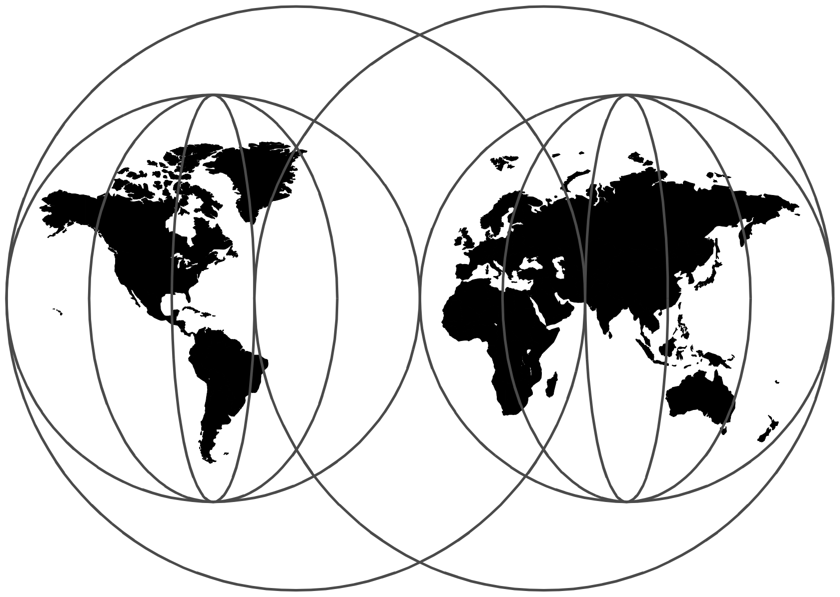
High Availability Scenarios for Tivoli Software

Vasfi Gucer, Bart Jacob, Nitin Pimplodkar, David Spurway
Jeff Mitzel, David Shiels



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-2032-01

SG24-2032-01

International Technical Support Organization

High Availability Scenarios for Tivoli Software

May 1999

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 103.

Second Edition (May 1999)

This edition applies to Version 3.6 of the Tivoli Enterprise Software for use in multiple operating environments.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept.OSJB Building 003 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

Contents

Figures

Tables

Preface

This redbook is a follow up to the previous redbook *Implementing TME 10 in High Availability Environments* (SG24-2032-00) that was written in December 1997 and fulfilled an important requirement in helping customers and Tivoli professionals in implementing Tivoli in high availability environments.

Since that time, the Tivoli product has been considerably changed from a two-tiered architecture to a three-tiered one. Also, function was added to the Tivoli framework to ease the implementation of Tivoli in HA environments.

This redbook covers new material, such as mutual takeover scenarios (TEC-TMR Server, Managed Node-Managed Node, Managed Node-TMR Server), and Tivoli Endpoint HA implementations as well as Microsoft Cluster Server scenarios in addition to HACMP and SUN Solstice examples.

We assume that the reader has a thorough understanding of high availability concepts and knowledge of at least one HA product. We also assume that the reader is familiar with the Tivoli framework and applications.

After reading this redbook, you should be well prepared to develop a detailed plan for implementing Tivoli in HA environments.

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's Austin Center.

Vasfi Gucer is an Advisory I/T Specialist at the International Technical Support Organization, Austin Center. He has worked at IBM Turkey for 10 years. He has been with the ITSO for 3 months. His areas of expertise include systems management and networking of distributed platforms.

Bart Jacob is a Consulting Software Engineer at the International Technical Support Organization, Austin Center. He writes extensively and teaches IBM classes worldwide on topics related to Tivoli. He has been with the ITSO since 1989 and has experience in systems management, distributed object frameworks, object-oriented application development and communications.

Nitin Pimplodkar is working at IBM India as a Technical Manager. He is supporting AIX, HACMP, and Tivoli implementations at IBM India.

David Spurway is working at IBM UK as an RS/6000 Pre-Sales Technical Support Specialist. He has 2 years of experience in HACMP implementation and extensive UNIX skills.

Jeff Mitzel is working as a Professional Services Specialist at IBM/Austin and has been working at IBM since 1992. He has experience in both High Availability development and services at IBM before joining Tivoli PS as a deployment specialist.

David Shiels is working as an Account Engineer in the Phoenix area reporting to the Tivoli Irvine office. He vast experience in implementing Tivoli in NT and UNIX environments.

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

Mark Adams
Tivoli Systems

Justine Patterson
Tivoli Systems

Mike Hahn
Tivoli Systems

Jeff Mills
Tivoli Systems

Thomas Knueppel
IBM Germany

Adam Majcherzyk
IBM Belgium

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 “ITSO Redbook Evaluation” on page 113 to the fax number shown on the form.
- Use the online evaluation form found at :`http://www.redbooks.ibm.com`

- Send us a note at the following address:

  `redbook@us.ibm.com`

Chapter 1. Introduction

Systems management is becoming an ever more important factor in the successful implementation of client/server environments. Users, and the companies they work for, depend on the availability of critical systems and applications. Systems management products, such as Tivoli, provide the facilities to manage the deployment and availability of client/server applications.

What happens when hardware or software errors occur that affect the availability of Tivoli applications? If one or more components of the Tivoli environment are not available, then you have temporarily lost the capability to manage the critical systems and applications in your network. Therefore, Tivoli should also be considered as an application whose availability is critical.

To help ensure the availability of specific systems and applications, there are many vendors that offer product solutions generally known as *high availability* (HA) products. HA products are designed to monitor critical systems and/or applications and restart those applications (possibly on a different physical system) in case of failure.

Hardware vendors, such as IBM, Sun, and HP, offer HA solutions specific to their hardware and operating environments. Also, Microsoft has an HA solution for its Window NT environment. In addition, other vendors, such as the Qualix Group, provide software-based solutions that run on different vendor's hardware and support a variety of operating systems.

Though each HA product has its own unique features and benefits, they all provide the same basic function. This redbook is not intended to differentiate among the various HA products but to simply indicate how Tivoli can be installed and configured to allow it to be managed by HA products. Specifically, the HA products that we tested were IBM's HACMP, Microsoft's Cluster Server, and Sun's Solstice. Though we were not able to test in an HP environment, we are confident that these same concepts and implementation techniques will work in an HP ServiceGuard environment.

By implementing Tivoli in an HA environment, you can more consistently ensure the capability to manage the resources in your network.

1.1 Tivoli Overview

Tivoli is the brand name given to a set of systems management products developed by Tivoli Systems and their partners. The Tivoli product family is based on an underlying framework (Tivoli Framework) that provides a set of common services used by the various Tivoli applications. The Tivoli Framework is an object-oriented environment in which operations are carried out through interfaces defined for various objects. The Tivoli Framework is based on the Object Management Group's Common Object Request Broker Architecture (CORBA), which is a commonly accepted industry standard for distributed object-based applications. From a high level view, the Tivoli Framework can be described as containing the following sets of services:

- Operating system interfaces/encapsulation
- Distributed object framework
- Distributed database
- Basic management functions
- Installation routines
- Command interface
- Graphical user interface (Tivoli Desktop)

The Tivoli Management Applications run on top of the Tivoli Framework and utilize the framework services. The applications provide systems management functions, such as:

- Inventory management
- Software distribution
- Distributed monitoring
- User and Security management
- Event processing
- and others

The Tivoli family of products are supported in a variety of operating environments such as

- Unix
- Windows NT
- Windows 98, 95, 3.x
- Netware 3, 4
- OS/2
- AS/400
- OS/390

In a Tivoli environment, systems are organized by regions called *Tivoli Management Regions* (TMRs)**.** Each TMR consists of a single TMR server that provides overall control of the distributed management database. The Tivoli database is a proprietary database that is spread across systems that make up a Tivoli Management Region. Communications between the TMR server and managed systems is handled through a process on each system called the oserv daemon. This daemon is an implementation of a CORBA-compliant object request broker. The oserv-oserv communications facility allows for the invocation of methods on objects representing managed resources.

Tivoli uses a TCP/IP-based protocol for its oserv-oserv communications. However, the protocol is hidden from users and application developers who simply reference specific objects in their method invocation requests, and the oserv daemon utilizes the services of the framework to locate objects and invoke methods on them.

Prior to the V3.6 release of Tivoli, the Tivoli Framework had a two-tiered architecture where there were two kinds of managed systems: *managed nodes* and *PC managed nodes.* A managed node is a client that runs the full Tivoli framework code including the client portion of the distributed database; whereas, PC managed node runs an agent that communicates with the PC managed node object on a managed node. Since managed nodes have a complete implementation of the framework, they can run all the Tivoli applications, but they require a considerable amount of system resources. On the other hand, PC managed nodes have less resource requirements, but they can only run a subset of Tivoli applications.

With Tivoli V3.6, Tivoli introduced a three-tiered architecture which completely overcomes the limitations of previous managed node and PC managed node concepts. The new architecture provides for lightweight clients that provide full function as a managed system.Three new object types serve as the backbone of this architecture:

- *Endpoints*
- *Endpoint Gateways*
- *Endpoint Managers*

Endpoints are typically installed on systems that are considered *managed only* systems. That is, like most end-user workstations, these systems will be managed, but they will not be involved in the management of other nodes. Endpoints run a small daemon called the TMA (Tivoli Management Agent) that is responsible for executing methods at the request of a managing system. When a TMA is installed, a minimal number of files are installed on the managed system. When an application invokes a method to be executed on the endpoint, the method is automatically downloaded to the endpoint and executed by the TMA. Methods that are downloaded to the endpoint are cached at the endpoint. As long as that method stays in the cache, it need not be downloaded again upon a second invocation of the same method.

The *Endpoint Gateway* is a component that runs on a full managed node enabling the managed node to operate as a gateway between a cluster of endpoints and the rest of the TMR. Each TMR can have multiple endpoint gateways. Currently, one TMR server can support about 200 endpoint gateways. By using gateways, the performance and scalability of the Tivoli environment is greatly enhanced since gateways perform much of the function that was handled by the TMR server previously.

The *Endpoint Manager* controls and stores the association between endpoint gateways and endpoints. The endpoint manager component resides in the TMR server. The endpoint manager plays a role in enforcing site-specific policies. For example, policies may be put in place that specify which endpoint gateway will be assigned to new endpoints joining the network.

We refer the reader to the product documentation referenced in the bibliography for more detailed information on the Tivoli Framework.

It is most important to understand that, if the TMR server experiences a failure, then continued management of the systems within the TMR is severely limited until the TMR server has recovered. Therefore, the TMR server is a prime candidate to become a resource managed by an HA product.

The TMR server is not the only Tivoli component that should be considered a candidate to be an HA resource. Application servers that have been traditionally included as HA resources are also prime candidates as Tivoli managed nodes or endpoints. Therefore, it is important when an application server fails and is restarted on another node in the HA cluster, that the Tivoli managed node or endpoint function also fails over. By doing this, the Tivoli administrators will still see the application server as a single entity independent of which node in the HA cluster it is currently running on.

One of the core Tivoli applications is the Tivoli Enterprise Console (TEC). This application is the cornerstone of Tivoli's applications for managing the availability of critical resources. It provides a single collection point for events generated by a variety of systems. The TEC server can reside on a TMR server but is usually installed on a managed node separate from the TMR server. TEC requires access to a Relational Database Management System (RDBMS) and has its own daemons that are used to process the events and carry out automated actions. The TEC server is key to automating the management of your environment and, therefore, is also a candidate to be an HA resource.

We discuss the HA implementation of TMR servers, managed nodes, endpoints, and TEC servers as HA resources in this redbook.

1.2 High Availability Concepts

There are several HA products available in the marketplace. Though each has its own unique benefits, they all work on the same basic concepts. This section discusses HA concepts and provides definitions of terminology that will be used throughout the rest of this redbook. We can start with the definition of availability itself. Though it seems obvious, it is useful if we all start with the same definition in mind. Availability is simply the proportion of time that a system can be used for its intended purpose. An acceptable level of availability is different for every system in every environment, and in general, is dependent upon the cost associated with the system or application being unavailable for a period of time.

For example, an airline reservation system that is unavailable during peek business hours may cost the airline millions of dollars in revenue. Cost is associated with ensuring that a critical system has acceptable availability. High availability systems usually consist of redundant hardware and specialized software that may set dormant until a failure on the primary system occurs. It is up to the individual business to determine how much they are willing to spend to ensure an acceptable level of availability.

1.2.1 Levels of Availability

Most HA specialists discuss availability in terms of four levels with the higher levels encompassing each of the lower levels. However, it may be that not every availability feature at a lower level is implemented by a higher level implementation. This is because other capabilities of the higher level replace the requirement for the features at the lower level. For example, a system doesn’t need redundant power supplies if the application can be taken over to another system with an acceptable downtime of the application. For a more detailed discussion of HA concepts, please see your HA product documentation.

- One system with reliable hardware and software
- Recovery procedure in place in case of failure
- Planned system outages for maintenance
- Unplanned system outages

1. Improved Availability
   - Hardware redundancy, such as disk mirroring or RAID 5, uninterruptable power supply (UPS) and so on
   - Critical hardware backup on hand (power supply, cooler, fan, and so on)
   - Data journaling and check summing
   - Hot pluggable devices (disk, adapters, and so on)
   - Faster recovery
2. High Availability
   - Eliminate single point of failures
   - At least two systems (high availability cluster) with cluster software
   - Transfer of service to another machine
   - Minimal recovery time for unplanned outage
3. Continuous Availability
   - 100 percent availability to the end-user
   - Redundancy in all components
   - Provide ability to perform all error recovery and change processes online

If it's not already clear, this redbook is addressing the third level (High Availability) in relationship to Tivoli.

1.2.2 Single Point of Failure in a Cluster

In an HA environment, one of the key techniques is to eliminate single points of failure. This means that if one part of your system fails, the system can continue to run, or recover in an acceptable period of time, with minimal impact to the end-users. Only in a case where a second error occurs before the first is fixed should a more prolonged outage occur.

These components that are typically considered single points of failure include:

- Individual processors or nodes in a cluster
- Disks
- Adapters, controllers, and cables used to connect the nodes to the disk
- Network adapters attached to each node

- Network backbones over which the users are connected to the cluster
- Asynchronous adapters
- Application programs

Most HA products provide for monitoring of these resources and dynamically utilizing redundant hardware in the case of a failure.

1.2.3 High Availability Terminology

The following list contains common terms and definitions used when describing most HA products and environments. Each operating environment and HA product uses slightly different terminology for the same concepts. This list is intended to be a general representation of the concepts and is defined in such a way that you should be able to map these terms to those used in the environment for which you are most familiar. These terms are used throughout this redbook when discussing HA solutions in each of the environments we cover. The definitions mostly describe cluster resources or cluster configuration concepts:

cluster	A cluster is a group of machines (nodes) with applications that are managed and controlled by HA software. The systems in a cluster typically provide redundancy for one another.
node	A node is a machine in a cluster.
resource	Resources are applications, services, or components running on the cluster. A resource is the smallest configurable component in a cluster. Examples of resources include applications, disks, files systems, or TCP/IP addresses.
resource group	To run successfully, some resources are dependent on other resources. Dependent resources are grouped together in a resource group.
hostname	The hostname of a machine is the name of a machine. To find out a hostname in a Unix environment, you use the command `uname -n`. In some Unix environments, you may also have the `hostname` command, which returns the hostname of the machine. Normally, a hostname is bound to an IP label of a network

	adapter providing the base IP address for this machine.
IP address	This is the basic Internet address associated with a network adapter.
IP label	An IP label is the name associated with a network interface of a network adapter.
logical IP address	A logical IP address is another IP address for a network interface of a network adapter. Therefore, each interface has one IP address and possibly one or more logical IP addresses.
alias	You can have more than one name associated with an IP address. These names are aliases. Aliases can be useful for differentiating the name used for different functions. For example, you may create an alias by prefixing your IP label with `www` to represent Web server function.
hardware address	This is the address of a network adapter. Every network adapter has a burned-in hardware address. This address is universally unique. Some network adapters support configurable hardware addresses, which are called logical hardware addresses.
mutual takeover	Mutual takeover is a cluster configuration where every member in the cluster may be running its own set of applications during normal operation but where every member has the ability to take on the function of the other members should a failure occur.
idle or hot standby	An idle standby configuration is a cluster configuration in which one node is running a resource group, and the other node is idle. The idle node stands by and is only put in production if the other node fails. This is often called hot standby mode.
concurrent access	Concurrent access is a disk configuration where more than one machine can directly access the same physical disk at the same

	time. For concurrent access, a lock manager is needed to control the access to the disk.
mirrored disk solution	A cluster with a mirrored disk solution uses real-time mirroring of data between the members of the cluster. The data is stored locally on each member of the cluster. Every node has its own exact copy of the data.
shared disk solution	A cluster with a shared disk solution uses an external disk to which every member of the cluster is connected. Only one node has ownership of the disk at a time. In case of failure, the ownership is switched to another node.
cascading resource groups	A cascading resource group has a hierarchy of nodes where a certain resource group should run. If a higher priority node rejoins the cluster, the resource group will be taken back to this node automatically.
rotating resource groups	A rotating resource group has no hierarchy of nodes where a certain resource group should run. If a node rejoins the cluster after a failure, the resource group will not be taken back to this node.
failover	This is the process of moving one or more resource groups to an alternate node in the cluster when a failure occurs.

1.3 Summary

This chapter provided a short overview of Tivoli, the reasons why you may want to implement Tivoli in an HA environment, and an overview of HA concepts.

In the next chapter, we describe the specific environments we tested and document in the remainder of this redbook.

Chapter 2. Failover Scenarios

Before we start addressing the technical issues involved with implementing Tivoli in HA environments, let's first define the scope of our objectives. We cannot address all HA products on all Tivoli supported platforms or take into account all Tivoli applications.

This chapter discusses the specific scenarios and environments we address throughout the rest of the redbook. We believe that the scenarios we have chosen will have the most widespread applicability. You should be able to extend the concepts and information provided here to environments and scenarios not explicitly discussed.

2.1 Operating Environments

The specific environments we were able to test during the creation of this redbook were IBM's AIX V4.3.2, Sun's Solaris V2.5.1, and Windows NT 4.0. We are not aware of any factors that would prevent successful implementations in other Unix environments, such as HP-UX.

2.2 Tivoli Configurations

When looking at the implementation of Tivoli in HA environments, the first consideration is the Tivoli framework. A specific system can be installed as a TMR server, managed node, or endpoint. In addition, the managed node may take on additional roles, such as an endpoint gateway or TEC server.

The TMR server is a critical element of the TMR and, therefore, is an obvious candidate to be implemented in an HA environment. Assuming that the network itself is available, the availability of the TMR server ensures that management operations can be carried out on active managed nodes.

Obviously, the primary motivation for increasing the availability of TMR servers is to help ensure that you can manage critical systems in your network. Managed systems will not only include end-user systems but will also include application servers and other systems critical to the day-to-day operations of your business.

These application servers are also potential candidates to be implemented as HA nodes. However, if these nodes will also be managed by Tivoli applications (for monitoring, software distribution, and so on), then they must also be Tivoli managed nodes or endpoints. If the application server fails over to another node in the HA cluster, then the Tivoli managed node or endpoint function should also fail over. Though system administrators may need to be alerted to the fact that a failover has occurred, they should not have to be aware of which physical node the application server is currently running on to perform other management functions. Therefore, the failover of a managed node or endpoint should be mostly transparent to the Tivoli framework.

Based on the above discussion, it is clear that we need to demonstrate how to fail over TMR servers, managed nodes, and endpoints.

