IP address in the message to that of the ***cluster gateway*** from that of the cluster node originating the message. Remote hosts see messages arriving only from the ***cluster gateway***, not from individual nodes of the cluster.

Managing Cluster Ports and Connections

To maintain the external appearance of the cluster being a single host, the UDP ports within the ***encapsulated cluster*** must be unique. For TCP communication, the connections must be unique, although a given port may participate in more than one connection. We perform port allocation locally in each node whenever possible, and communicate with the gateway only when necessary to maintain the image of a single host. Ports and connections used only for intra-cluster communication are not visible outside the cluster, and do not require any special cluster processing.

We use a hybrid algorithm to manage ports and connections in the ***encapsulated cluster***:

- we do no processing for the ports in the range 1 to 512; consequently, each of the services associated with those ports start normally in each node of the cluster. Unless a routing function is installed in the gateway ***message switch*** table, external requests for one of these services will be processed within the gateway.
- when a request is made for a specific TCP port in the privileged range (512-1024) preexisting logic is used to grant or not grant the request locally in each cluster node. This may result in the situation where a TCP port appears to be unique in a cluster node, but may also have been allocated in another cluster node. When TCP ports are connected to ports at external hosts, that connection is recorded in the cluster gateway, and is guaranteed to be a unique connection. If an attempt to connect fails because of a previous connection from the cluster to the same external address from the same cluster port, address-in-use is returned, and TCP clients retry using a different port number. The connection information is used in the gateway to route subsequent inbound messages for the connection to the correct cluster node.

- UDP ports in the privileged range are guaranteed to be unique. When an application attempts to bind to a UDP port in this range, communication with the cluster gateway ensures that the port has not been previously bound at another cluster node, and registers the port and its node, so that subsequent inbound messages to that port are correctly routed.

- requests for arbitrary non-privileged TCP or UDP ports are handled locally in the nodes. The port number range from 16k to 64k is reserved for such requests, and is partitioned among the cluster nodes. Each node allocates such requests from the subrange of port numbers assigned to it. The ***cluster gateway*** can route inbound messages with destination port numbers in this range, using just the port number, since it knows the partitioning algorithm.

Prototype Implementation

Overall Approach

The internal composition of the ***encapsulated cluster*** is invisible outside the cluster, and therefore we have total freedom to change software inside the cluster as required to attain our goals. We adopted the ground rule that we can require no software changes to code running outside the cluster. We added cluster functions incrementally to a set of running systems, so that that they were operational almost all the time. We used existing software wherever possible, and made those changes necessary to demonstrate the feasibility of our approach.

Initially Available Capabilities

Our initial implementation allows users to perform RLOGIN, REXEC, RSH, and TELNET commands directed to the ***encapsulated cluster***. The session associated with such a command occurs on one of the cluster nodes, chosen by a scheduling algorithm. All the subsequent communication for that session is routed between the user and the cluster node through the ***encapsulated cluster gateway***.

X-station sessions are similarly distributed among ***encapsulated cluster*** nodes with X-clients connecting to X-servers outside the cluster. (X-stations are not physically connected to the internal cluster node connection medium.)

Users who have such sessions see the same filespace independent of the cluster node chosen. This is accomplished by mounting the required filesystems on each node of the cluster. Filesystems that reside on nodes of the cluster are accessible at all nodes of the cluster by "cross-mounting" them.

Hosts outside the ***encapsulated cluster*** view its file system tree as a single tree, even though it may consist of subtrees physically resident on different nodes of the ***encapsulated cluster***. As with a single host, portions of its filetree can be exported and NFS-mounted on other machines and accessed by NFS clients on those machines. Since NFS does not support inherited mounts a special mechanism was provided to accomplish this.

Encapsulated Cluster MOUNT Server

MOUNT acquires a privileged port at startup and registers it with PORTMAPPER, at the ***encapsulated cluster gateway*** as well as at each node where MOUNT is running. PORTMAPPER has been modified to call a system extension, informing it of the port associated with MOUNT. At the cluster gateway, MOUNT's port is inserted into the ***message switch***, in the prespecified entry associated with MOUNT's message routing function. At cluster non-gateway nodes, a message is sent to the gateway to enter into an auxiliary table MOUNT's port on the node. MOUNT's routing function uses this table to send an external MOUNT request to the appropriate node, when it determines which cluster node should process an external MOUNT request.

A configuration file in the gateway defines which directories in the ***encapsulated cluster*** are "cluster-exported", and records the cluster node on which each resides. When an incoming message is addressed to MOUNT's gateway port, the MOUNT message processing function reads the message for the requested directory and compares it against the file entries. If a match is found, or the request is for a subdirectory of an entry in the configuration file, the MOUNT function forwards the message to the MOUNT daemon executing at the specified node.