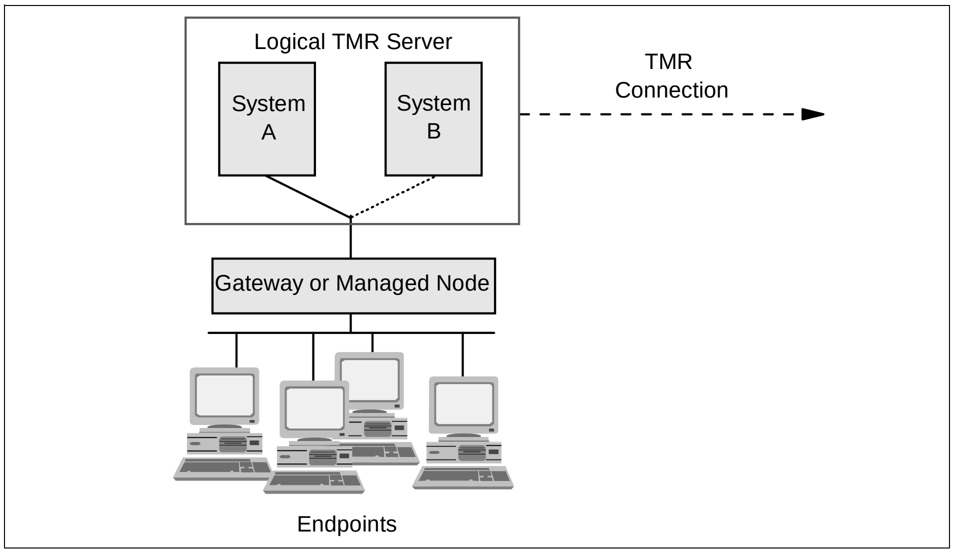


Figure 1. TMR Server Failover - Transparent to Managed Systems

In the above figure, we show two physical systems that are both capable of taking on the role of the TMR server. The actual system that is currently running the TMR server should be hidden from the managed systems and any interconnected TMRs.

Likewise, the following figure shows the failover of a managed node and how it should be transparent to the TMR server

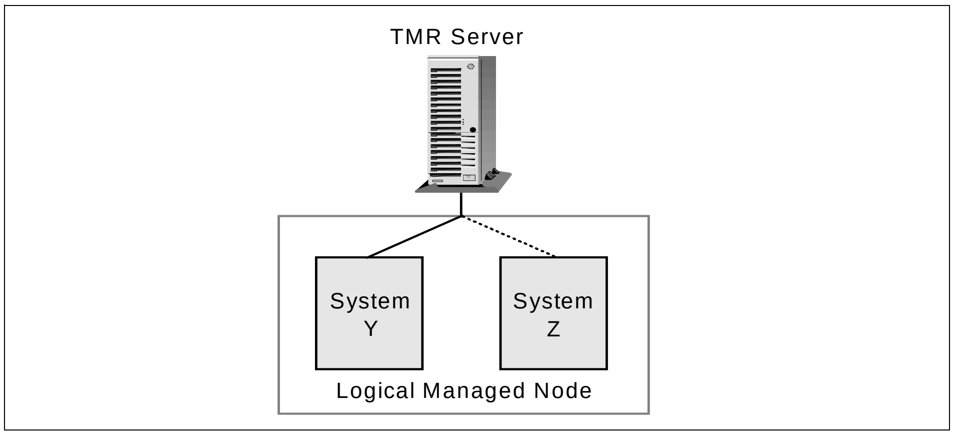


Figure 2. TMR Managed Node Failover - Transparent to TMR Server

Tivoli Management Agent endpoints, which were introduced by the Tivoli 3.6 release, also should be considered as a highly available resource since they may be used to manage an application server. The following figure shows the failover of an endpoint and how it should be transparent to the TMR server and endpoint gateway.

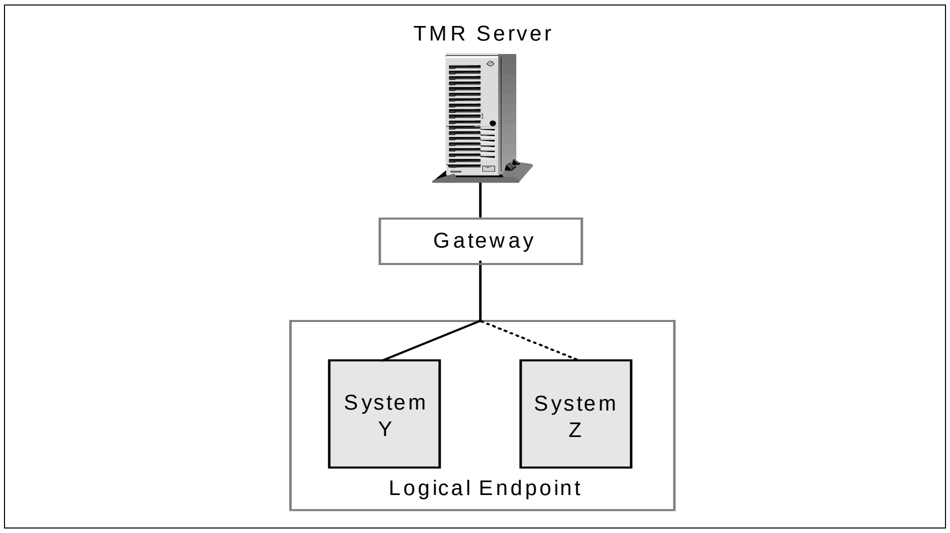


Figure 3. TMR Endpoint - Transparent to TMR Server and Gateway

In addition, the Tivoli Enterprise Console (TEC) is a core Tivoli application that is used to manage the availability of systems in a Tivoli environment. It helps manage availability by providing the reception, consolidation, and automated handling of events generated by a variety of systems and applications throughout the network. Administrators use TEC to monitor the overall status of the network and ensure that problems are resolved possibly even before users become aware of them. Therefore, the TEC server is also a critical system to consider implementing as a resource in an HA cluster as in the following figure.

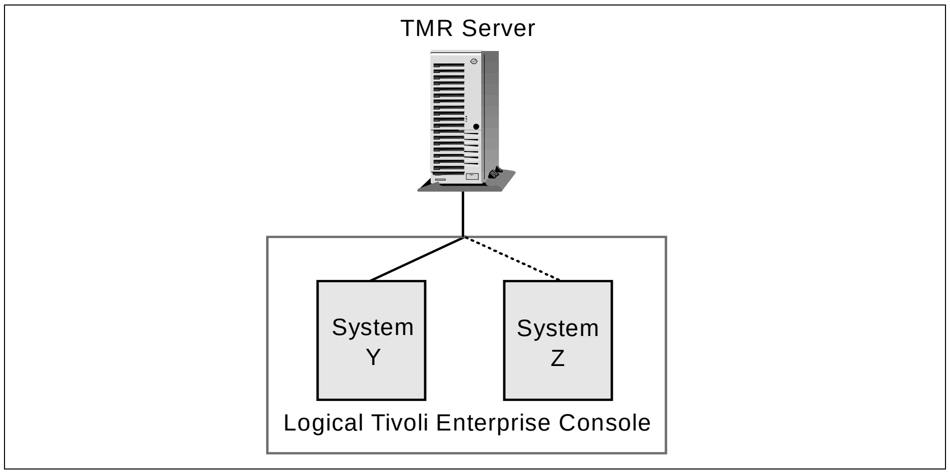


Figure 4. Tivoli Enterprise Console - Transparent to TMR Server

In summary, the four Tivoli configurations that we will specifically address are:

- Tivoli TMR server
- Tivoli managed node
- Tivoli endpoint
- Tivoli Enterprise Console server

In each of our scenarios, we are utilizing version 3.6 of the Tivoli framework and applications.

2.2.1 Scenarios Not Covered

Though our scenarios covered TEC, and some of our testing included ensuring that applications, such as Tivoli Distributed Monitoring, continued to work after a failover, we did not do extensive testing of specific applications or application scenarios. In fact, we basically ensured that the framework could be restarted successfully by the HA product. We did not address the restarting of transactions that may have been in progress when the failover occurred. For instance, if a failover occurs during a software distribution, nothing we have done will allow the software distribution to pick up where it left off and complete the distribution successfully.

We do see this as an important factor in implementing a true HA environment. Complete transaction recovery and restart is an application issue. That is, each Tivoli application must determine how it can take advantage of the underlying transaction subsystem within the Tivoli framework to provide recovery in failover conditions.

Until this is fully addressed by all of the applications, the best we can do at this point is to ensure that Tivoli system administrators be made aware that a failover has taken place. Procedures can be put in place to help administrators understand the state of a transaction at the time of the failover and what they can do to back out or restart the transaction. It would be quite straightforward in the HA environments we tested to be able to generate a TEC event in the case of a failover. The appropriate administrators could then be notified to carry out the procedures to determine the status of any management operations that were in progress at the time of the failure.

Also, we didn't cover the gateway failover scenarios for the following reason: The primary purpose of gateways are to manage endpoints on behalf of the TMR server. If for any reason a gateway goes down, the endpoints associated with that specific gateway can automatically switch to another gateway that may well have a hostname and IP address different from the original gateway. Endpoints can be configured for how long they should wait when they lose contact with their gateway before looking for another gateway. So, we can say, HA capability is already embedded into the gateway architecture.

2.3 High Availability Products and Configurations

During the project that resulted in this redbook, we worked with three different HA products:

- IBM HACMP V4.3.0 on AIX
- Sun Solstice HA 1.3 on Solaris
- Microsoft Cluster Server on NT 4.0

As it turned out, the steps required to implement Tivoli in each of these environments was similar. The key differences centered around how the IP addresses through which connections were made was implemented. The differences between the different implementations are detailed in later chapters. We have a high level of confidence that similar steps could be taken in other HA environments with similar results.

With most HA products, including those we tested, there are various options as to how the cluster is configured. Though there are several variations, depending on the specific HA product, the two configuration types most often discussed are idle or hot standby and mutual takeover. We described these briefly in 1.2.3, “High Availability Terminology” on page 7.

Basically, a hot standby configuration assumes a second physical system capable of taking over for the first. This second system is not utilized except in the case of a failover. The mutual takeover configuration consists of two systems, each with their own set of applications, that can take on the function of the other in the case of a failure. In this configuration, typically the applications on both systems will run in a degraded mode after the takeover, since one system is doing the job previously done by two. Mutual takeover is typically a less expensive choice in terms of hardware costs since it avoids having a system installed that is only used when a failure occurs.

With the level of Tivoli that we are using, version 3.6, support is now provided to run multiple oserv processes on a single system. This function is currently supported only for Unix environments. This issue will be addressed for NT environments in a future release of Tivoli. In Appendix A, you can find a brief overview of the implementation details of this capability. For our project, we have built clusters for both hot standby and mutual takeover configurations. A highly requested configuration is one in which the TMR server and a managed node are configured in a cluster for mutual takeover. Specifically, a TMR server and TEC server configured in this way would be very desirable to help ensure manageability of your environment.

2.4 Summary

The following chapters describe methods and techniques for implementing Tivoli servers, managed nodes, endpoints, and TEC servers in various HA environments. The specific HA products that we have used and document in the following chapters are HACMP, Solstice, and Microsoft Cluster. Though the only two Unix operating environments we tested were AIX and Solaris, we have confidence that the information presented here will be applicable, or at least, easily adapted to other environments, such as HP-UX.

In the next chapter, we describe the resources used by Tivoli that you must be aware of to successfully implement Tivoli in an HA environment.

Chapter 3. Tivoli Failover Considerations

In this chapter, we describe the aspects of the Tivoli software that must be considered when planning an implementation of Tivoli in an HA environment. The information presented here is generally independent of the platform on which you will be running Tivoli. In the cases, where information is unique to a specific platform, it will be noted.

The HA product-specific configuration steps described in later chapters refer to the information presented here.

3.1 High Availability Considerations for Tivoli Framework

The first topic we discuss is the Tivoli framework. Information pertaining to TEC is discussed in a later section.

In the HA implementation of any application, you must understand what resources and facilities are used by the application. For instance, the typical information that must be gathered for any application before configuring the HA product to support it includes:

- Resources used by the application, such as files and network facilities
- System configuration requirements
- Information regarding how to start and stop the application as well as how to test whether or not it is currently running

For the HA purist, the above list is incomplete but certainly gives us a good place to start.

3.1.1 Files and File Systems

When you install a TMR server or managed node using the default settings, files are created and/or modified in:

/usr/local/Tivoli	This directory tree contains the binary files required to run the Tivoli environment. In general, system- or TMR-specific information is not stored in this directory tree. Therefore, it is relatively static with one exception described below.
/var/spool/Tivoli	This directory tree contains the TMR database. It is dynamically updated during the operation of Tivoli, and its availability and consistency is critical to the on-going operation of Tivoli. In an HA environment,

	the file system containing this tree should be located on a shared disk whose ownership can be taken over by another node in the cluster when a failure occurs. In this way, the node that has taken over for a failing node has full access to the Tivoli database.
/etc/Tivoli	When Tivoli is installed, several files customized for the specific environment are placed in this directory. For instance, the setup_env.sh file that sets all the appropriate environment variables for proper Tivoli operation is included in this directory. As we will mention later, the Tivoli Enterprise Console also places customized files within this directory tree by default. The number and size of the files in this directory are quite small and they are typically not modified after they are created. Therefore, you can easily duplicate this directory across nodes in your cluster.
/usr/lib/X11/app-defaults	Any required resource files used by the X-based user interface for the Tivoli Desktop are stored in this directory. Once Tivoli is installed on the first node in a cluster, the files added by the Tivoli installation can be copied to the same directory on the other node(s) in the cluster.
/tmp	Tivoli generates some log files in the /tmp directory. These are typically created during installation, and, in general, you need not worry about sharing or duplicating the files in this directory for proper operation of Tivoli.
/opt/Tivoli/lcf	This directory resides in Unix endpoints. It contains binaries and configuration information of the endpoint in various subdirectories. It is possible to install the whole directory structure on a shared disk as we did in our scenario. For NT 4.0 endpoints, the endpoint directory resides in the following path: %SystemDrive%\Program Files\Tivoli\lcf

In addition to the directories listed above, some standard system files are modified by the installation of Tivoli. When setting up your HA environment, it is important to understand these modifications so that they can be replicated while setting up alternate nodes in the cluster. The following files are modified on an AIX system. Similar modifications are made in other operating environments.

/etc/services A port for Tivoli's udp and tcp communications is added to this file. Below are the lines that are appended to the file.

```
#
# Tivoli framework daemon
#
objcall        94/tcp          # Tivoli daemon
objcall        94/udp         # Tivoli daemon
```

/etc/rc.nfs The following commands for starting the oserv daemon are added to the `rc.nfs` file. Normally, this file is called at system start through the /etc/inittab file. During installation, an option is presented related to whether the oserv daemon should be automatically started when the system is rebooted. For an HA environment, you do not want the oserv daemon started through this process. Rather, it is imperative that the HA product be responsible for starting the oserv daemon. If during installation you chose to have the oserv daemon automatically started on system restart, you should edit the rc.nfs file and comment out the following lines:

```
#
# Start the Tivoli daemon
if [ -f /etc/Tivoli/oserv.rc]; then
/etc/Tivoli/oserv.rc start
echo "Tivoli daemon started."
fi
```

/etc/inetd.conf An entry for starting the Tivoli Environment is added. This entry is activated through a call to the inetd superdaemon and is used by the Tivoli remote start process.

```
# Tivoli Framework daemon
objcall dgram udp wait root /etc/Tivoli/oserv.rc \
/etc/Tivoli/oserv.rc inetd
```

3.1.2 Placing Files on Local or Shared Disk

The convention in HA clusters is to place binaries relating to shared applications on a local disk. This is commonly done by installing on one system, deleting the data from the shared disk, then reinstalling on the other node after making the shared disk available to the other system. This is normally done for three reasons:

1. If a product is installed with the binaries on the shared disk, and the product fails to stop cleanly, then orphan processes can be left attempting to access the binaries. This can result in a failure when attempting to unmount the file system containing these binary files. If this file system is

required to be made available to the other system, this can cause a failover to fail.

2. There can be an issue regarding licensing. If a product is licensed on a per system basis, then placing the binaries on a shared disk and, thereby, making them available to more than one system can violate the licensing restrictions.
3. Maintenance can be made more difficult if the binaries are placed on a shared disk. If you perform an upgrade on a system, then having a copy of the older versions on the secondary system can be helpful if you need to back out the changes quickly. Also, the system may keep a record of what levels of software are installed on it. If the binaries are on a shared disk and are upgraded on one system, the second system will have incorrect information regarding system levels.

With Tivoli, the binaries themselves are static, but within this directory tree are the binaries for Tivoli Tasks. These are downloaded from the TMR server and need to be made available to the system running the service. To allow the Task binaries to be made available on the shared disk and keep the normal Tivoli binaries on local disk, the script in Appendix-B can be used. We, however, made the decision to place the binaries onto the shared disk. We did this mostly for simplicity but also for the following reasons:

1. The oserv process that is central to Tivoli opens the database and leaves it open for the duration of its life. Therefore, the fact that the binaries may be left in use should the process exit uncleanly is not important and the database that is required for the oserv process to operate will have the same problem. By placing the binaries on the shared disk, we have made the issue no worse.
2. The installation process used by Tivoli does not limit or prevent you from installing the Tivoli binaries and database files in the directory of your choice. So, placing the binaries on shared disk to make them available to both systems is not a problem.

   Should an upgrade take place, then the Tivoli database itself would be altered. Therefore, returning to use older binaries without backing out the changes from the database could produce unpredictable results. Also, as the install process is designed for open systems, no attempt is made to update the system tables to reflect that the product has been installed. Therefore, no records are kept that could become out of step through placing the binaries on shared disk.

With our configuration, each system has its own copy of the binaries and data in its own unique sub-directory structure. As failovers occur, the file systems containing these directories can be moved to the standby system. By using an appropriate naming convention for the directory structure, it is easy to track what system has which service.

3.1.3 /etc/Tivoli

As we have discussed, we have made the decision to place a complete, distinct image for each instance of the oserv on separate shared disks. This is made possible by the flexibility of the Tivoli install process. Either through the GUI frontend or through the command line, you can specify where you wish to place the various binaries that are required and the database for the oserv process. However, as part of a Unix install, Tivoli also creates a directory called /etc/Tivoli. Although there is a variable called *EtcTivoli* that you can set before commencing a server install to alter the location of these files, a number of Tivoli processes require the files to remain in /etc/Tivoli as the location is hard coded and not altered by the use of the EtcTivoli variable. For the most part, the files in /etc/Tivoli are not customized for the particular instance of oserv that is being installed and are only ever read from, not written to.

You can handle this directory in one of two ways. Either the /etc/Tivoli directory tree can be manually copied to the standby system after initial installation, or the /etc/Tivoli mount point could be on a shared file system.

In a mutual takeover environment, the install of the second oserv on the second system will create the /etc/Tivoli directory. Though you can copy these files from one system to another in a variety of ways, the following command will allow you to do this in a single step.

```
tar -cvpf - /etc/Tivoli | rsh <target host> ’( tar -xvpf -)’
```

As mentioned previously, for the most part, the files and links present in this directory are not customized for a particular install of the oserv process and are only read from. The exceptions to this are the following:

- setup_env.sh
- setup_env.csh
- oserv.rc
- The ./tec directory
- The ./tecad directory

The setup_env files are used to configure the environment so that the user or calling script can interact with a particular instance of the oserv process. The oserv.rc file is used to start, stop, or restart a particular oserv instance. The

./tec and ./tecad directories are used by the application called TEC and will be discussed later.

As different setup_env and oserv.rc files are needed to interact with different oserv processes, we copied these to the shared disk and then called them directly with our start and stop scripts. It is necessary to leave behind a copy of the files as they are used by Tivoli's remote restart procedure.

3.1.4 Hostname Resolution

When Tivoli is installed, the oserv binds itself to the hostname of the system on which it is being installed, and this information is stored within the database. If the hostname of the system is changed later, the oserv daemon will fail to start. This means that, if you move the TMR server to another machine, you also have to move the hostname to this machine. Since Version 3.2 of the Tivoli framework, it is possible to set an environment variable called WLOCALHOST to define a logical hostname to be used in place of the actual hostname. When installing the TMR server, this variable should be set to the name associated with the service interface (IP label) of the cluster before the installation of Tivoli. The service interface must be up and running during installation.