MOUNT requests for entries not cluster-exported are processed locally in the gateway.

When a MOUNT request succeeds, the client receives from the server a "filehandle", a thirty-two byte token opaque to the client, which is used for subsequent access to the mounted directory. Our implementation stores a cluster node identifier in an unused field of the filehandle which is used to route subsequent NFS requests for that mounted filesystem to the cluster node on which it resides.

Encapsulated Cluster NFS Server

NFS servers receive messages on a "well-known" port, 2049, using UDP, a connectionless protocol. At each cluster node at which an NFS server executes, it receives its messages at port 2049. Each request (except NOOP) includes as data a previously obtained "filehandle" which identifies the file to which the request is addressed and, by our modification, identification of the cluster node to which the message should be routed. MOUNT returns the first modified filehandle. Each subsequent directory lookup returns a similarly modified filehandle.

The NFS message routing function, invoked through the ***message switch*** whenever a message to port 2049 appears at the gateway, finds the cluster node in the filehandle in the NFS request, and routes the message appropriately. The NFS NOOP function contains no filehandle, and is processed in the ***cluster gateway***.

Encapsulated Cluster TCP-Based Servers

RLOGIN, REXEC, RSH and TELNET are examples of TCP connection based services associated with well known port numbers listed in /etc/services. Clients establish connections with the well known port associated with the service they want. The treatment

for these services is the same, except for the value of the well known port, so we limit the discussion here to RLOGIN, whose well known port is 513. Figure 2 illustrates the data structures and control flow for RLOGIN.

On each node of the cluster, including the gateway, the normal rlogind daemon is started. Each "listens" on port 513, waiting to "accept" connection requests from remote rlogin clients. While there are multiple occurrences of port 513 and the associated rlogin daemon within the cluster, when viewed from the internet the cluster is a single host, with a single rlogin daemon.

A routing function for rlogin is installed in the gateway message switch table for TCP port 513, and is invoked for each inbound message directed to this port. If a client initiating a connection is detected, a node for the new login session is chosen by a load-balancing algorithm. The algorithm can choose any cluster node, including the gateway node. The connection and the selected node are recorded in a connection table, and the packet is forwarded to port 513 at that node. The TCP connection and the login session are processed at the selected node in the usual manner. Subsequent inbound packets for this connection are forwarded by the gateway to the node selected for the connection. When the flag bits in the TCP header indicate a connection is being terminated, the gateway removes the table entry for the connection.

Support for X-stations.

Our objective of using the ***encapsulated cluster*** to support X-stations did not require any additional work beyond that needed for REXEC. When an X-station is turned on it begins a 2 step process. The first step is to download a binary image of the X server, and initialize that server. The second is to initiate a login process in an xterm window on the X-station. The machine booting the X server, and the machine running the login process need not be the same. At the end of the boot step, the user is given the option of selecting an alternate node for login. If the user selects the ***encapsulated cluster***, an rexec command is sent to the cluster gateway. When the rexec request arrives at the ***cluster gateway***, the message routing function for rexec, like rlogin, selects a cluster node, and forwards the rexec request to that node for processing. The rexec'd command starts an xterm process, running login, with the xterm window directed to the users X-station. In this way, X-station users connect to the ***encapsulated cluster*** and have their sessions distributed among the nodes of the cluster.

Performance Considerations

The ***encapsulated cluster*** provides a degree of load balancing, in that a node for a login session is explicitly chosen by a scheduling algorithm when the request is received at the gateway. The algorithm is currently very simple round-robin, but can easily be replaced.

The cluster architecture can be expected to provide enhanced performance for usage that involves communication between nodes of the cluster, in comparison with the same usage involving communication between hosts on a conventional network. Our cluster nodes are connected through an optical point-to-point switch whose bandwidth is more than ten times that of token ring. The effective gain thus far realized, for NFS file

transfers over the switch, as compared to over token ring, is approximately three times.(6) Since the cluster can be used as a compute and file server, we would expect users whose computation *and* file access is provided within the cluster to experience performance improvements in all aspects of their sessions except response time for terminal operations.

To quantify the effect of a remote session vs. a local session, we observed the responsiveness of "ping" operations, both to a different host on the same token ring, and through the cluster gateway to a node. Ping to another host on a token ring takes 4 to 5 ms. Ping to a cluster node, from a host on a token ring, through the cluster gateway on the same token ring, to a cluster node, take 6 to 7 ms. We have observed that for RLOGIN sessions, both to another host on the local token ring, and to a cluster node, response time is equivalent. In both instances, keystroke echo is sufficiently quick to keep up with keys depressed in repeat mode.

Future Directions

The next phase of our research, currently in progress, is to investigate how the high-availability technology described in (1) can be applied to the *cluster gateway*, so that a failure at the gateway can be detected, and its function taken over by a designated alternate. In this way, processes executing on nodes other than the failing gateway will continue without interruption.

We plan to install the highly-available filesystem of (1) in the *encapsulated cluster*

We intend to use the *encapsulated cluster* as a framework for research in how to provide reliability for user processes; that is, being able to restart processes which had been executing on a node which failed on surviving cluster nodes.

References

1. Douglas E. Comer, Internetworking with TCP/IP, Prentice Hall, N.J.

2. Bhide et al., "A Highly Available Network File Server", USENIX - Winter '91 - Dallas, TX.

3. G. J. Popek et.al., The LOCUS Distributed System Architecture, MIT Press.

4. AIX Transparent Computing Facility, IBM Corp., Armonk, NY.

5. Distributed Computing Environment, Open Software Foundation Cambridge, MA.

6. M. Blount, M. Butrico and K. Rader, Advanced RISC Systems, IBM Corp., T. J. Watson Research Center, Yorktwon Heights, N.Y., Private Communication.

7. Ami Litman, The DUNIX Distributed Operating System, Operating Systems Review, ACM Press, NY, Vol.22, No.1, January, 1988.

8. Andrew S. Tanenbaum et.al., Amoeba - A Distributed Operating System for the 1990s, IEEE Computer, May 1990.

9. Larry Peterson et.al., The x-kernel: A Platform for Accessing Internet Resources, IEEE Computer, May 1990.

10. Ousterhout, J.K., et al., The Sprite Network Operating System, IEEE Computer, February, 1988.

11. David R. Cheriton, The V Distributed System, Communications of the ACM, March 1988.

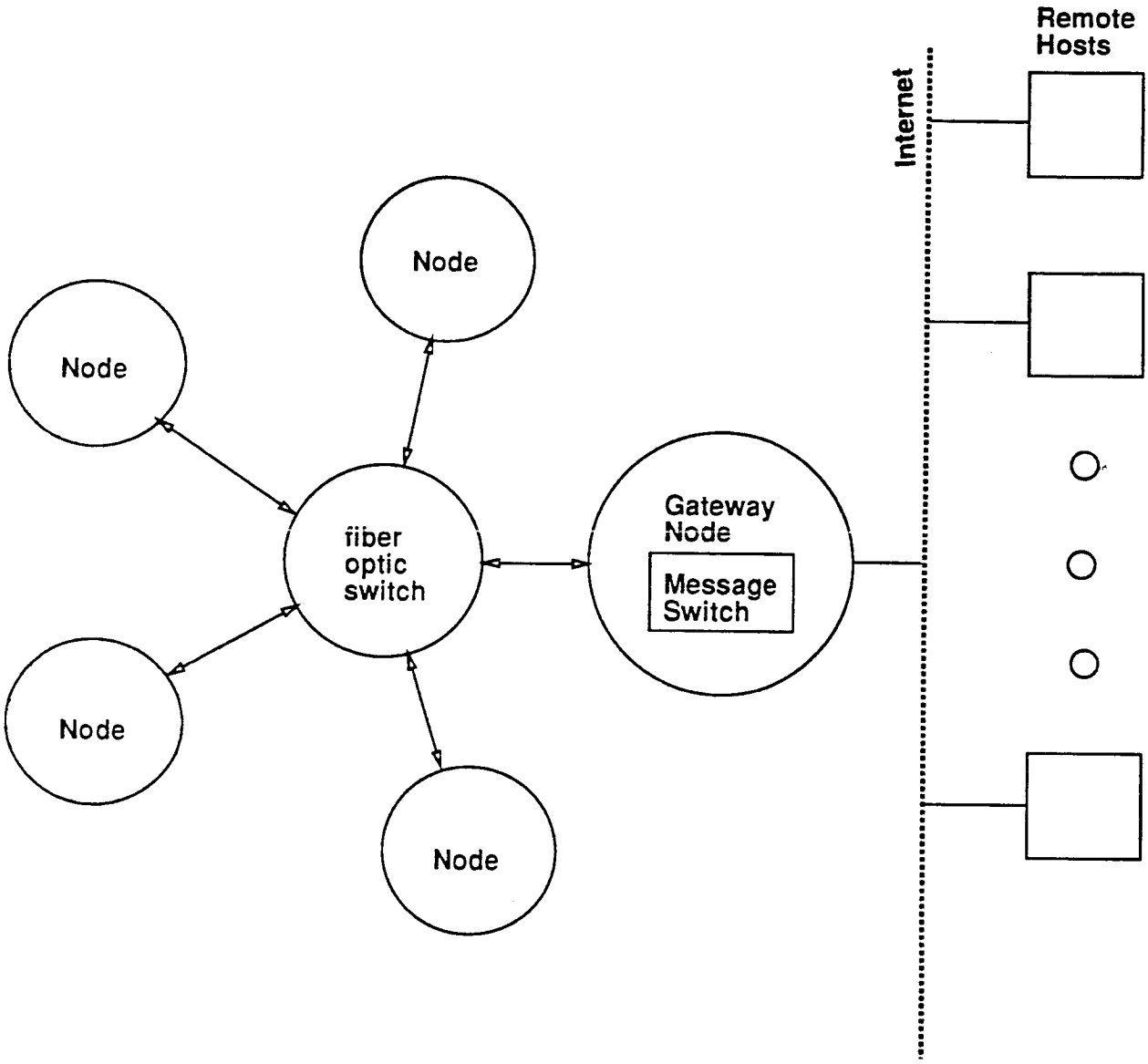

Fig. 1 Configuration of the Encapsulated Cluster

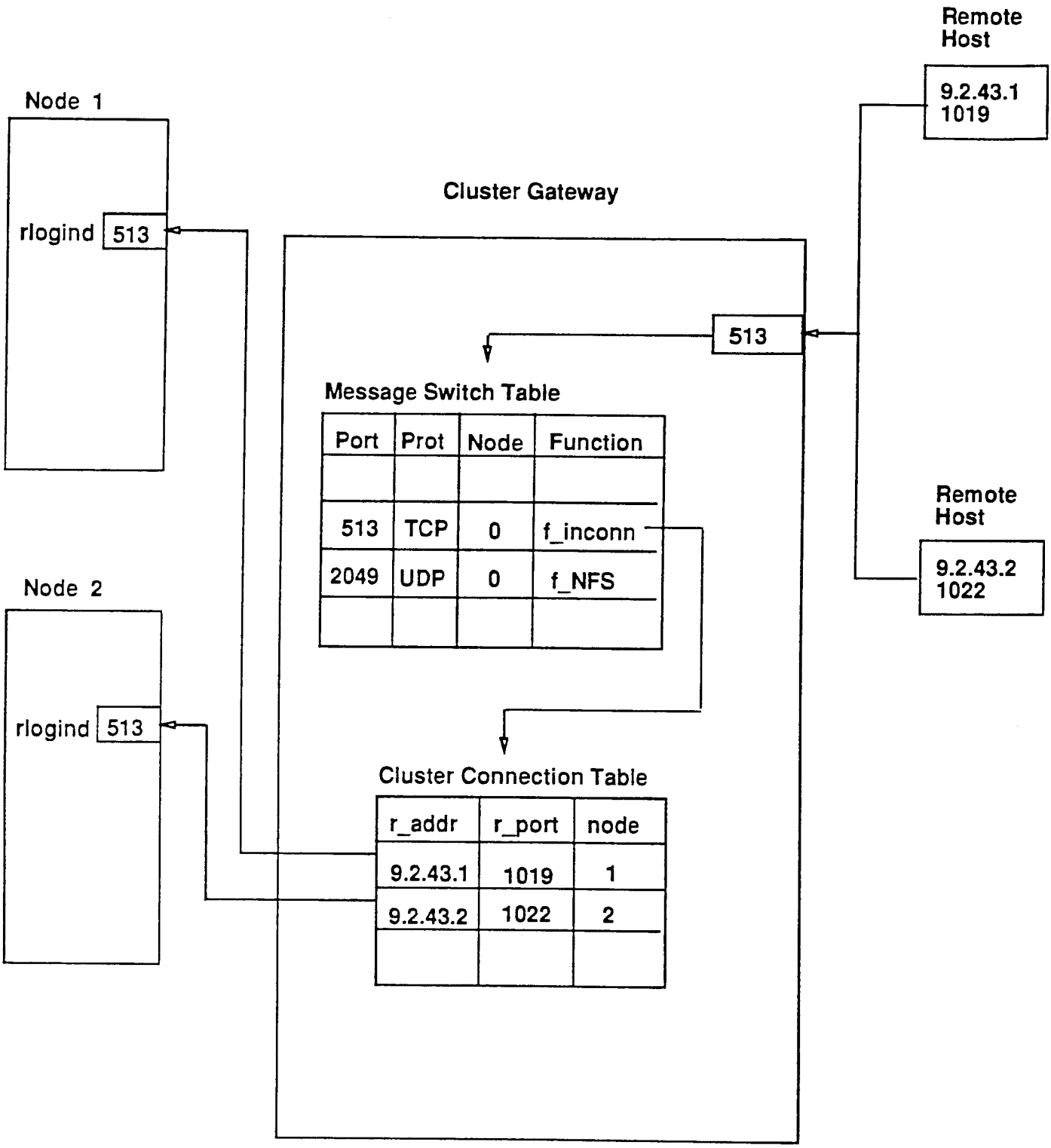

Fig 2. Managing Remote Login Connections