During installation, whether using the GUI or the command line interface, you will need to specify the name of the server to be the same as that specified by the WLOCALHOST environment variable. Otherwise, the oserv will bind to the actual hostname (the name returned by the `hostname` command.) If this occurs, you will not be able to move the Tivoli node to an alternate system, giving the following error:

```
1999/02/23 11:11:18 +06: $Database mismatch (from
brown.itsc.austin.ibm.com/9.3.187.134)
1999/02/23 11:11:18 +06:
!/Tivoli_tmr/usr/local/Tivoli/bin/aix4-r1/bin/oserv: odlist init failed.
internal resource corrupted. (54)
```

If you receive this error, then you can resolve the situation by setting the WLOCALHOST variable to that which the server expects. This can be done by editing the /etc/Tivoli/setup_env.sh and /etc/Tivoli/oserv.rc files. Rather than setting the WLOCALHOST environment variable, it is also possible to get the same effect by creating a file called /etc/wlocalhost. This file should simply contain a single line identifying the desired hostname.

In a mutual takeover environment, however, multiple oserv processes may be running on a single system. Each instance of the oserv may require its own particular value for WLOCALHOST. In this situation, you would not wish to use the /etc/wlocalhost file as this variable would then be system wide. Setting the WLOCALHOST variable only effects the shell in which you are running and processes started from that shell.

3.1.5 Interconnected TMRs

Interconnected TMRs do not actually add any additional considerations to an HA environment. Information about the other TMRs to which it has been connected is stored in the TMR database. When the TMR server is restarted, it will attempt to reconnect to any other TMR servers to which it had previously been connected.

Since all of this occurs after the oserv daemon has successfully restarted, there is nothing that should prevent this from occurring. We tested both one-way and two-way interconnections in each of our HA environments with great success.

In the case where a TMR server has failed over abruptly, you may want to execute the following two commands that check the consistency of the TMR database and update the resources between TMRs, respectively:

```
wlsconn -u <region-number of connect TMR>
wchkdb -ux
```

Whether you automate the execution of these commands or simply execute them as part of your standard procedures after a failover has occurred is up to you.

3.2 TEC Considerations

In this section, we look at the considerations for implementing a TEC server in an HA environment. While the oserv daemon can be installed and run in a relatively standard Unix environment, the TEC server has several additional dependencies that make its HA implementation a bit more complex.

We assume the reader has a thorough understanding of TEC, and we will not discuss the various TEC components and structure in detail.

The following figure represents the dependencies for the various TEC components. (Please note that portmapper is not a dependency on an NT TEC server.)

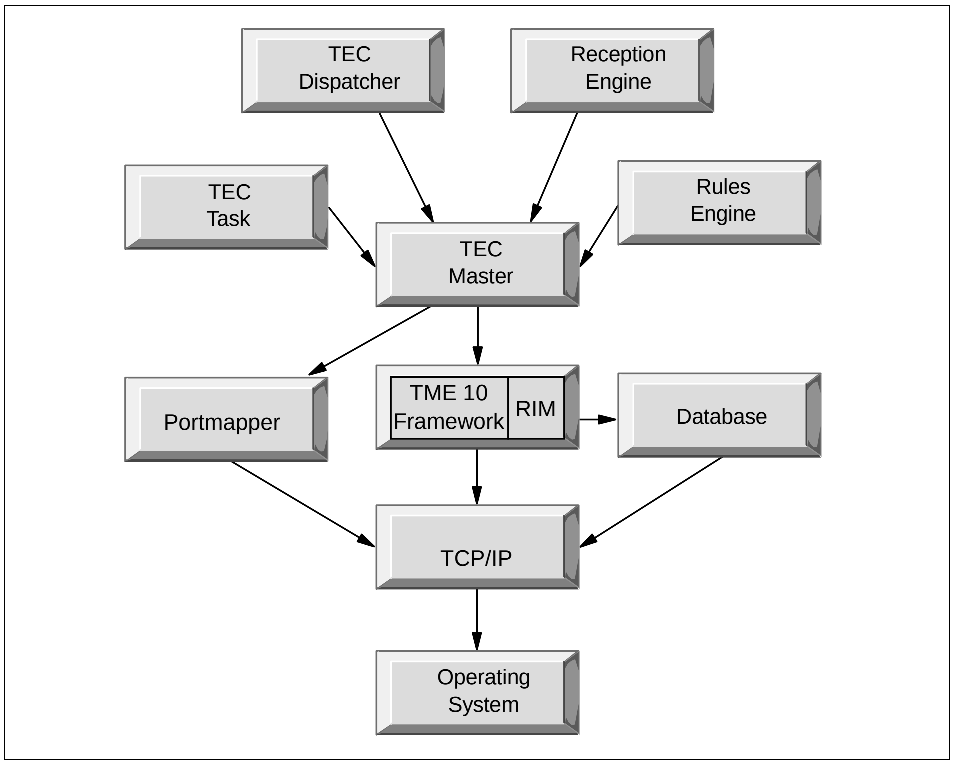


Figure 5. TEC Process Dependencies

The TEC master process is the core of the TEC server. It has a dependency on the portmapper daemon, the Tivoli framework itself (either TMR server or managed node), and a relational database. It actually uses the Relational Database Interface Module (RIM) that is part of the framework. However, the framework has no dependency on the database. The dependency is created when applications, such as TEC, are installed and started.

RIM allows you to use the relational database of your choice. Currently, the databases supported by RIM include Oracle, Sybase, Informix and DB2/6000, although at the time of the writing of this book, DB2/6000 was only supported by the inventory application, not TEC.

The database can be installed locally or can be accessed remotely. Therefore, to implement TEC in an HA environment, the database component (the database itself or its distributed client) must also be included as an HA resource. We do not discuss in detail the HA implementation of the database products in this redbook. Refer to your database or HA product documentation for information on how this can be implemented. You may want to refer to the redbook titled *Bullet-Proofing Your Oracle Database with HACMP: A Guide to Implementing AIX Databases with HACMP,* SG24-4788.

Aside from the TEC master process, there are four other processes associated with TEC that you need to be aware of. These include the reception engine, the rules engine, the dispatcher, and a TEC task process.

The main process we need to be concerned with is the TEC master process. If we can get this to failover and restart on another node, the other four processes should start with no additional dependencies.

The implementation details of the TEC server are explained in the next chapter.

3.3 Managed Node Considerations

There are only a few considerations related to managed nodes, over and above those described earlier in this chapter applying to TMR servers. When you install a managed node, the service interface should be up, and the shared filesystem should be mounted. If possible, install and configure the HA product first and bring the resource group online. After installing, you will have to synchronize the same files and file systems as we have already described.

You don’t have to worry about tasks on managed nodes since the tasks are stored only on the TMR server. If you execute a task on a managed node, the task is copied from the TMR server to the managed node’s /tmp directory and executed from there. Of course, if a task is in the process of being executed at the time of a failover, it may end prematurely and will not be restarted automatically on the alternate node.

3.3.1 Distributed Monitoring Scenario

The Tivoli Distributed Monitoring application allows you to distribute monitors to managed nodes that are periodically executed and report back to the server when thresholds are reached. The monitors and information about them are distributed as profiles to managed nodes and stored in the Tivoli database on the managed node.

When a managed node that has the monitoring engine installed on it is restarted, its monitoring engine is automatically restarted and will read the information it requires from the database to execute the appropriate monitors. Since all of the information required to do this is in the database on the managed node, we should not expect any problems in having monitors restart after a failover.

The only steps that must be taken to ensure this happens is to synchronize the /etc/Tivoli directories after the installation of the distributed monitoring engine on the managed node.

In our environments, we did go through the exercise of defining and distributing monitors and verifying that they restarted after a failover on the managed nodes. Also, we tested the distributed monitoring scenario on endpoints after the mutual failover in section 4.8.7.

In the following section, we describe a possible technique for running multiple oservs on a single system. This would allow for a more complete mutual takeover environment.

3.4 Mutual Takeover Environments

In the past, mutual takeover environments for Tivoli were not possible. This was because the oserv process listens on a given port, which is by default 94. Should a second oserv process attempt to start and operate against the same port number, then you would get the following error:

```
1999/02/24 11:27:04 +06: Tivoli Framework (tmpbuild) #1 Tue Jul 21 11:04:01
CDT 1998
Copyright Tivoli Systems, 1997.  All Rights Reserved.
TMR 1669180253. ORB 2. TMR server brown.itsc.austin.ibm.com:94. Port 94.
pid 10312
1999/02/24 11:27:46 +06: ^port 94: In use.  Waiting...
1999/02/24 11:28:06 +06: ^port 94: In use.  Waiting...
1999/02/24 11:28:26 +06: ^port 94: In use.  Waiting...
```

The process listing would show both the oserv processes, but eventually the second oserv would timeout and would fail.

With Tivoli Version 3.6, a new variable was added, *set_force_bind*. By default, this variable is set to false. In this state, an attempt to start a second oserv would still fail. To set the variable, the following Tivoli command must be issued after the environment has been set up through the use of setup_env.sh:

```
odadmin set_force_bind TRUE {od number}
```

This command should be run against all the oservs that may be required to run on the same system in the future. After the command has been run, you need to restart the oserv process for the change to take effect. Once the oserv has had *set_force_bind* set to *true*, then it only binds to the IP address that matches the WLOCALHOST variable. As each instance of the oserv process will be binding to a different IP address, then you can start multiple oserv processes on the same system. You need to use separate setup_env files and individual oserv.rc files for each instance of the oserv being started. Once started, running the corresponding setup_env.sh will allow you to interact with the oserv process.

It is also possible to set the value of the set_force_bind variable at install for managed nodes. It is not possible to pre-configure the TMR server at install time, but then it is a simple matter to set the variable after installation is complete. For some platforms, it is necessary to install the oserv process with this variable set, but we shall discuss these issues in later sections. You can not use the GUI-based install to perform this action; it is only possible from the command line with the following command:

```
wclient -c {cdrom directory} {path variables} @ForceBind@=1
```

3.5 Endpoint Considerations

Implementing an endpoint as a highly available resource shouldn't be a more challenging task than a managed node. As we have mentioned before, endpoints place their configuration and binary files under the /opt/Tivoli/lcf directory for Unix endpoints and to %SystemDrive%\Program Files\Tivoli\lcf directory on NT 4.0 endpoints by default. It is possible to put this directory on the shared disk.

One point of consideration is: Each time an endpoint is installed, regardless of whether installation is successful or not, it also creates a new subdirectory of /etc/Tivoli/lcf. This subdirectory begins with a number that matches the number of install attempts to this system. For example, the first install will create /etc/Tivoli/lcf/1, the second attempt will create /etc/Tivoli/lcf/2, and so on. To overcome this situation, you should delete an endpoint from the endpoint manager before attempting a new installation. If /etc/Tivoli is on a local disk, you need to copy these files to a shared disk for an HA implementation.

3.6 Summary

This chapter discussed all of the major considerations we found in implementing Tivoli in an HA environment. We specifically addressed the TMR server, managed node, and endpoint, as well as TEC server. In the next three chapters, we walk you through our actual implementation steps using

HACMP, Solstice, and NT. Each of these chapters refers back to, and assumes knowledge of, the material covered in this chapter.

Chapter 4. High Availability Cluster Multi-Processing (HACMP) 4.3.0

HACMP for AIX Version 4.3.0 is a control application that can link up to 32 RS/6000 servers or SP nodes into a single highly available cluster. Clustering servers or nodes can enable parallel access to their data, which can help provide the redundancy and fault resilience required for business-critical applications.

HACMP clusters can be configured in several modes for different types of processing requirements. The *concurrent access mode* suits environments where all of the processors must work on the same workload and share the same data at the same time. In the *mutual takeover mode*, the processors share the workload and back each other up. *Idle standby* allows one node to back up any of the other nodes in the cluster.

4.1 HACMP and AIX Configuration

The specific hardware and software configuration we used in our environment are described in the following sections. We assume the reader is familiar with HACMP concepts.

4.1.1 Hardware Configuration

Our lab environment for using HACMP consists of the following:

1. One RS/6000 Model 520 with AIX4.3.2.0, with a hostname of *black.*
2. One RS/6000 Model 530 with AIX4.3.2.0, with a hostname of *brown.*
3. Two token-ring adapters on each system.
4. One serial connection between the two RS/6000s.
5. One 7137 RAID array with five, 1 GB disks, that are connected to both systems with SCSI Fast/Wide, single-ended connectors.

4.1.2 Diagram of the Hardware Configuration

We connected the given hardware in the following manner:

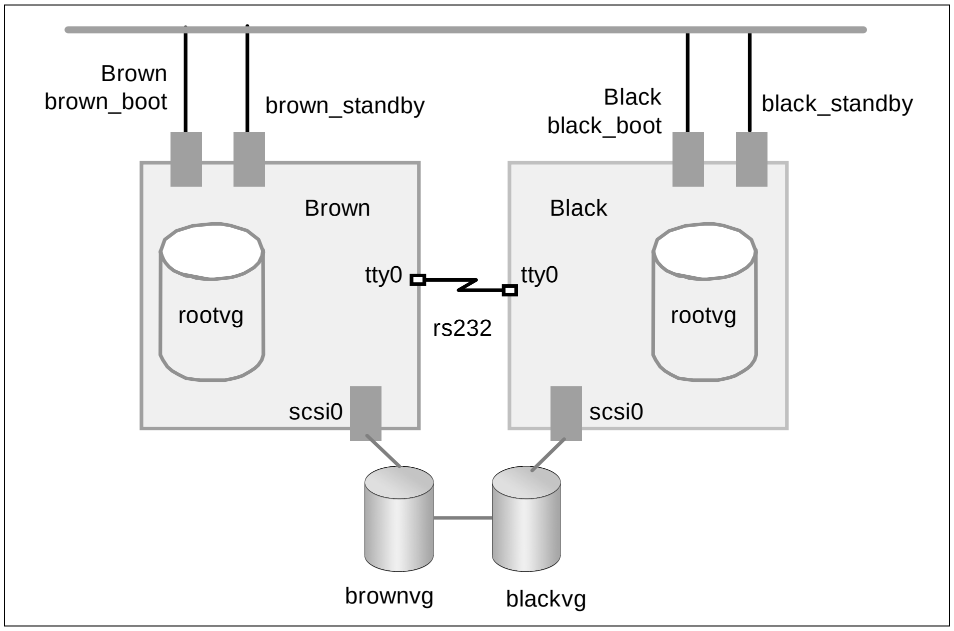


Figure 6. HACMP Hardware Configuration

4.1.3 File System Mount Points

In all of the following examples, we have used a format by which the file systems used by an application are identified by the file system name in the following manner:

- For TMR server, mount from /Tivoli_tmr.
- For managed node, mount from /Tivoli_mn.
- For TEC server, mount from /Tivoli_tec.

In the examples where a mutual takeover of like applications was used, then we distinguished the file systems by adding numbers, that is, /Tivoli_mn1 and /Tivoli_mn2.

4.2 Idle Standby of a Managed Node

In this section, we will describe how to build a managed node into a hot standby configuration using RS/6000s and HACMP. We will discuss the installation of the managed node *black* from a stand alone server. The managed node will be configured to run on the system *black* unless there is a failure. At this point, the managed node function will be restarted on the system *brown*. This will allow the managed node *black* to be seen by users and the TMR server alike with minimum down time. The switch over that has occurred between systems will be transparent to the TMR server and the end users.

4.2.1 File System and Disk Configuration

As can be seen from Figure 1, we created a volume group for each system. With this idle standby cluster, only the single volume group was required. We built the following file systems in the volume group blackvg.

Table 1. File Systems for Idle Standby of a Managed Node

Mount Point	Size (in 512 byte blocks)
/Tivoli_mn/usr/lib/X11/app-defaults/Tivoli	8192
/Tivoli_mn/usr/local/Tivoli	204800
/Tivoli_mn/var/spool/Tivoli	65536
/Tivoli_mn/etc/Tivoli	8192

4.2.2 Installation Process

To install a managed node in our cluster, we did the following:

1. We created the following start script:

   ```
   #!/bin/ksh
   . /Tivoli_mn/etc/Tivoli/setup_env.sh
   /Tivoli_mn/etc/Tivoli/oserv.rc start
   ```

   and the following stop script:

   ```
   #!/bin/ksh
   . /Tivoli_mn/etc/Tivoli/setup_env.sh
   /Tivoli_mn/etc/Tivoli/oserv.rc stop
   ```

   and placed these in the /usr/local directory in rootvg on both systems. We used chmod 755 {filename} to make the scripts executable.

2. We created a resource group called blackres in HACMP containing the four file systems and the service IP address of black. We configured the node relationship such that the resource would normally reside on the system black but would cascade to brown in the event of a failure. As this is a cascading resource, when black reintegrates into the cluster, the resource will be taken over again by black.

3. We then synchronized the changes to all the nodes and started HACMP on both nodes.

HACMP mounted all the necessary file systems and made the service IP address available. Next, we started the install of the managed node from the TMR server.

4. Once this had completed, we copied the files that had been placed in /etc/Tivoli into /Tivoli_mn/etc/Tivoli.
5. We had to update the /etc/services file on the system brown with the following:

   ```
   objcall94/tcp
   objcall94/udp
   ```

   as the install process had done this for us on black. The oserv process uses the port number 94 (94 being the default value) for communication between the managed node and the TMR server. If this change is not made on both systems, then the oserv process will fail to start as the port it needs would not be available.

Note

The name of the managed node should match the service IP label. For example, in our environment, we have matched the name of the managed node to the service IP label on the system black.

It is also advised that it should be fully qualified with the domain name, that is, `black.itsc.austin.ibm.com`. If a nameserver is in use, then you should be able to perform address resolution of both the IP address and IP label.

It is not necessary for the name of the managed node to match the system hostname, as the WLOCALHOST variable removes this requirement. In our environment, the system hostname is the same as the service IP label, but this is purely for simplicity.

4.2.3 The WLOCALHOST Variable

Provided that the TMR server had the WLOCALHOST variable set, then the setup_env.sh and oserv.rc files for the managed node will have the WLOCALHOST variable set by the install process; so, no further changes are necessary to these files. However, if the TMR server that was used to install the managed node did not have the WLOCALHOST variable set, then the following changes will be necessary:

1. In the setup_env.sh file, change the following:

   ```
   LIBPATH="${LIBDIR}:/usr/lib${LIBPATH:+:$LIBPATH}"
   export LIBPATH
   set +e; unset WLOCALHOST
   ```

to the following:

```
LIBPATH="${LIBDIR}:/usr/lib${LIBPATH:+:$LIBPATH}"
export LIBPATH
# set +e; unset WLOCALHOST
WLOCALHOST=black
export WLOCALHOST
```

where `black` is the hostname of the system on which the managed node normally runs.

2. If you are running with a csh environment, then you will need to make the same change in the setup_env.csh file.
3. In the oserv.rc file, change the following:

```
LIBPATH="${LIBDIR}:/usr/lib${LIBPATH:+:$LIBPATH}"
export LIBPATH
set +e; unset WLOCALHOST
```

to the following:

```
LIBPATH="${LIBDIR}:/usr/lib${LIBPATH:+:$LIBPATH}"
export LIBPATH
# set +e; unset WLOCALHOST
WLOCALHOST=black
export WLOCALHOST
```

4.2.4 Application Server

In order for HACMP to control the starting and stopping of the managed node, we need to configure our start and stop scripts into an *application server* within HACMP. This application server should then be added to the resource group. To make these two changes take effect, we will need to stop HACMP, on both nodes before we synchronize. At this time, HACMP will not issue any commands to stop the oserv process. Therefore, before stopping HACMP you should issue the following to stop the oserv process:

```
# . /etc/Tivoli/setup_env.sh
# /etc/Tivoli/oserv.rc stop
```

Now that the oserv process has been stopped, HACMP will be able to stop cleanly. We therefore stopped HACMP on both nodes in the normal *graceful* manner. After the stop had completed, we synchronized and verified the resources, using the following SMIT fastpath:

```
# smitty clsyncnode.dialog
```

Once the synchronization process had completed successfully, then we were able to restart HACMP on both systems. The oserv process started on black. We were able to see this by running the following commands:

```
# ps -ef | grep oserv
    root 14268     1   0   Feb 26      - 29:29
/Tivoli_tmr/usr/local/Tivoli/bin/aix4-r1/bin/oserv -p 94 -k
/Tivoli_tmr/var/spool/Tivoli/black.db
root 23848 19948   3 12:19:32  pts/3  0:00 grep oserv
# . /etc/Tivoli/setup_env.sh
# odadmin odlist
Region      Disp  Flags  Port            IPaddr   Hostname(s)
1669180253     1    ct-    94       9.3.187.136
black.itsc.austin.ibm.com,brown
               2 ct-    94       9.3.187.138   pooh.itsc.austin.ibm.com
```

Once these steps had been completed, we were able to failover from black and the oserv process started on brown. After the failover had completed, the TMR server was able to communicate with the managed node known to it as black.

4.3 Mutual Takeover of Managed Nodes

In this section, we shall deal with building two managed nodes into a mutual takeover environment from a stand alone TMR server. These managed nodes will be known as black and brown. Under normal conditions, each will run on its own system. Should either suffer a failure, then the remaining system will run both managed nodes. As before, this takeover of resources will be transparent to the TMR server and end users. Minimal down time will be seen. When both managed nodes are operating on a single system, this will result in two independent instances of the oserv process running on the same system. Much of the process has already been covered in Chapter 4.2, “Idle Standby of a Managed Node” on page 30.

4.3.1 File System and Disk Configuration

As can be seen from Figure 6 on page 30, we created a volume group for each system. In the same manner as was done for the idle standby configuration, each system needs to have its own file systems built on the appropriate volume group with the following sizes:

Table 2. File Systems for Managed Node Black

Mount Point	Size (in 512 byte blocks)
/Tivoli_mn1/usr/lib/X11/app-defaults/Tivoli	8192
/Tivoli_mn1/usr/local/Tivoli	204800
/Tivoli_mn1/var/spool/Tivoli	65536
/Tivoli_mn1/etc/Tivoli	8192

Table 3. File Systems for Managed Node Brown

Mount Point	Size (in 512 byte blocks)
/Tivoli_mn2/usr/lib/X11/app-defaults/Tivoli	8192
/Tivoli_mn2/usr/local/Tivoli	204800
/Tivoli_mn2/var/spool/Tivoli	65536
/Tivoli_mn2/etc/Tivoli	8192

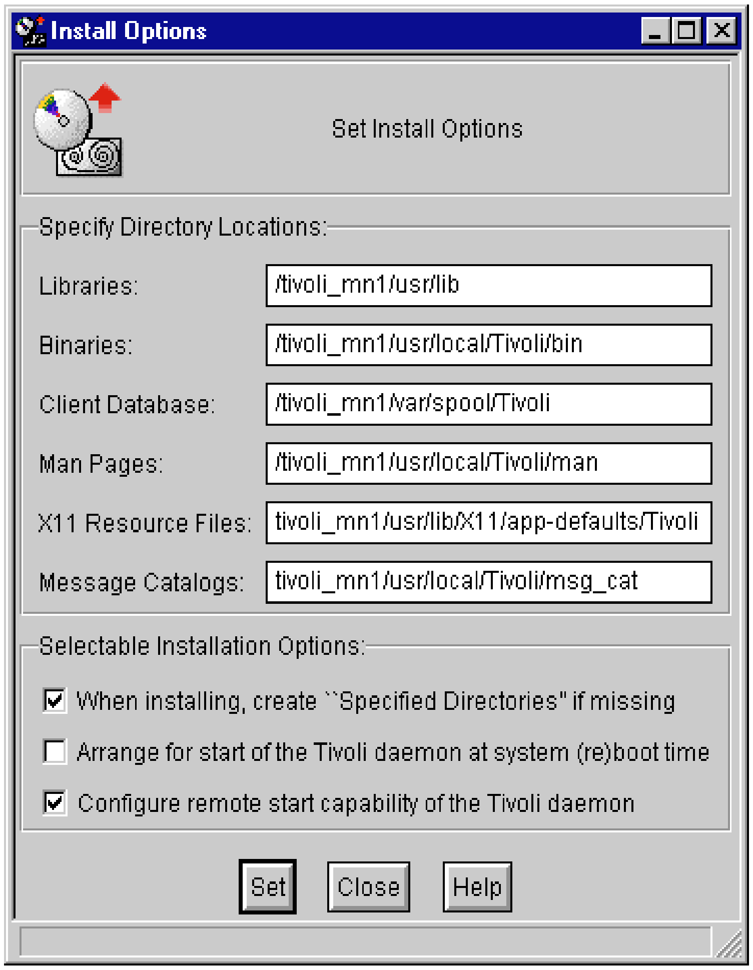
4.3.2 Installation Process

Just as before, to install the managed nodes in our cluster, we did the following:

1. We created the appropriate start scripts, which point to the appropriate setup_env and oserv.rc files. These were placed in the /usr/local directory in rootvg on both systems.
2. We created two resource groups in HACMP each containing the four appropriate file systems and service IP addresses. These were called blackres and brownres. We configured the node relationship such that the resource group blackres normally runs on the system black but would failover to the system brown and the reverse for the other resource group, brownres.
3. We synchronized the changes to all the nodes and started HACMP on both nodes.

   HACMP mounted all the necessary file systems and made the service IP addresses available.
4. We installed the managed node black specifying the appropriate paths so that all files are placed into the shared disk. The install options can be seen in the following figure:

:



Specify Directory Locations:

Libraries:	/tivoli_mn1/usr/lib
Binaries:	/tivoli_mn1/usr/local/Tivoli/bin
Client Database:	/tivoli_mn1/var/spool/Tivoli
Man Pages:	/tivoli_mn1/usr/local/Tivoli/man
X11 Resource Files:	tivoli_mn1/usr/lib/X11/app-defaults/Tivoli
Message Catalogs:	tivoli_mn1/usr/local/Tivoli/msg_cat

Selectable Installation Options:

- [✓] When installing, create ``Specified Directories'' if missing
- [ ] Arrange for start of the Tivoli daemon at system (re)boot time
- [✓] Configure remote start capability of the Tivoli daemon

Figure 7. Installation Options for Managed Node Black

5. We repeated the process and installed the managed node brown again giving the appropriate path values such that all files are placed into the shared volume group, brownvg. It is important *not to check* the box called Arrange for start of the Tivoli daemon at system (re)boot time.
6. Once this had completed, we copied the files that had been placed in /etc/Tivoli on each system to the appropriate location on a shared disk with the following commands

On black: `cp - p /etc/Tivoli/* /Tivoli_mn1/etc/Tivoli/`

On brown: `cp - p /etc/Tivoli/* /Tivoli_mn2/etc/Tivoli/`

7. We set the appropriate value for the WLOCALHOST variable in the setup_env.sh and oserv.rc files for each managed node.

4.3.3 The set_force_bind Variable

In previous versions of Tivoli, the oserv process would bind to its given port number but would listen on all IP addresses. This could be seen by running `netstat -n | grep 94` where 94 is the port number in use. This would return the following:

```
tcp4       0      0 *.94         *.*                   LISTEN
udp4       0      0  *.94                  *.*
```

Should a second oserv process attempt to start using the same port number, then it would return the following error before timing out and failing:

```
1999/02/24 11:27:04 +06: Tivoli Framework (tmpbuild) #1 Tue Jul 21 11:04:01
CDT 1998
Copyright Tivoli Systems, 1997.  All Rights Reserved.
TMR 1669180253. ORB 2. TMR server brown.itsc.austin.ibm.com:94. Port 94.
pid 10312
1999/02/24 11:27:46 +06: ^port 94: In use.  Waiting...
1999/02/24 11:28:06 +06: ^port 94: In use.  Waiting...
1999/02/24 11:28:26 +06: ^port 94: In use.  Waiting...
```

or the following error:

```
1999/03/02 15:26:27 +06:
!/Tivoli_tec/usr/local/Tivoli/bin/aix4-r1/bin/oserv: odlist init failed.
requested resource not found. (30)
```

It appears that the first time the condition is encountered, you receive the first error. On subsequent attempts to start the second oserv process, you receive the second error. In any event, you will not be able to start two oserv processes that are bound to the same port if they do not bind to a specific IP address.

With Tivoli Version 3.6, there has been a new variable added that forces the oserv process to bind to the IP address that has been assigned to it. This variable is the set_force_bind variable. The default value for this variable is set to FALSE. It is, therefore, necessary to change the value of this variable to TRUE using the following command on all systems requiring the change:

```
# odadmin set_force_bind TRUE {od number}
```

The `od number` can be obtained by running the following on any system in the TMR region:

```
# odadmin odlist
```

```
Region      Disp  Flags  Port           IPaddr   Hostname(s)
1669180253     1    ct-    94       9.3.187.134
brown.itsc.austin.ibm.com,brown
               2    ct-    94       9.3.187.136   black.itsc.austin.ibm.com
               3    ct-    94       9.3.187.138   pooh.itsc.austin.ibm.com
```

where the value in the `Disp` column is the `od number` that is required.

Once the value has been set for the appropriate instances of the oserv process, then they will need to be stopped and restarted in order for the change to take effect.

It is also worth noting that it is possible to set the value of set_force_bind for all instances. This does not, however, change the default state of the variable for new installs. All new installs will still have the value set to FALSE. You can check the state of the variable by running the following:

```
# odadmin
Region = 1669180253
Dispatcher = 1
Interpreter type = aix4-r1
Database directory = /Tivoli_tmr/var/spool/Tivoli/brown.db
Install directory = /Tivoli_tmr/usr/local/Tivoli/bin
Inter-dispatcher encryption level = simple
Kerberos in use = FALSE
Remote client login allowed = TRUE
Install library path = /Tivoli_tmr/usr/local/Tivoli/lib/aix4-r1:/usr/lib:
Force socket bind to a single address = TRUE
Perform local hostname lookup for IOM connections = FALSE
Tivoli Framework (tmpbuild) #1 Tue Jul 21 11:04:01 CDT 1998
Copyright Tivoli Systems, 1997.  All Rights Reserved.

Port range = (not restricted)
State flags in use = TRUE
State checking in use = TRUE
State checking every 180 seconds
Dynamic IP addressing allowed = FALSE
```

Once the oserv process has been restarted with the new value of TRUE, then the output of the command `netstat -n` looks like the following:

```
# netstat -n | grep 94
tcp4       0      0  9.3.187.136.34150      9.3.187.138.94
ESTABLISHED
tcp4       0      0  9.3.187.134.94         9.3.187.136.34147
ESTABLISHED
```

```
tcp4       0      0  9.3.187.136.34147      9.3.187.134.94
ESTABLISHED
tcp4       0      0  9.3.187.136.94         *.*                    LISTEN
tcp4       0      0  9.3.187.134.94         9.3.187.138.3239
ESTABLISHED
tcp4       0      0  9.3.187.134.94         *.*                    LISTEN
```

which shows two oserv processes bound to different IP addresses using the same port number of 94.

4.3.4 Application Server

In order for HACMP to control the starting and stopping of the managed node, we need to configure our start and stop scripts into an application server within HACMP. This application server should then be added to the resource group. To make these two changes take effect, we will need to stop HACMP on both nodes before we synchronize. At this time, HACMP will not issue any commands to stop the oserv process. Therefore, before stopping HACMP, you should issue the following to stop the oserv process:

```
# . /etc/Tivoli/setup_env.sh
# /etc/Tivoli/oserv.rc stop
```

Now that the oserv process has been stopped, HACMP will be able to stop cleanly. We, therefore, stopped HACMP on both nodes in the normal *graceful* manner. After the stop had completed, we synchronized and verified the resources using the following SMIT fastpath:

```
# smitty clsyncnode.dialog
```

Once the synchronization process had completed successfully, we then were able to restart HACMP on both systems. The oserv process started on black. We were able to see this by running the following commands:

```
# ps -ef | grep oserv
    root 14268     1   0   Feb 26      - 29:29
/Tivoli_tmr/usr/local/Tivoli/bin/aix4-r1/bin/oserv -p 94 -k
/Tivoli_tmr/var/spool/Tivoli/black.db
root 23848 19948   3 12:19:32  pts/3  0:00 grep oserv
# . /etc/Tivoli/setup_env.sh
# odadmin odlist
Region      Disp  Flags  Port            IPaddr   Hostname(s)
1669180253     1    ct-    94       9.3.187.136
black.itsc.austin.ibm.com,brown
               2    ct-    94       9.3.187.138
pooh.itsc.austin.ibm.com
```

Once these steps had been completed, we were able to failover from ’black’ and the oserv process started on ’brown’. After the failover had completed, the TMR server was able to communicate with the managed node known to it as ’black’.

4.4 TMR Servers in Either Idle Standby or Mutual Takeover

The TMR server is little more than a specialized version of the managed node. Both use the oserv process and use the same structure for file placement. As a result, all that has been said up until now for building managed nodes into HACMP Clusters can be directly applied to the TMR server.

Note

Maybe we should explain here why you might need more than one TMR server in a your Tivoli environment. The answer is in the fact that one TMR server can manage up to 200 managed nodes. Gateways should also be counted as managed nodes since a gateway is actually resides on a managed node. So, if you need more than 200 managed nodes or gateways in your environment, you should think of deploying more than one TMR server. Tivoli provides a for TMR interconnection that allows you to manage resources that belong to different TMRs from a single console.

4.4.1 File System and Disk Configuration

As can be seen from Figure 6 on page 30, we created a volume group for each system. For the idle standby cluster, only the single volume group was required. We built the following file systems in the volume group blackvg.

Table 4. File Systems for Idle Standby of a TMR Server

Mount Point	Size (in 512 byte blocks)
/Tivoli_tmr/usr/lib/X11/app-defaults/Tivoli	8192
/Tivoli_tmr/usr/local/Tivoli	204800
/Tivoli_tmr/var/spool/Tivoli	65536
/Tivoli_tmr/etc/Tivoli	8192

For the mutual takeover environment, both volume groups were used, and the following file systems were created:

Table 5. File Systems for First TMR Server Black

Mount Point	Size (in 512 byte blocks)
/Tivoli_tmr1/usr/lib/X11/app-defaults/Tivoli	8192
/Tivoli_tmr1/usr/local/Tivoli	204800
/Tivoli_tmr1/var/spool/Tivoli	65536
/Tivoli_tmr1/etc/Tivoli	8192

Table 6. File Systems for Second TMR Server Brown

Mount Point	Size (in 512 byte blocks)
/Tivoli_tmr2/usr/lib/X11/app-defaults/Tivoli	8192
/Tivoli_tmr2/usr/local/Tivoli	204800
/Tivoli_tmr2/var/spool/Tivoli	65536
/Tivoli_tmr2/etc/Tivoli	8192

4.4.2 Variable Considerations with the TMR Server

It is worth noting that you can set the WLOCALHOST variable before running the install process for the TMR server with the following command:

```
# WLOCALHOST=black.itsc.austin.ibm.com
# export WLOCALHOST
```

The TMR server install process will proceed to pick up this variable and add it to the setup_env and oserv.rc files in /etc/Tivoli. If you do not set the WLOCALHOST variable before running the install process, then you can manually alter the files to include the correct value.

Whichever method you choose for setting the value of WLOCALHOST, it is necessary to set this parameter to avoid the oserv process from binding to the system hostname. Failover will not be possible if the system hostname is used instead of WLOCALHOST because the standby system will have a different hostname.

One other point of note is that there is a documented variable called EtcTivoli, which makes it possible to have the install process for the TMR server to write the files that are normally placed into /etc/Tivoli to a different location. However, despite the fact that moving /etc/Tivoli would be useful, we would advise that you do not use this variable and leave the location of these file to the default location. This is because the use of the EtcTivoli variable does not alter the paths by which other Tivoli products, such as TEC, will attempt to access the files in /etc/Tivoli. This could cause the operation of these products to fail.

4.4.3 TMR Server Summary

Once the relevant file systems have been created as described and the WLOCALHOST variable set, the TMR server can be installed from CDROM onto the system or systems. In all other aspects, the processes for integrating the TMR server into an HACMP cluster is the same as that described for the managed node in Chapter 4.2, “Idle Standby of a Managed Node” on page 30 or 4.3, “Mutual Takeover of Managed Nodes” on page 34.

4.5 Tivoli Enterprise Console in an Idle Standby HACMP Cluster

We have taken the view that the failover of the relational database on which TEC depends has already been covered in other specialized redbooks. Therefore, we have configured our cluster such that the Oracle database that we used is running on a stand-alone server outside of our cluster. The RIM host that enables TEC to interact with the database is also resident on the same stand-alone system that contains the database. We feel that splitting up the location of the TEC server and its database has the dual effect of making the HACMP implementation simpler and also results in a faster failover.

4.5.1 File System and Disk Configuration

Once the complications of the database and the RIM host have been removed, TEC is again very similar to the managed node. Additional software is installed on the shared disk, but the same file system structure is used as for the managed node, as shown below:

Table 7. File Systems for Idle Standby of a TEC Server

Mount Point	Size (in 512 byte blocks)
/Tivoli_tec/usr/lib/X11/app-defaults/Tivoli	8192
/Tivoli_tec/usr/local/Tivoli	204800
/Tivoli_tec/var/spool/Tivoli	65536
/Tivoli_tec/etc/Tivoli	8192

4.5.2 Installation Process

In the following sections, we describe the steps we used for installing TEC into the HACMP cluster.

4.5.2.1 Database Installation and Configuration

In our test environment, we happened to use the Oracle database product on NT in conjunction with TEC. In this section, we briefly describe the steps we used to start and stop the Oracle process. If you will be using another database product, you will need to provide similar scripts.

Installing Oracle Database

We installed Oracle 7.3 on an NT Managed node called pooh. The following services were installed from the Oracle CDROM \NT_X86\INSTALL directory to the C:\ORANT directory:

- Oracle Documentation

- Oracle SQL Plus Product

- Oracle7 Server

- Oracle7 Client

Since we created the RIM Object on the database server we didn’t need to install Oracle7 Client on another managed node. If you create the RIM Object in a node other than server, you should also install and configure the database client on that host.

Important

The RIM support for Oracle for Windows NT is linked against orant71.dll,which is the DLL supplied with Oracle 7.1. The name of this library was changed in Oracle 7.2 and Oracle 7.3, and oracle71.dll doesn’t get installed with Oracle 7.2 or Oracle 7.3. However, orant71.dll is available on the Oracle 7.2 and Oracle 7.3 installation media.

From the Oracle 7.3 media, you can copy the ORANT71.DLL, CORENT23.DLL and MSVCRT10.DLL from the NT_X86\V7\RSF2 directory to the $ORACLE_HOME\bin directory.

Starting and Stopping the Oracle Database

All of the commands shown below must be executed by the Oracle user. There are two portions of the Oracle environment we must deal with. One is the listener, which enables you to connect to the database from a client program. The other part is the database itself. The programs depend on the ORACLE_HOME and the ORACLE_SID environment variables. Therefore, you must ensure they are set properly before any of these commands are executed. In our scenario they were `C:\ORANT` and `ORCL`, respectively.

For the listener component:

Start Script:
```
lsnrctl start <listenername>
```

Stop Script:
```
lsntctl stop <listenername>
```

For the database component:

Start Script:
```
dbstart
```

Stop Script:
```
dbshut
```

4.5.2.2 TEC Installation and Configuration

In our environment, we installed the TEC server and one TEC console on black. Also, we installed one logfile adapter on black and one NT event adapter on pooh. Our configuration is shown in the following figure:

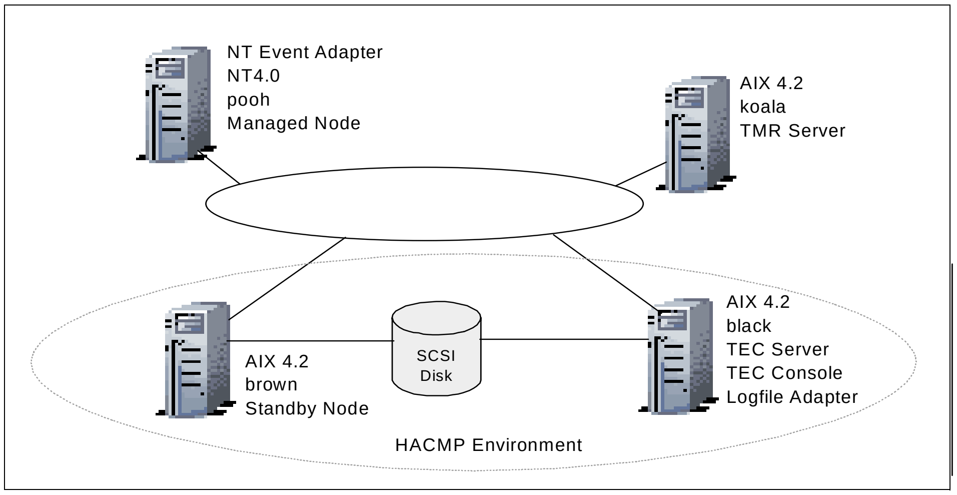
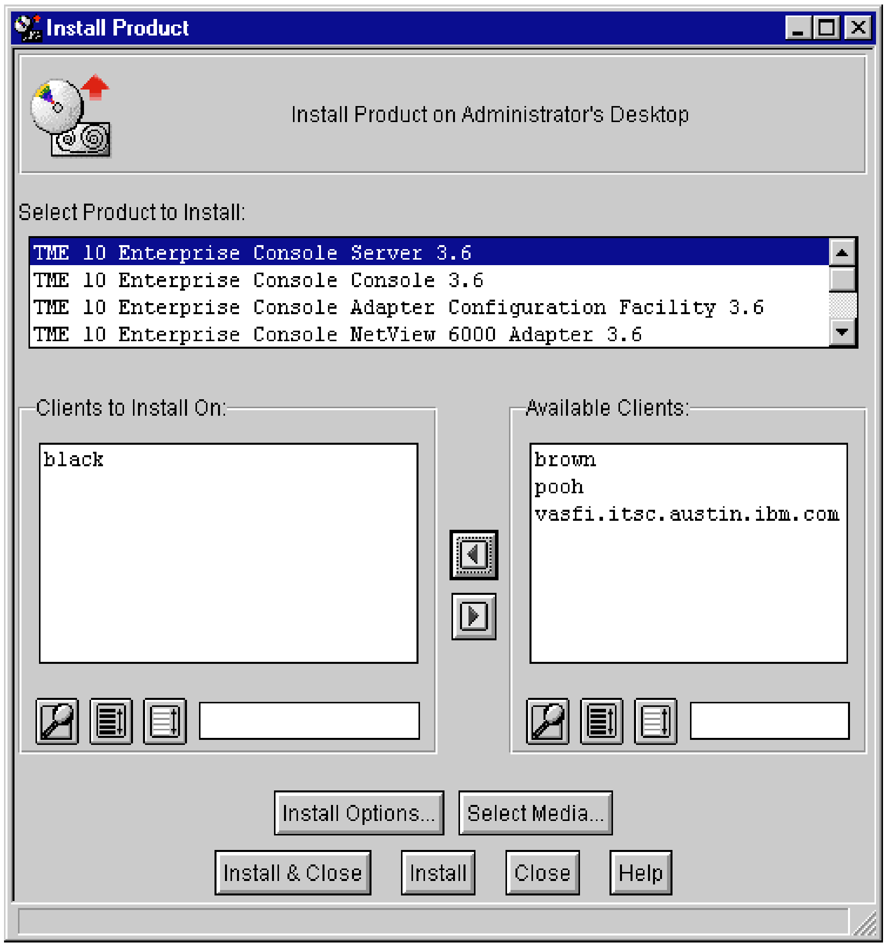


Figure 8. Environment for TEC Hot Standby Scenario

We should perform the following steps:

1. Install the TEC server component from the desktop to managed node black as follows:



```
TME 10 Enterprise Console Server 3.6
TME 10 Enterprise Console Console 3.6
TME 10 Enterprise Console Adapter Configuration Facility 3.6
TME 10 Enterprise Console NetView 6000 Adapter 3.6
```

Clients to Install On:

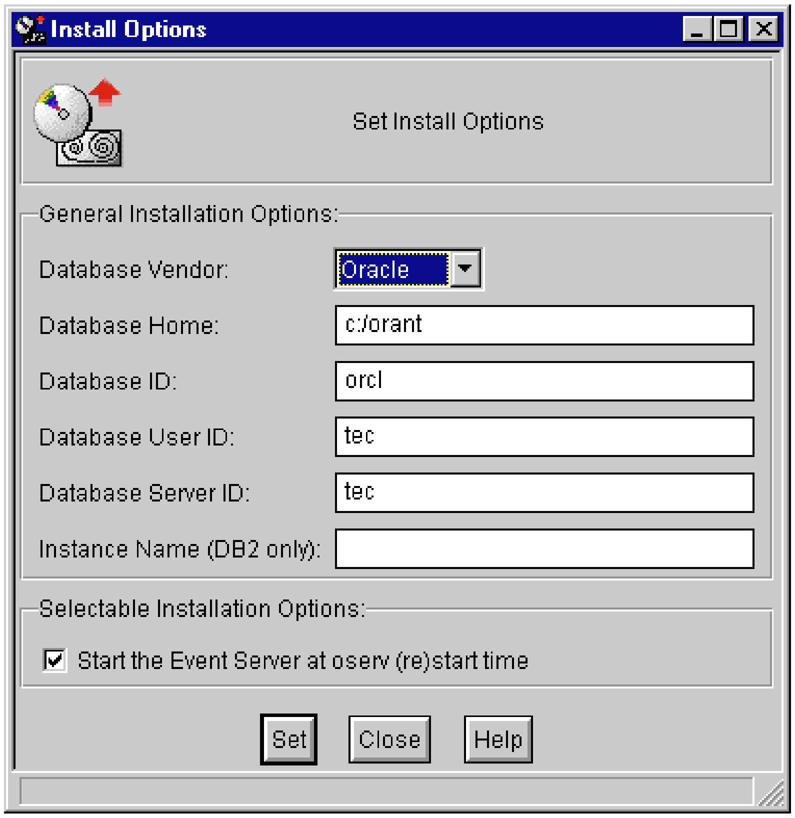
```
black
```

Available Clients:

```
brown
pooh
vasfi.itsc.austin.ibm.com
```

Figure 9. TEC Installation Managed Node Black

The install option screen has the following parameters:


Install Options

Set Install Options

General Installation Options:

Database Vendor:	Oracle
Database Home:	c:/orant
Database ID:	orcl
Database User ID:	tec
Database Server ID:	tec
Instance Name (DB2 only):	

Selectable Installation Options:

[✓] Start the Event Server at oserv (re)start time

Figure 10. Install Options Parameters

2.Create the TEC Tables for Oracle.

Since the scripts necessary to create the TEC database schema are residing on the TEC server black, we use ftp to copy all files under the $BINDIR/Tivoli/tec/sql directory of the TEC server black to a directory (C:\TMP in our case) on the NT managed node pooh, which is used as the database server and RIM host. Under the bash environment on the NT managed node pooh, run the following script:

```
$bash>cr_tec_db.sh
```

We use the Oracle default Admin user `sys` and default password `change_on_install` when we are prompted. At that point, the tables should be created successfully.

3.Install the TEC console to black from the desktop.

4. Create the TEC console on black with the following command:

```
wcrtconsole -h @ManagedNode:black @Root_brown-region
```

5.Add all roles for EventServer and TEC36Region resources to the root administrator in order for the Administrator to run and customize the TEC application.

6. Delete the default RIM object and re-create the RIM on the pooh managed node.

```
wdel @RIM:tec
wcrtrim -v Oracle -h pooh -d orcl -u tec -H c:\orant -s tec tec
```

Instead, you can also use the `wsetrim` command to change the default RIM parameters. The reason for this step is that when you install the TEC server, Tivoli automatically creates a RIM object *on the same server* that TEC is installed on using the parameters given in the TEC install Options panel. If you intend to create a RIM object on a node other than the TEC server (which is the recommended method by Tivoli for performance reasons), you should change the default RIM object created.

7.Set the password for `tec` user to `tectec`:

```
wsetrimpw tec tectec
```

This is the default password that TEC uses.

8. View the RIM configuration from the output of the `wgetrim tec` command as follows to ensure the parameters are set correctly.

```
wgetrim tec
RIM Host:       pooh
RDBMS User:     tec
RDBMS Vendor:   Oracle
Database ID:    orcl
Database Home:  c:orant
Server ID:      tec
Instance Home:
```

9. Start the TEC server by issuing the command:

```
wstartesrv
```

10.Verify the TEC server installation.

We use the command `wpostemsg` to send a dummy event to the console to see whether it arrives successfully. For example:

```
wpostemsg -m "Test_Event" TEC_Notice black
```

We also checked the five TEC server processes with the `ps -ef | grep tec_` command, the result of which is shown below:

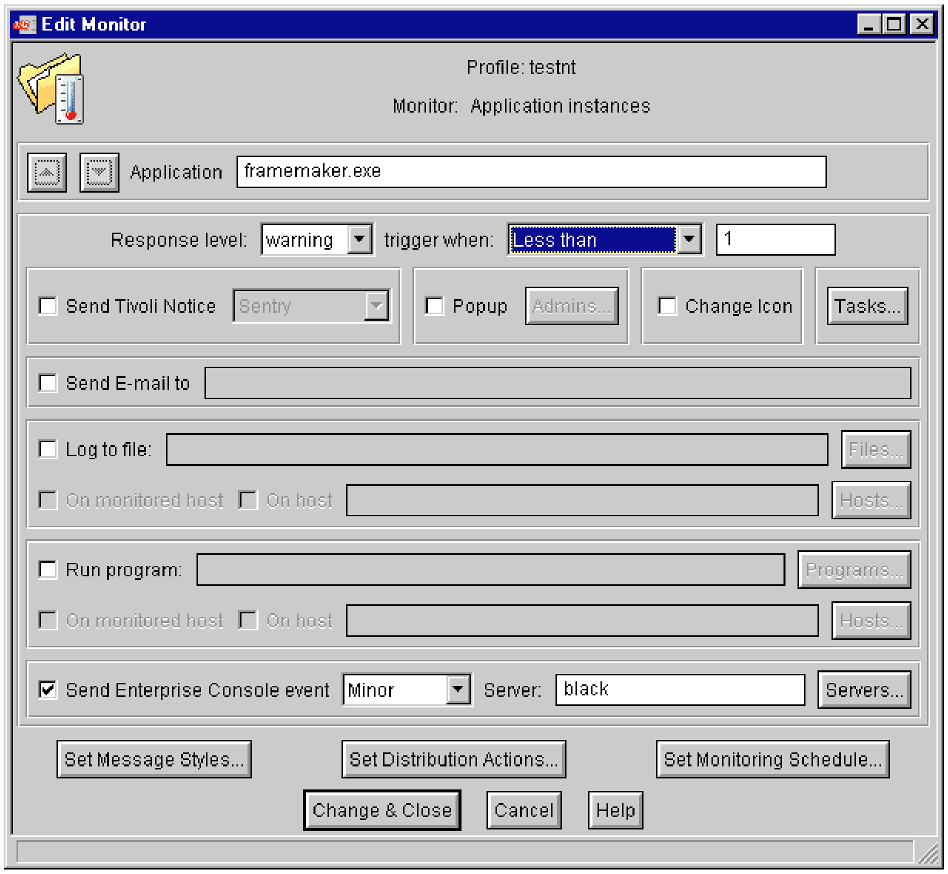
```
ps -ef | grep tec_

root  5210 23368   0 11:31:46 0:01 tec_server
root  25288 5210   0 11:58:48 -  0:02 tec_task -config
/usr/local/Tivoli/bin/aix4-r1/Tivoli/TEC/.tec_config
root  6449 19066   3 15:10:50 pts/0  0:00 grep tec_
root  4998 5210    0 11:31:48 -  0:06 tec_reception -config
/usr/local/Tivoli/bin/aix4-r1/Tivoli/TEC/.tec_config
root 23748 5210    0 11:31:48 -  0:05 tec_rule -config
/usr/local/Tivoli/bin/aix4-r1/Tivoli/TEC/.tec_config
root 11172 5210    0 11:31:48 -  0:11 tec_dispatch -config
/usr/local/Tivoli/bin/aix4-r1/Tivoli/TEC/.tec_config
```

Important

If you issue the `ps -ef | grep tec_` command and see only the tec_server process running. This *does not* mean that TEC is running successfully. All five processes should be running successfully before concluding an error-free TEC server installation.

11. For further testing, we create a simple distributed monitor profile as follows:



Profile: testnt

Monitor: Application instances

Application: framemaker.exe

Response level: warning

trigger when: Less than 1

☐ Send Tivoli Notice: Sentry

☐ Popup

☐ Change Icon

☐ Send E-mail to

☐ Log to file:

☐ On monitored host ☐ On host

☐ Run program:

☐ On monitored host ☐ On host

☑ Send Enterprise Console event: Minor

Server: black

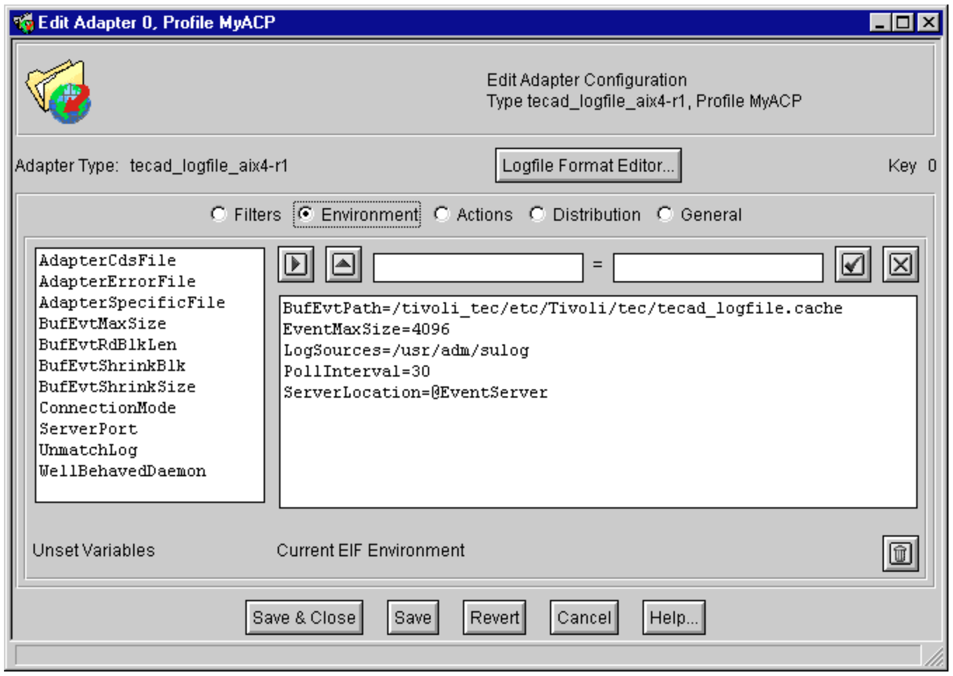
Figure 11. DM Profile for Monitoring Framemaker.exe Program

This simple monitor checks the occurrence of the framemaker.exe program on pooh, and if it doesn't find any instance, sends a minor event to the TEC server. After distributing the profile to pooh, we killed the framemaker.exe program from the Windows Task Manager to see that we receive the TEC event successfully.

12.In order to test the logfile adapter, we install the Adapter Configuration Facility to black, pooh, and the TMR server koala from the desktop.

13. Install a logfile adapter to black, creating an ACP profile called MyACP for the log adapter and distributing the profile to black. There are two important parameters in the profile related to our environment:

BufEvtPath: This parameter specifies the path that the log adapter caches its events in case communication between the TEC server and the adapter is broken. The default path is /etc/Tivoli/tec. Since we want it to reside on the shared disk, we changed the default path as follows:



Selected tab: Filters [✓ Environment] Actions Distribution General

Unset Variables:

```
AdapterCdsFile
AdapterErrorFile
AdapterSpecificFile
BufEvtMaxSize
BufEvtRdBlkLen
BufEvtShrinkBlk
BufEvtShrinkSize
ConnectionMode
ServerPort
UnmatchLog
WellBehavedDaemon
```

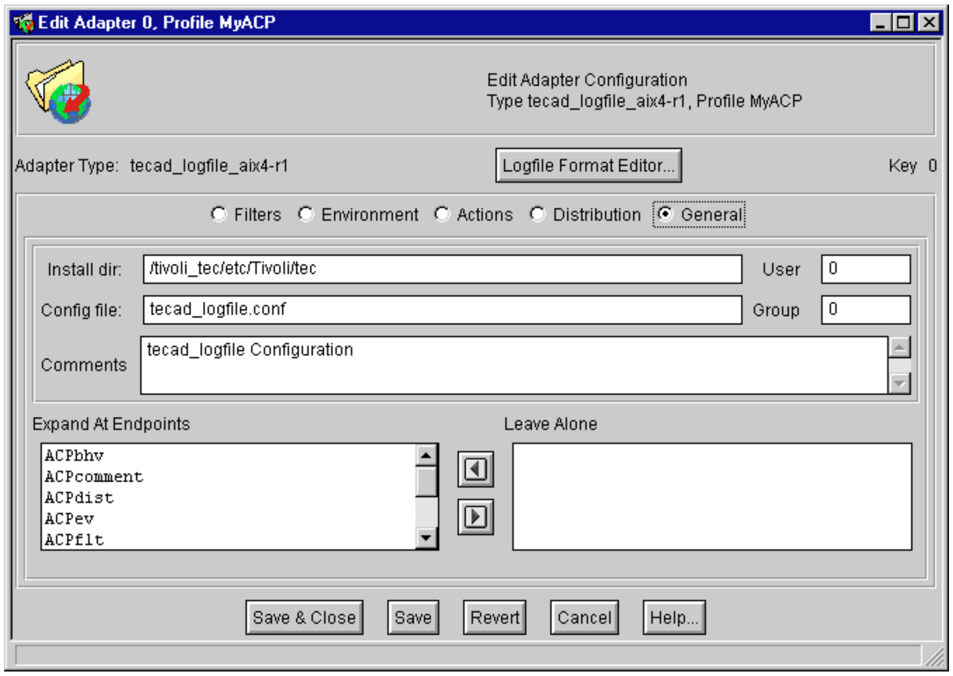
Current EIF Environment:

```
BufEvtPath=/tivoli_tec/etc/Tivoli/tec/tecad_logfile.cache
EventMaxSize=4096
LogSources=/usr/adm/sulog
PollInterval=30
ServerLocation=@EventServer
```

Figure 12. Changing BufEvtPath Variable

If the adapter is already installed, it is also possible to change this entry from the adapter configuration file tecad_logfile.conf directly.

Install dir: This is the installation directory of the logfile adapter configuration file. The default directory is /etc/Tivoli/tecad/etc. Since we want to put it on the shared disk, we change the default value as follows:


Edit Adapter 0, Profile MyACP

Edit Adapter Configuration
Type tecad_logfile_aix4-r1, Profile MyACP

Adapter Type: tecad_logfile_aix4-r1

Key 0

Filters [✓ General] Environment Actions Distribution

Install dir: /tivoli_tec/etc/Tivoli/tec

User 0

Config file: tecad_logfile.conf

Group 0

Comments: tecad_logfile Configuration

Expand At Endpoints:

ACPbhv
ACPcomment
ACPdist
ACPev
ACPflt

Leave Alone:

Figure 13. Changing Installdir Variable

14. Install an NTEvent adapter to pooh using ACP again, but this time with the default parameters.

15. Verify that both of our adapters are running correctly and sending events to the TEC server.

16. Now we are ready to fail over. After the failover, oserv automatically starts the tec-master daemon since this option was set when we installed the TEC server on Figure 8 page 18. The Tec-master daemon starts the rest of the four TEC processes. We repeat steps 10, 11, and 15 to verify the failover.

In our scenario, the TEC binaries were installed in the /Tivoli_tec/usr/local/Tivoli directory on the shared disk; so, we didn't have to synchronize TEC binaries across the cluster. The default rule base resides in $BINDIR/TME/TEC/default-rb. But, this is also part of the binary tree of the Tivoli environment that we installed on the shared disk. So, we didn't have to synchronize this directory either.

4.5.3 Some Diagnostic Utilities

There are several diagnostic utilities that can help you in case of a problem. Most of the TEC problems occur due to database problems or invalid RIM objects. To check if you have a connection to the RDBMS, you can use the `wtdbstat` command. The output of this command should look like this:

```
wtdbstat
The RDBMS database server is running.
```

Also, you can test if you have a valid RIM connection or not with the `wrimtest` command. The command syntax is: `wrimtest [-l RIM-object-label]`

For example, if you run `wrimtest -l tec`, and if you receive a "`Opening Regular Session ... Session Opened`" message, you can be sure that your RIM connectivity has no problem.

There is also a `wrimtrace` command that you may find useful. The command syntax is:`wrimtrace RIM-object-label [INFORMATION|ERROR|TRACE_OFF]`

This command writes the contents of the IOM packets passed between the RIM object and client program and RDBMS errors to a file located in /tmp/rim_db_log by default

You can use the product documentation *TME 10 Framework Reference Manual* for more information about RIM related commands.

The status of the TEC server can be found by using the `wstatesrv` command. The command syntax is `wstatesrv [-S server]`.

If the TEC server is running, you can also check if there are error files in the /tmp directory. The names of these files are:

- /tmp/tec_master
- /tmp/tec_reception
- /tmp/tec_rule
- /tmp/tec_task
- /tmp/tec_dispatch

The product documentation, *TME 10 Enterprise Console Reference Manual*, can be useful for other debugging commands.

4.5.4 Modifications to the Control Scripts

When the oserv process of the managed node is started, it also starts the tec-master daemon that starts the rest of the four TEC processes. There is, therefore, no need to alter the start scripts for HACMP.

However, stopping the oserv process does not necessarily stop all the TEC processes as well. Therefore, it is recommended to stop the TEC server before stopping the oserv. This is done using the following command:

```
# $BINDIR/bin/wstopesvr
```

where `$BINDIR` is the variable set within the setup_env scripts that give the location of all Tivoli binaries.

Note

During our tests, we found that, in some occasions, a single execution of the `wstopesvr` command was not sufficient to stop all the TEC processes. In some cases, we ran the command three times before the TEC processes were stopped successfully. As a result, our modified stop script actually looked like the following:

```
!/bin/ksh
. /Tivoli_tec/etc/Tivoli/setup_env.sh
$BINDIR/bin/wstopesvr
$BINDIR/bin/wstopesvr
$BINDIR/bin/wstopesvr
/Tivoli_tec/etc/Tivoli/oserv.rc stop
```

4.5.5 TEC Server Summary

The processes for integrating the TEC server into an HACMP cluster is the same as that described for the managed node in “Idle Standby of a Managed Node” on page 30. Care should be given to the configuration of log files.

Normally, the database that we used on NT should also be considered as a single point of failure in this scenario, but we assumed that it was implemented as a highly available database resource. How to do this can be found in various database related sources including redbooks, such as *Bullet-Proofing Your Oracle Database with HACMP: A Guide to Implementing AIX Databases with HACMP,* SG24-4788.

Although we were able to build two TEC servers into a mutual takeover cluster using the same process as described for the managed nodes, we do not see this as a likely customer requirement. We have, therefore, only documented the idle standby cluster.

4.6 TMR Server and Managed Node Mutual Takeover

This section will cover mutual takeover of a TMR server and managed node. In our cluster, one system has a TMR server running, and on the other, a managed node is configured.

In the event of either failing, the other node will take over the function of the failed node. We configured the TMR server on node black and managed node on brown. The file system structure is as follows.

Table 8. File System Structure for TMR Server on Black

Mount Point	Size (in 512 byte blocks)
/Tivoli_tmr/usr/lib/X11/app-defaults/Tivoli	8192
/Tivoli_tmr/usr/local/Tivoli	204800
/Tivoli_tmr/var/spool/Tivoli	65536
/Tivoli_tmr/etc/Tivoli	8192

Table 9. File System Structure for the Managed Node on Brown

Mount Point	Size (in 512 byte blocks)
/Tivoli_mn/usr/lib/X11/app-defaults/Tivoli	8192
/Tivoli_mn/usr/local/Tivoli	204800
/Tivoli_mn/var/spool/Tivoli	65536
/Tivoli_mn/etc/Tivoli	8192

4.6.1 Installation Process

The installation process is similar to installing the TMR server and managed node in a hot-standby scenario on individual systems.

As discussed earlier, we need to set the WLOCALHOST variable to prevent the oserv from binding to the host name. This is required to allow multiple oservs to run on a single system during failure.This was done by running the following command:

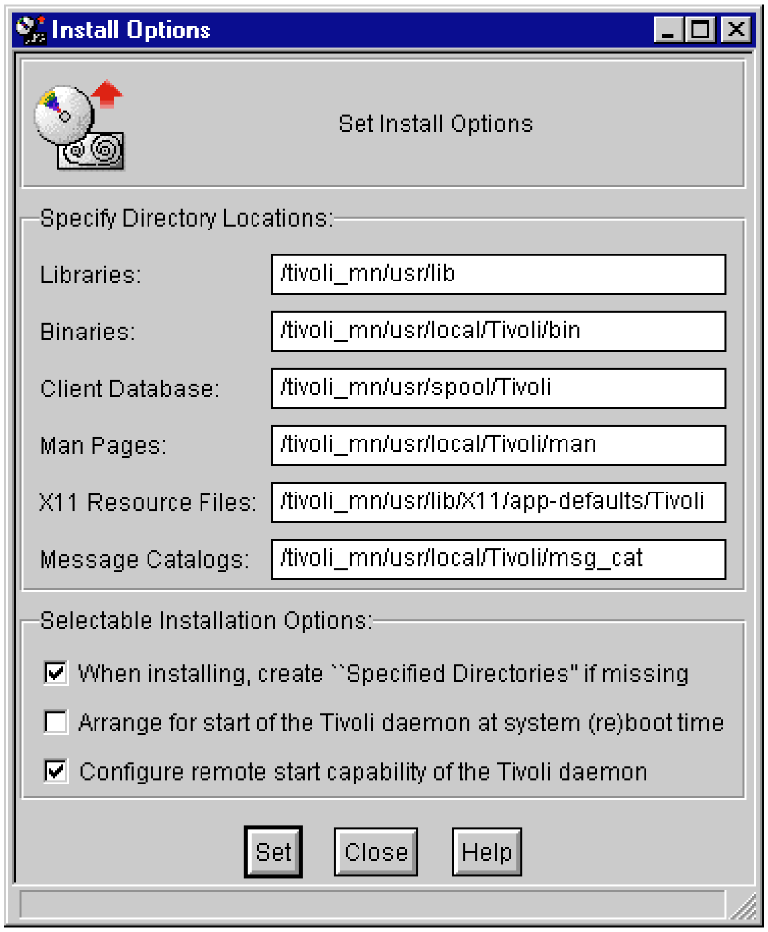
```
# WLOCALHOST=black.itsc.austin.ibm.com
# export WLOCALHOST
```

Now, the TMR server was installed on the service address of the system black with the file systems shown as above.

After the successful installation, we confirmed the TMR server was running by the following command:

```
# . /etc/Tivoli/setup_env.sh
# odadmin odlist
Region      Disp  Flags  Port            IPaddr   Hostname(s)
1669180253     1    ct-    94       9.3.187.136
black.itsc.austin.ibm.com,brown
```

Now, we are ready to install the managed node on system brown. As described earlier, we set the WLOCALHOST variable before the installation. We installed managed node from the newly created TMR server using the GUI interface. We installed to the service IP address of the system brown as seen in the following figure.



Specify Directory Locations:

Libraries:	/tivoli_mn/usr/lib
Binaries:	/tivoli_mn/usr/local/Tivoli/bin
Client Database:	/tivoli_mn/usr/spool/Tivoli
Man Pages:	/tivoli_mn/usr/local/Tivoli/man
X11 Resource Files:	/tivoli_mn/usr/lib/X11/app-defaults/Tivoli
Message Catalogs:	/tivoli_mn/usr/local/Tivoli/msg_cat

Selectable Installation Options:

- ✓ When installing, create ``Specified Directories" if missing
- ☐ Arrange for start of the Tivoli daemon at system (re)boot time
- ✓ Configure remote start capability of the Tivoli daemon

Figure 14. Installation Options for Managed Node Brown

Now that the TMR server and managed node are up, we run the `odadmin odlist` command with the following output:

```
# odadmin odlist
Region      Disp  Flags  Port           IPaddr   Hostname(s)
1669180253     1    ct-    94      9.3.187.134
brown.itsc.austin.ibm.com,brown
               2    ct-    94      9.3.187.136
black.itsc.austin.ibm.com
```

Since in a failed condition there will be two oservs running on one system, it is necessary to set the set_force_bind variable. This is done by running the following command:

```
# odadmin set_force_bind TRUE {od number}
```

We need to stop and start the oserv so that the changes are effective. After the oserv is restarted, the output of the odadmin command is:

```
# odadmin
Region = 1669180253
Dispatcher = 1
Interpreter type = aix4-r1
Database directory = /Tivoli_tmr/var/spool/Tivoli/brown.db
Install directory = /Tivoli_tmr/usr/local/Tivoli/bin
Inter-dispatcher encryption level = simple
Kerberos in use = FALSE
Remote client login allowed = TRUE
Install library path = /Tivoli_tmr/usr/local/Tivoli/lib/aix4-r1:/usr/lib:
Force socket bind to a single address = TRUE
Perform local hostname lookup for IOM connections = FALSE
Tivoli Framework (tmpbuild) #1 Tue Jul 21 11:04:01 CDT 1998
Copyright Tivoli Systems, 1997.  All Rights Reserved.
Port range = (not restricted)
State flags in use = TRUE
State checking in use = TRUE
State checking every 180 seconds
Dynamic IP addressing allowed = FALSE
```

Now, if either of the systems fail, the second system takes over the function of the first one.

4.6.2 Summary

This mutual takeover is similar to earlier mutual takeovers with one exception. We need to keep in mind that, when starting both systems, the TMR server should be started first and then the managed node. In the case where the managed node is started first, it will fail to start and will need to be manually started once the TMR server is up and running.

4.7 TEC Server and TMR Server Mutual Takeover

In this case, we have a TEC server running on one system and a TMR server running on the other system.The following figure shows our configuration.

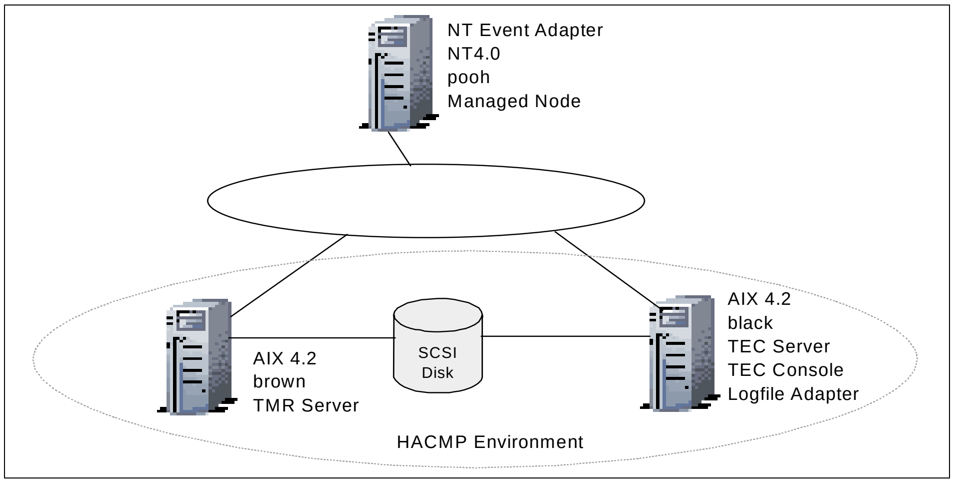


Figure 15. TEC and TMR Server Mutual Takeover Configuration

As in the earlier case, we will configure the database server outside of this cluster on a stand alone system.The RIM host that enables TEC to interact with the database is also resident on this same stand alone system that contains the database. This helps with both a faster implementation and faster failover.

4.7.1 File Systems

The TEC server scenario is similar to a managed node. As with the case of managed nodes, we have the following file systems.

Table 10. File System Structure for TEC Server on Black

Mount Point	Size (in 512 byte blocks)
/Tivoli_tec/usr/lib/X11/app-defaults/Tivoli	8192
/Tivoli_tec/usr/local/Tivoli	204800
/Tivoli_tec/var/spool/Tivoli	65536
/Tivoli_tec/etc/Tivoli	8192

Table 11. File System Structure for TMR Server on Brown

Mount Point	Size (in 512 byte blocks)
/Tivoli_tmr/usr/lib/X11/app-defaults/Tivoli	8192
/Tivoli_tmr/usr/local/Tivoli	204800

Mount Point	Size (in 512 byte blocks)
/Tivoli_tmr/var/spool/Tivoli	65536
/Tivoli_tmr/etc/Tivoli	8192

4.7.2 Installation Process

The installation process is described in the following sections.

4.7.2.1 TMR Server and Managed Node Installation

The installation procedure is similar to the earlier mutual takeover scenario. We first install the TMR server on the system brown. Once the TMR server is up and running, we install the manage node on the system black. After installing the managed node, TEC should be installed on black.

As with the earlier cases, we need to set the variable WLOCALHOST before the installation. Also, once installation is complete, and the oservs are running on both systems, we need to set the set_force_bind parameter to TRUE.

4.7.2.2 TEC Installation

TEC installation was exactly the same as in section 4.5.2.2, "TEC Installation and Configuration" on page 46. So, we won't repeat this procedure here.

The rest of the scenario is very similar to section 4.6, "TMR Server and Managed Node Mutual Takeover" on page 54. Once the TMR server and managed node are started on the same server, the tec-master process is started by the oserv process of the managed node, which, in turn, starts the rest of the four TEC processes.

In this scenario, we also tested the installation of logfile adapters on both black and brown using different Install dir and BufEvtPath directories as such: /tivoli_tmr/etc/Tivoli/tec for brown and /tivoli_tmr/etc/Tivoli/tec for black. We have been able to perform a successful mutual takeover, but we noticed that we received duplicate logfile events. This would be expected since although the configuration files of the two log file adapters were in different directories, they were both pointing to one single file: /usr/adm/su.log. So, if there is a need for using logfiles on both TMR server and TEC server in a mutual takeover scenario, a possible solution is to stop one logfile adapter just before the failover and restart it again on the failed system when the failed system comes back online.

4.7.3 Summary

As in the previous cases, care should be taken that the TMR server should be started first followed by the managed node on which the TEC processes depend.

Logfile adapters can add complexity to the environment but again we have the option of installing both the cache and configuration files on the shared disk.

4.8 Endpoints

An endpoint is a workstation running the Tivoli Management Agent. With the 3.6 version, Tivoli introduced a three-tiered architecture that included the TMR server, gateway, and endpoint where endpoints can dynamically subscribe to gateways that manage the endpoints on behalf of the TMR server. In this way, much of the burden on the TMR server is distributed to the gateways, and scalability of the overall management environment is greatly increased. Also, the cost of management decreases dramatically since endpoints require very small resources in terms of both memory and disk space, and once they are installed, they have the ability to update themselves automatically from the server or gateway in terms of new patches or new releases, which greatly reduces maintenance.

4.8.1 Endpoint Manager

An endpoint manager establishes and maintains the relationship between an endpoint and a gateway. An icon representing the endpoint manager is automatically created on the Tivoli desktop when you install a TMR server.

A primary role of the endpoint manager is to assign an endpoint to a gateway when the endpoint first logs in. If an endpoint’s assigned gateway stops responding to the endpoint, the endpoint manager is involved again to assign a new gateway to the endpoint.

4.8.2 Endpoint Gateways

Gateways assume some of the functions of a TMR server. Gateways perform all communications with the assigned endpoints without requiring additional information from the TMR server. A gateway must be created on a managed node.

4.8.3 File System and Disk Configuration

Two file systems were created on each system’s shared disk as shown in the following table.

Table 12. File Systems for Endpoint on System Brown

Mount Point	Size (in 512 byte blocks)
/Tivoli_ep/opt/Tivoli/lcf	16384
/Tivoli_ep/etc/Tivoli/lcf	8192

4.8.4 Endpoint Installation

Endpoints can not be installed from the Tivoli Desktop. Only gateways can be configured through the Tivoli Desktop.

There are two ways to install endpoints remotely:

1. Using Software Installation Services

2. From the command line using `winstlcf`

The syntax of `winstlcf` command is:

```
# winstlcf -g {gateway ip address} -d {destination directory} -i {end point
ip address}
```

For example, in our case, it would be:

```
# winstlcf -g 9.3.187.139 -d /Tivoli_ep/opt/Tivoli/lcf -i
brown.itsc.austin.ibm.com.
```

This will install the required files in the destination directory and /etc/Tivoli. As mentioned earlier, we need to copy the files from /etc/Tivoli to the shared disk.

4.8.5 HACMP Configuration

To be able to have the endpoint started and stopped by HACMP, we need to build the following start and stop scripts. These scripts will be configured into an application server.

The LCF Start script is:

```
#!/bin/ksh
. /Tivoli_ep/etc/Tivoli/lcf/1/lcf_env.sh
$LCF_DATDIR/lcfd.sh start
```

The LCF Stop script is:

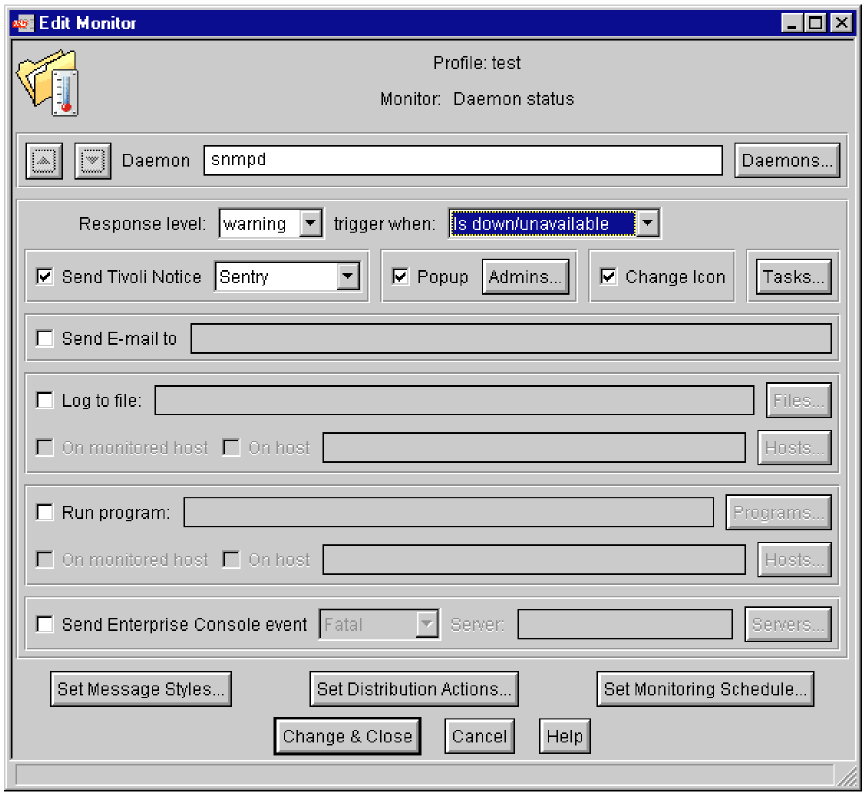
```
#!/bin/ksh
. /Tivoli_ep/etc/Tivoli/lcf/1/lcf_env.sh
$LCF_DATDIR/lcfd.sh stop
```

Once the application server has been configured, a resource group including the application server, the service IP address, and the two file systems, needs to be created. In order to synchronize this new information to all nodes in the cluster, HACMP should be stopped on all nodes. Once the synchronization process has completed successfully, then HACMP can be restarted, and this will bring online the necessary file systems and IP address followed by starting the endpoint service. This can be verified by running the following:

```
# ps -ef | grep lcf
root 17836     1   0 09:53:51      -  0:02 /Tivoli_ep/opt/Tivoli/lcf/bin/ai
x4-r1/mrt/lcfd
```

4.8.6 Distributed Monitoring Scenario on Endpoints

In this scenario, we tested a monitor to run on the endpoints and verified that the monitor continued to run without a problem. We created a monitor to test the status of the cron daemon on the brown and black endpoints as follows.


Edit Monitor

Profile: test

Monitor: Daemon status

Daemon: snmpd

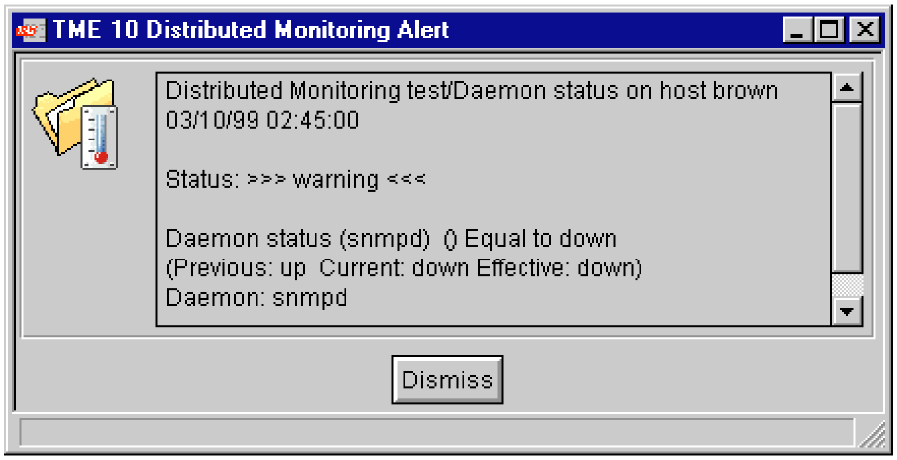
Response level: warning

trigger when: is down/unavailable

- [✓] Send Tivoli Notice: Sentry
- [✓] Popup
- [✓] Change Icon
- [ ] Send E-mail to
- [ ] Log to file:
- [ ] On monitored host
- [ ] On host
- [ ] Run program:
- [ ] On monitored host
- [ ] On host
- [ ] Send Enterprise Console event: Fatal

Figure 16. DM Profile to Monitor snmpd Daemon Up Status

After distributing the profile to brown and black, we killed the snmpd daemon on both systems and verified that we received two notifications one from brown and one from black. After the brown endpoint failed over onto black, we verified that we received the pop-up window from brown as well as black as follows:



Distributed Monitoring test/Daemon status on host brown
03/10/99 02:45:00

Status: >>> warning <<<

Daemon status (snmpd) () Equal to down
(Previous: up Current: down Effective: down)
Daemon: snmpd

Figure 17. Pop-up Notification after Failover

This is an expected result because of the following: The configuration file lcfd.dat file created by the lcfd daemon contains the two important pieces of information regarding endpoint login. One is the assigned gateway, and the other is the login interfaces information. The assigned gateway is the endpoint gateway that the endpoint has last logged into, and the login interfaces lists are the list of the alternate gateways where the endpoint will attempt to perform the endpoint login when the assigned gateway is not available. This file is carried to the failover node as part of the lcf installation tree; so, we can expect that the endpoint can successfully login to its last gateway. Once the login process completed, it can download the required monitor information from its gateway.

4.8.7 Endpoint Summary

In this section, we verified that the endpoint mutual, or idle takeover is not very different from the previous scenarios. We tested a distributed monitoring scenario to verify our failover was successful.

4.9 Summary

In this chapter, we described the HACMP specific considerations for implementing Tivoli TMR servers, managed nodes, endpoints, and TEC servers in HA environments. That, along with the information presented in Chapter 3, should provide you with the information you need to implement Tivoli in both hot standby and mutual takeover HA scenarios using HACMP.

Chapter 5. Solstice HA

Sun Microsystem's Solstice High Availability (Solstice HA) is the software that runs on Sun's Ultra Enterprise Cluster HA servers. The combination of hardware and software provides high availability support and automatic data services takeover. The specific configuration we tested consisted of two Sun servers running Solaris 2.5.1 (= SunOS 5.5.1), Solstice HA 1.3, and Solstice DiskSuite 4.1.

5.1 Solstice HA Overview

Solstice HA provides two possible configurations for the cluster: Symmetric and asymmetric. In a symmetric configuration, both machines are active and provide data services assuming a mutual takeover in the event of failure. In an asymmetric configuration, there is only one host running data services, and the second host is a hot standby.

The takeover in the Solstice HA is a rotating takeover—that is, the server that has given away its resources won't automatically take them back when it becomes active again in the cluster.

The core concept in Solstice HA is that of a data service, which is equivalent to a resource group in HACMP. The Solstice HA will associate within its data service the following elements:

- *A logical hostname* that is an IP label linked to an IP address and that will be used as a name of the application server (for example TMR server or managed node). Because this is not the hostname as reported by the `hostname` command, it can create problems with applications that are based on the actual hostname of the machine. In Solstice HA terminology, this last name is called *the physical hostname*.
- *Disksets* (similar to volume groups in AIX) are a group of disks that can move as a unit between HA servers. The Solstice HA cluster can have one or two disksets. Only one server can master a diskset at any point in time. That is, there is no concurrent access.
- The application-specific HA scripts (start, stop, abort, and so forth).

The Solstice configurations cover the following types of failures:

- Operating system failure (panic or crash)
- Data service application failure

• Server hardware failure
• Network interface failure
• Disk media failure

Important

The external disk failure is tolerated by using a mandatory mirroring option and possibly by a hot spare drive. However, in the case of a root disk failure, an automatic takeover might not occur at all. To avoid this situation, the mirroring of the internal root disk should also be implemented.

5.1.1 Solstice HA Data services

The Solstice HA product provides the procedures to install, configure, and administer some specific data services provided with the product. These services include the following:

• HA-NFS
• HA-DBMS for Oracle 7
• HA-DBMS for Sybase
• HA-DBMS for Informix
• HA Internet Pro

Any other product that will operate in the Solstice HA environment should be set up using the Solstice data services API that is used for registration, activation, providing the stop/start scripts, and so on. We include examples of Tivoli's module implementation in this chapter.

5.1.2 Solstice HA Restrictions

According to the Solstice HA documentation, the following is a list of current restrictions you may need to be aware of.

• HA cluster servers should not be configured as mail servers because *sendmail* is not supported in the Solstice HA environment. The mail directories should not reside on the servers either.
• There should be no applications that access the HA-NFS file system locally on the cluster.
• The servers cannot be used as routers.
• The current Solstice configuration is limited to two servers.

- The systems cannot provide the NIS or NIS+ services.
- The Solstice HA cluster does not support Secure NFS or the use of Kerberos with NFS.
- The network time protocol (NTP) is not supported on the cluster.
- Because Solstice HA cluster requires mirroring, the RAID 5 feature in the Solstice DiskSuite product is not supported.
- File system quotas are not supported.
- The HA administrative file system cannot be grown using the DiskSuite `growfs` command.
- All NFS client mounts must be *hard* mounts.
- No concurrent access of the disksets is allowed.
- For the two servers in a Solstice HA cluster, the hardware configurations must be identical regarding the following points:
  - Number and size of local disks
  - Number and types of connections to external disks
  - Number and types of network connections
  - Number and location of SBus cards

5.2 Our Lab Environment

The following figure represents the specific hardware configuration we used in our lab environment.

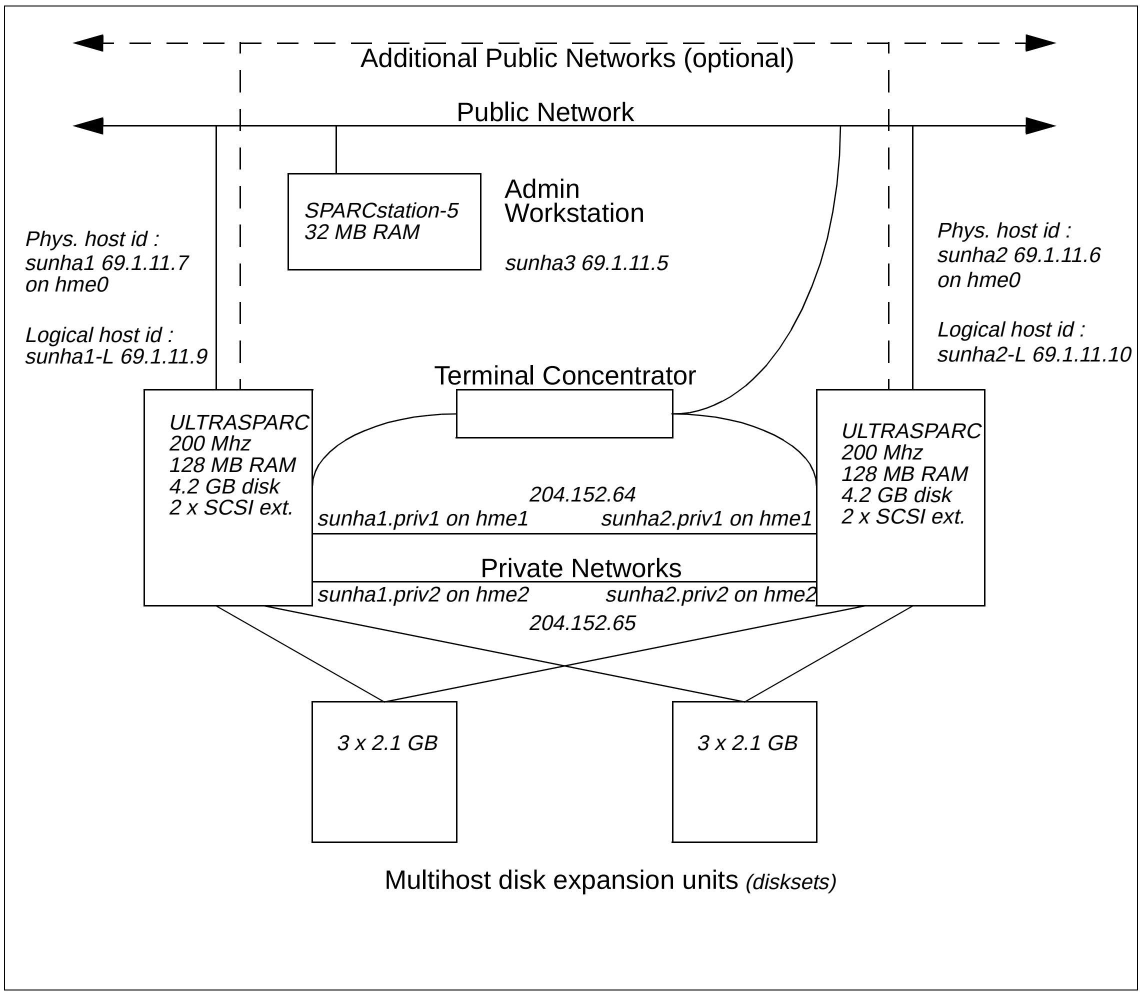


Figure 18. Ultra Enterprise Cluster HA Hardware Configuration

This configuration is the minimum required configuration and could be further expanded by adding more multi-host disk units and more public network interfaces.

The admin workstation (sunha3) is the only machine with a physical console; so, both servers will be accessed through this console as an X server.

The system software installed on the cluster is the following:

- Solstice HA 1.3
- Solstice DiskSuite 4.1

- Solaris 2.5.1

5.3 Failover Scenarios

In the following sections, we describe the specific scenarios we tested in the Solstice HA environment and provide specific information regarding their implementations.

5.3.1 TMR Server - Hot Standby Configuration

In this scenario, the machine sunha1 will be the TMR server, the host sunha2 will be a standby server, and we will use the admin workstation, sunha3, as both a managed node and as another TMR server interconnected with the primary TMR server.

The configuration for sunha1 is:

- Logical hostname (TMR server name) - sunha1-L.
- Physical hostname - sunha1.
- The data service will be called TMR_1.
- One diskset, sunha1-L (4 x 2.1 GB with mirroring), mounted under /sunha1-L/1:

```
File system               kbytes  used    avail capacity Mounted on
/dev/md/sunha1-L/dsk/d10 3818266 532203 2904243 16%     /sunha1-L/1
```

- Host table (`/etc/inet/hosts`):

```
127.0.0.1localhost
69.1.11.5sunha3
69.1.11.6sunha2
69.1.11.7sunha1 loghost
69.1.11.8annex
69.1.11.9sunha1-L
69.1.11.10sunha2-L
204.152.64.1sunha1-priv1
204.152.64.2sunha2-priv1
204.152.65.1sunha1-priv2
204.152.65.2sunha2-priv2
```

- Routing and network interfaces:

```
Destination      Gateway         Flags Ref  Use   Interface
--------------------------------------------------------
localhost         localhost        UH    0   43    lo0
```

```
69.1.11.0         sunha1          U     5   385   hme0
69.1.11.0         sunha1-L        U     5   0     hme0:1
204.152.65.0      sunha1-priv2    U     2   231   hme1
204.152.64.0      sunha1-priv1    U     2   231   hme2
224.0.0.0         sunha1          U     5   0     hme0
default           69.1.11.240     UG    0   31
```

- The installation option for the automatic starting of the Tivoli daemon should be disabled.
- The installation of the Tivoli framework has been done using the default paths for the binaries and database altered as shown below with /usr and /var replaced by a path pointing to the diskset owned by sunha1-L:

  ```
  /usr/local --> /sunha1-L/1/usr/local/Tivoli
  /var/spool --> /sunha1-L/1/var/spool/Tivoli
  ```

  During installation, we used the sunha1-L.region for the region name and sunha1-L for the TMR server name.
- The stop and start scripts for the TMR server have been placed in the /etc/Tivoli directory. This directory will be copied to sunha2 as shown in the next step. These scripts could reside in a different location but must be accessible by both servers during the takeover.

  Script stop_TMR_1:

  ```
  #! /bin/ksh
  #
  #stop local oserv daemon
  #
  /etc/Tivoli/oserv.rc stop
  #
  #  test the RC and print the message
  #
  if test $? -eq 0
    then
      echo "Tivoli daemon stopped"
  fi


  script start_TMR_1:

  #! /bin/ksh
  #
  # initialize the environment
  #
  . /etc/Tivoli/setup_env.sh
  #
  #  start oserv dispatcher
  ```

```
/etc/Tivoli/oserv.rc start
#
#  set the DISPLAY and start Tivoli GUI
export DISPLAY=sunha3:0
Tivoli
```

- The /etc/Tivoli directory on both servers in the cluster should now be synchronized (copied from sunha1 to sunha2). We have used the following commands to copy the files and to establish the links:

  On sunha1:

  ```
  tar -cvpf - /etc/Tivoli | rsh sunha2 tar -xvpf -
  ```

  On sunha2 from /etc/Tivoli:

  ```
  ln -s /sunha1-L/1/local/Tivoli/bin/solaris2/contrib/bin bin
  ln -s /sunha1-L/1/local/Tivoli/bin/solaris2/contrib/lib lib
  ```

5.3.1.1 Configuration of Solstice HA

1. The first step is to register the data service using the Solstice `hareg` command. The syntax for `hareg` is:

   ```
   hareg -r service_name -m START=start_script, STOP=stop_script
   ```

   In our specific environment, we issued the command:

   ```
   hareg -r TMR_1 -m \ START=/etc/Tivoli/start_TMR_1,STOP=/etc/Tivoli/stop_TMR_1
   ```

2. The next step is the activation of the data service, which results in the execution of the start script:

   ```
   hareg -y TMR_1
   ```

   To check whether a data service is active or not, Solstice provides the following command (returns 1 for active and 0 for inactive):

   ```
   haget -f service_is_on -s TMR_1
   ```

3. Another useful command to check the overall state of the cluster and data services is `hastat`, which should generate output similar to that shown below (truncated):

```
         Configuration State: Stable
        sunha2-L - Owned by sunha2
         sunha1-L - Owned by sunha1

 sunha2 - 10:10am up 5 day(s), 23:49, 1 user,load average:0.00,0.02,0.04
 sunha1 - 10:10am up 1 day(s), 17:52, 1 user,load average:0.00,0.01,0.02

        Local metadevices: sunha2 - (none); sunha1 - (none)
        Local metadb replicas: sunha2 - Ok; sunha1 - Ok
         Diskset sunha2-L
                metadevice status: Ok
                mediator status: Ok
                replica status: Ok
         Diskset sunha1-L
                metadevice status: Ok
                mediator status: Ok
                replica status: Ok
         Private nets: Ok
        Public nets: sunha2 - Ok; sunha1 - Ok
```

There are two disksets defined in this configuration: Sunha1-L and sunha2-L. However, only sunha1-L is used in this scenario.

• Takeover

At this point in time, we've got two servers, sunha1 and sunha2, owning their disksets, sunha1-L and sunha2-L. Sunha1 is running the data service TMR_1.

In the case of a failure of sunha1, its diskset, sunha1-L, logical hostname, sunha1-L, and IP address will be switched over by Solstice HA to the sunha2 host. The data service, TMR_1, will be automatically restarted on sunha2 using the appropriate start script.

We can simulate such a failure by the following Solstice HA commands:

```
hactl -g -s TMR_1 -l sunha1-L
```

It effectively performs the takeover, but at the same time forces the server sunha1 to reboot. The `haswitch` command is much more practical. The syntax is:

```
haswitch phys_destination_host logical_host
```

and in our specific environment:

```
haswitch sunha2 sunha1-L
```

During this phase of the takeover, both cluster servers produce the following sequence of events:

```
Oct 27 14:12:29 sunha2 hadf: NOTICE:fdl_checknameservice: fdl_checknameservice
succeeded
```

```
Oct 27 14:12:31 sunha2 hadf:NOTICE:switchoversub: last phase of haswitch command,
switchover file will contain: sunha1-L sunha2
Oct 27 14:12:33 sunha2 hadf:NOTICE:switchoversub: haswitch about to
reconfig:switchover file contains: sunha1-L sunha2
Oct 27 14:12:33 sunha2 hadf: NOTICE: starting "return" transition switchover
started - reconfiguration in progress...
Oct 27 14:12:33 sunha2 hadf: NOTICE: finished "return" transition
Oct 27 14:12:33 sunha2 ID[SUNWcluster.clustd.reconfig.1000]:hadf cluster
reconfiguring on node 0(sunha1)
Oct 27 14:12:34 sunha2 hadf: NOTICE: starting "step1" transition
Oct 27 14:12:35 sunha2 hadf:NOTICE: cltrans_hadf_init:HA_MEMBERSHIP is BOTH
Oct 27 14:12:40 sunha2 hadf: NOTICE: cltrans_hadf_init:
Computed HA_METASETSERVE = sunha1-L
Oct 27 14:12:40 sunha2 hadf: NOTICE: cltrans_hadf_init: Computed
HA_NO_METASETSERVE = sunha2-L
Oct 27 14:12:40 sunha2 hadf: NOTICE: cltrans_hadf_init: Computed
HA_SIBLING_METASETSERVE = sunha2-L
Oct 27 14:12:40 sunha2 hadf: NOTICE: cltrans_hadf_init:
Computed HA_MAINT_DISKSET =
Oct 27 14:12:40 sunha2 hadf: NOTICE: cltrans_hadf_init: The registered Data
Services are:TMR_1
Oct 27 14:12:40 sunha2 hadf:NOTICE:cltrans_hadf_init:The registered Data
Services that are on are: TMR_1
Oct 27 14:12:41 sunha2 hadf: NOTICE: finished "step1" transition
Oct 27 14:12:41 sunha2 hadf: NOTICE: starting "step2" transition
Oct 27 14:12:41 sunha2 hadf: NOTICE: finished "step2" transition
Oct 27 14:12:41 sunha2 hadf: NOTICE: starting "step3" transition
Oct 27 14:12:42 sunha2 hadf: NOTICE: finished "step3" transition
Oct 27 14:12:42 sunha2 hadf: NOTICE: starting "step4" transition
Oct 27 14:12:43 sunha2 hadf: NOTICE: callmethod: Calling STOP method of data
service TMR_1
/etc/Tivoli/stop_TMR_1
Oct 27 14:12:44 sunha2 hadf:NOTICE:callmethod:Call to STOP method of data
service TMR_1 returned, exit code was 0
Oct 27 14:12:44 sunha2 hadf: NOTICE: finished "step4" transition
Oct 27 14:12:45 sunha2 hadf: NOTICE: starting "step5" transition
Oct 27 14:12:45 sunha2 hadf: NOTICE: finished "step5" transition
Oct 27 14:12:46 sunha2 hadf: NOTICE: starting "step6" transition
Oct 27 14:12:53 sunha2 hadf: NOTICE: finished "step6" transition
Oct 27 14:12:53 sunha2 hadf: NOTICE: starting "step7" transition
Oct 27 14:12:54 sunha2 hadf: NOTICE: finished "step7" transition
Oct 27 14:12:55 sunha2 hadf: NOTICE: starting "step8" transition
Oct 27 14:12:55 sunha2 hadf: NOTICE: callmethod: Calling START method of data
service
TMR_1 /etc/Tivoli/start_TMR_1
TME 10 Framework (tmpbuild) #1 Wed Aug 27 16:00:05 CDT 1997
Copyright Tivoli Systems, 1997. All Rights Reserved.
TMR 1710867091. ORB 1. TMR server local:94. Port 94.
9535
Tivoli daemon oserv started
Region = 1710867091
Dispatcher = 1
Interpreter type = solaris2
Database directory = /sunha1-L/1/spool/Tivoli/sunha1-L.db
Install directory =/sunha1-L/1/local/Tivoli/bin Inter-dispatcher encryption
level=simple
Kerberos in use = FALSE
Remote client login allowed = TRUE
Install library path =
/sunha1-L/1/local/Tivoli/lib/solaris2:/usr/openwin/lib:/sunha1-L/1/
local/Tivoli/
install/iblib/solaris2:/usr/openwin/lib:/usr/lib:/usr/ucblib:/sunha1-L/local/
Tivoli/lib/solaris2:/usr/openwin/lib:/usr/lib:/usr/ucblib
```

```
TME 10 Framework (tmpbuild) #1 Wed Aug 27 16:00:05 Copyright Tivoli Systems, 1997.
All Rights Reserved.
Port range = (not restricted)
State flags in use = TRUE
State checking in use = TRUE
State checking every 180 seconds
Dynamic IP addressing allowed = FALSE
Tivoli oserv object dispatcher up and running
Oct 27 14:12:59 sunha2 hadf: NOTICE: callmethod: Call to START method of data
service
Tivoli returned, exit code was 0
Oct 27 14:12:59 sunha2 hadf: NOTICE: starting "step9" transition
Oct 27 14:13:03 sunha2 hadf: NOTICE: cltrans_ipaddrs_up: sunha1-L now being
served by this machine
Oct 27 14:13:04 sunha2 hadf:NOTICE:cltrans_ipaddrs_up:sunha2-L now being serve
by sunha2
Oct 27 14:13:04 sunha2 hadf: NOTICE: finished "step9" transition
Oct 27 14:13:04 sunha2 hadf: NOTICE: starting "step10" transition
Oct 27 14:13:04 sunha2 hadf: NOTICE: finished "step10" transition
Oct 27 14:13:04 sunha2 hadf: NOTICE: starting "step11" transition
Oct 27 14:13:06 sunha2 hadf: NOTICE: finished "step11" transition
Oct 27 14:13:06 sunha2 hadf: NOTICE: starting "step12" transition
Oct 27 14:13:07 sunha2 hadf: NOTICE: finished "step12" transition
```

After the takeover has been completed, the configuration of the sunha2 node is changed in the following way:

- The diskset sunha1-L is now mastered and mounted on sunha2:

```
/sunha2-L/1/ (/dev/md/sunha2-L/dsk/d49): 3674164 blocks 949772
files
/sunha1-L/1 (/dev/md/sunha1-L/dsk/d10):7482454 blocks 1901010 files
```

- The logical hostname, sunha1-L, and its IP address has been moved to the interface hm0:2:

```
Routing Table:
Destination     Gateway          Flags  Ref   Use   Interface
----------------------------------------------------------------
127.0.0.1       127.0.0.1          UH     0    173  lo0
69.1.11.0       69.1.11.6          U      5   1363  hme0
69.1.11.0       69.1.11.10         U      5      0  hme0:1
69.1.11.0       69.1.11.9          U      5      0  hme0:2 <-----
204.152.65.0    204.152.65.2       U      2    545  hme1
204.152.64.0    204.152.64.2       U      2    545  hme2
224.0.0.0       69.1.11.6          U      5      0  hme0
default         69.1.11.240        UG     0    591
```

  - The hardware address has not been taken over, but the ARP entry relative to sunha1-L has been adapted to point to the right adapter:

    ARP table on the sunha2:

```
Net to Media Table
Device   IP Address     Mask      Flags   Phys Addr
----------------------------------------------------------
hme0   sunha1       255.255.255.255       08:00:20:7e:1f:40
```

```
hme0   sunha3        255.255.255.255       08:00:20:75:c5:17
hme1   sunha1-priv2  255.255.255.255       08:00:20:7e:1f:40
hme2   sunha1-priv1  255.255.255.255       08:00:20:7e:1f:40
hme2   sunha2-priv1  255.255.255.255 SP    08:00:20:89:d4:11
hme1   sunha2-priv2  255.255.255.255 SP    08:00:20:89:d4:11
hme0   sunha2-L      255.255.255.255 SP    08:00:20:89:d4:11
hme0   sunha1-L      255.255.255.255 SP    08:00:20:89:d4:11
hme0   sunha2        255.255.255.255 SP    08:00:20:89:d4:11
hme2   224.0.0.0     240.0.0.0       SM    01:00:5e:00:00:00
hme1   224.0.0.0     240.0.0.0       SM    01:00:5e:00:00:00
hme0   224.0.0.0     240.0.0.0       SM    01:00:5e:00:00:00
```

5.3.2 Managed Node - Idle Standby

This can be treated in exactly the same way as the TMR server Idle Standby.

5.3.3 Managed Node - Mutual Takeover

For idle standby configurations, there is very little difference (from the Tivoli point of view) between the Sun Solstice environment and the IBM HACMP environments. However, when it comes to mutual takeover scenarios, the method by which Solstice makes the logical hostname IP address available adds some differences to the installation and configuration process.

At installation time of a managed node, the install process will attempt to start the oserv process to initialize the Tivoli database. Despite the fact that the logical IP address was specified as the installation target, when the new oserv process attempts to contact the TMR server for the first time, it will use the physical IP address. This occurs because the logical IP address is only an alias that is added to the physical IP address. The TMR server will initialize the database with this physical IP address instead of the correct logical IP address. This can be seen by running the following command:

```
# odadmin odlist
Region      Disp  Flags  Port            IPaddr   Hostname(s)
1669180253     1    ct-    94       69.1.11.5
sunha3.austin.lab.Tivoli.com,sunh3
               2 ct-    94       69.1.11.7
sunha1.austin.lab.Tivoli.com,sunha1
```

For an idle standby configuration where only a single instance of the oserv process will ever be running on any given system, this does not present anything more than a documentation problem. However, when it becomes necessary to use the set_force_bind variable because you will be running in a mutual takeover environment. This mismatch of IP addresses does cause a problem.

If you change the value of the set_force_bind variable from its default of FALSE to TRUE and then attempt to restart the oserv process to have the change take effect, then the restart of the oserv will fail with the following error:

```
1999/02/23 11:12:30 +06:
!/Tivoli_tmr/usr/local/Tivoli/bin/aix4-r1/bin/oserv: odlist init failed.
internal resource corrupted. (54)
```

Any subsequent attempt to start the oserv process will fail with the same error. Unfortunately, you need to have the oserv process running in order to change the value of the set_force_bind variable. The mismatch between the name that is expected for the managed node within the database and that which is recorded in the odlist resulted in us having to reinstall the managed node.

There are two possible solutions to this problem. The first is to set the value of set_force_bind at install time. This has the effect of binding the oserv process to the logical IP address the first time it attempts to contact the server. This means that the `odlist` output matches the name of the managed node, and there is no mismatch to cause a problem. This is only currently possible from the command line as there is no option in the GUI install to set the value of this variable. The following command will set the set_force_bind variable to TRUE at install time:

```
wclient {path variables} @ForceBind@=1 {managed node name}
```

So, to give the full example of the command, we used:

```
wclient -c /cdrom/cdrom0 BIN=/sunha1-L/1/usr/local/Tivoli/bin
LIB=/sunha1-L/1/usr/local/Tivoli/lib MAN=/sunha1-L/1/usr/local/Tivoli/man
CAT=/sunha1-L/1/usr/local/Tivoli/msg-cat
DB=/sunha1-L/1/var/spool/Tivoli/db @ForceBind@=1
sunha1-L.austin.lab.Tivoli.com
```

If the managed node has already been installed, then it is necessary to change the `odlist` information before attempting to change the set_force_bind variable. This is done with the following steps:

1. We need to have IP address and hostname of the logical machine, which, in our case, is sunha1-L.

   ```
   odadmin odlist change_ip 1 69.1.11.9 TRUE
   ```

   This command will change the IP address of the dispatcher to sunha1-L.

2. We associate an additional hostname to the object dispatcher, which is the logical IP address.

```
odadmin odlist add_hostname_alias 1 69.1.11.9 sunha1-L
```

3. We now delete the old host name associated with the object dispatcher.

   ```
   odadmin odlist delete_hostname_alias 1 69.1.11.9 sunha1
   ```

Once these steps have been completed, the output of `odadmin odlist` should match the logical IP address and label. Once this is the case, you can change the value of the set_force_bind variable to TRUE. The stop and restart of the oserv will now be successful.

The name of the clients to be used should be sunha1-L and sunha2-L, respectively, and installation paths should be modified as follows:

On sunha1:

```
/usr/local --> /sunha1-L/1/usr/local/Tivoli
/var/spool --> /sunha1-L/1/var/spool/Tivoli
```

On sunha2:

```
/usr/local --> /sunha2-L/1/usr/local/Tivoli
/var/spool --> /sunha2-L/1/var/spool/Tivoli
```

Next, we define the new data services for managed nodes on the cluster, MN_1 on sunha1-L and MN_2 on sunha2-L.

Note

In this example, we will use the Solstice registration procedure using a START_NET method and not just START, as in the TMR server data service.

The difference is due to the calling sequence during the different phases of the takeover. The START method calls the start scripts just after the host performed the takeover of the diskset and mounted its file systems. The START_NET method, in addition to that, waits until the logical network addresses are up. For more details, see the product documentation, *Solstice HA 1.3 Programmer's Guide.*

The registration commands we issued were:

```
hareg -r MN_1 STAR_NET=/etc/Tivoli/start_MN_1,STOP=/etc/Tivoli/stop_MN_1

hareg -r MN_2 STAR_NET=/etc/Tivoli/start_MN_2,STOP=/etc/Tivoli/stop_MN_2
```

To activate the data services, invoke the following `hareg` commands:

```
hareg -y MN_1
hareg -y MN_2
```

The start and stop scripts are shown below:

stop_MN_1:

```
#! /bin/ksh
#
#  stop local oserv daemon
#
/etc/Tivoli/oserv.rc stop
#
#  test the RC and print the message
#
if test $? -eq 0
then
echo "Tivoli daemon stopped"
fi
```

script start_MN_1:

```
#! /bin/ksh
#
#  initialize the environment
#
. /etc/Tivoli/setup_env.sh
#
#  start oserv dispatcher
/etc/Tivoli/oserv.rc start
#
```

After the takeover of sunha1-L by sunha2, the sunha2 configuration is shown below:

- ARP table on sunha2:

```
Net to Media Table
Device   IP Address               Mask      Flags   Phys Addr
------ -------------------- --------------- ----- ---------------
hme0   sunha1               255.255.255.255       08:00:20:7e:1f:40
hme0   sunha3               255.255.255.255       08:00:20:75:c5:17
hme1   sunha1-priv2         255.255.255.255       08:00:20:7e:1f:40
hme2   sunha1-priv1         255.255.255.255       08:00:20:7e:1f:40
hme2   sunha2-priv1         255.255.255.255 SP    08:00:20:89:d4:11
hme1   sunha2-priv2         255.255.255.255 SP    08:00:20:89:d4:11
hme0   sunha2-L             255.255.255.255 SP    08:00:20:89:d4:11
hme0   sunha1-L             255.255.255.255 SP    08:00:20:89:d4:11
hme0   sunha2               255.255.255.255 SP    08:00:20:89:d4:11
hme2   224.0.0.0            240.0.0.0       SM    01:00:5e:00:00:00
hme1   224.0.0.0            240.0.0.0       SM    01:00:5e:00:00:00
```

```
hme0   224.0.0.0           240.0.0.0       SM    01:00:5e:00:00:00
```

- oserv dispatchers running on the sunha2:

```
root 4529  1 0 17:24:32 ? 0:03 oserv -p 94 -k /sunha2-L/1/local/Tivoli/sunha2-L.db
root 15411 1 0 09:34:23 ? 0:00 oserv -p 94 -k /sunha1-L/1/spool/Tivoli/sunha1-L.db
```

- `odadmin odlist` on sunha3 will produce the following output:

```
Region   Disp  Flags   Port IPaddr   Hostname(s)
1185035979 1    ct-   94 69.1.11.5 sunha3.austin.lab.Tivoli.com,sunha3
          2    ct-  94 69.1.11.9 sunha1-L.austin.lab.Tivoli.com
           3     ct-  69.1.11.10  sunha2-L.austin.lab.Tivoli.com
```

In addition to executing the `odadmin` shown above to verify that the oservs are running, we have performed the following functions to further test that our Tivoli managed nodes are indeed functioning properly.

- `wping` command from sunha3
- Stopping and starting the managed nodes dispatchers. The oserv daemons could be stopped either locally by using appropriate stop_MN script or from the TMR server using `odadmin shutdown` command. To restart them, we have to use the local start_MN script.
- Checking the properties of the managed nodes from the server will always show the same output for both nodes.

5.4 Solstice HA/Tivoli Considerations

Some considerations that we identified during our testing are listed below:

- No function or script based on the `hostname` command should be implemented in this environment. This would be misleading and point to the physical name of the machine that is not used to identify a TMR server or managed node. Instead, always use the logical hostname, which will be given by a local variable, WLOCALHOST.
- During a real takeover, a reboot of a server as well as running any of the following Solstice HA commands:
  - `hareg -y|n Data_Service`
  - `hactl`
  - `hastop`
  - `hastart`
  - `haswitch`

  will require a cluster reconfiguration. That is, not only the requested data service, but *all* data services running on the cluster will be stopped and restarted. Therefore, a single application cannot be failed over during testing. Be careful when adding a Tivoli service or any other service to an existing Solstice HA environment because testing the failover will cause other running applications on that node to also failover.

- A Solstice HA data service behaves like a rotating resource group—that is, a server which has, for any reason, released its data service won't get it automatically back when the server becomes active again.

5.5 Summary

In this chapter, we described several scenarios that we were successfully able to implement in our Solstice HA environment. The information in this chapter should be used in conjunction with the information in Chapter 3 to plan, design, and implement a Tivoli HA environment using Solstice HA.

Chapter 6. Microsoft Cluster

In this chapter, we will shall discuss the Microsoft Cluster and how Tivoli can be implemented in an NT HA environment. Unfortunately, in the current version of Tivoli (Version 3.6.1), there are limitations in the Tivoli implementation that do not allow us to implement Tivoli in the same way as we can in HA environments on Unix platforms. Specifically, the capability to run multiple oservs on a single NT system is severely limited. Therefore, mutual takeover scenarios are not practical at this time. The solutions we describe here will not be as complete as in the previous chapters. However, Tivoli does consider it important to enable their products for HA environments, and you can expect to see these limitations lifted in upcoming release.

This chapter also covers Microsoft Cluster server basics and clustering architecture.

6.1 What Is a Microsoft Cluster?

A server cluster is a group of independent servers managed as a single system for higher availability, easier manageability, and greater scalability. The Microsoft Cluster Server (MSCS) was known in early releases as *Wolfpack*.

6.1.1 What Is Microsoft Cluster Server (MSCS)?

MSCS is a built-in feature of Windows NT Server Enterprise Edition. It is software that supports the connection of two servers into a *cluster* for higher availability and easier manageability of data and applications. MSCS can automatically detect and recover from server or application failures. It can be used to move server workload to balance utilization and to provide for planned maintenance without downtime.

The initial release of MSCS supports clusters with two servers. A future version of MSCS is expected to support larger clusters.

6.1.2 How Does MSCS Provide High Availability?

MSCS uses software *heartbeats* to detect failed applications or servers. In the event of a server failure, it employs a *shared nothing* clustering architecture that automatically transfers ownership of resources (such as disk drives and IP addresses) from a failed server to a surviving server. It then restarts the failed server’s workload on the surviving server. All of this, from detection to restart, typically takes under a minute. If an individual application fails (but the server does not), MSCS will try to restart the application on the same server. If that fails, it moves the application's resources and restarts it on the other server.

MSCS does not require any special software on client computers; so, the user experience during failover depends on the nature of the client side of their client-server application. Client reconnection is often transparent because MSCS restarts the application using the same IP address.

If a client is using stateless connections, such as a browser connection, then it would be unaware of a failover if it occurred between server requests. If a failure occurs when a client is connected to the failed resources, then the client will receive whatever standard notification is provided by the client side of the application in use.

For a client side application that has *statefull* connections to the server, a new logon is typically required following a server failure.

No manual intervention is required when a server comes back online following a failure. When a server running Microsoft Cluster Server, say server A, boots, it starts the MSCS service automatically. MSCS in turn checks the interconnects to find the other server in its cluster, say server B. If server A finds server B, then server A rejoins the cluster and server B updates it with current cluster info. Server A can then initiate a failback, moving back failed-over workload from server B to server A.

6.1.3 Concepts and Terminology

In this section, we cover some of the terminology used in conjunction with the Microsoft Cluster server.

Shared Nothing:
Microsoft Cluster employs a *shared nothing* architecture in which each server owns its own disk resources (that is, they share nothing at any point in time). In the event of a server failure, a shared nothing cluster has software that can transfer ownership of a disk from one server to another.

Cluster Services:
This is the collection of software on each node that manages all cluster specific activity.

Resource:
It is the canonical item managed by the Cluster Service. A resource may include physical hardware devices, such as disk drives and network cards, or

logical items, such as logical disk volumes, TCP/IP addresses, entire applications, and databases.

Group:
A group is a collection of resources to be managed as a single unit. A group contains all of the elements needed to run a specific application and for client systems to connect to the service provided by the application. Groups allow an administrator to combine resources into larger logical units and manage them as a unit. Operations performed on a group affect all resources within that group.

Failback:
Failback is the ability to automatically rebalance the workload in a cluster when a failed server comes back online. This is a standard feature of MSCS. For example, say server A has crashed, and its workload failed-over to server B. When server A reboots, it finds server B and rejoins the cluster. It then checks to see if any of the cluster group running on server B would *prefer* to be running in server A. If so, it automatically moves those groups from server B to server A. Failback properties include information such as which group can failback, which server is preferred, and during what hours the time is right for a failback. These properties can all be set from the cluster administration console.

Quorum Disk:
A quorum disk is a disk spindle that MSCS uses to determine whether or not another server is up or down.

When a cluster member is booted, it searches whether the cluster software is already running in the network. In case it is running, the cluster member joins the cluster. If it is not, the booting member establishes the cluster in the network. A problem may occur if two cluster members are restarting at the same time, thus, trying to form their own clusters. This potential problem is solved by the *Quorum Disk* concept. This is a resource that can be owned by one server at a time and for which servers negotiate for ownership. The member who has the Quorum Disk creates the cluster. If the member that has the Quorum Disk fails, the resource is reallocated to another member, which in turn, creates the cluster. Negotiating for the quorum drive allows MSCS to avoid *split brain* situations where both servers are active and think the other server is down.

Load balancing:

Load Balancing is the ability to move work from a very busy server to a less-busy server.

Virtual Server:
A virtual server is the logical equivalent of a file or application server. There is no physical component in the MSCS that is a virtual server. A resource is associated with a virtual server. At any point in time, different virtual servers can be owned by different cluster members. The virtual server entity can also be moved from one cluster member to another in the event of a system failure.

6.1.4 Tivoli in an MSCS Environment

To set up an MSCS cluster, we need to have:

1. A common shared SCSI bus with proper termination at both ends. Also the drive letters should be assign to all disk resources on the shared SCSI bus and must have the same drive letter on both nodes before installing MSCS.
2. The nodes of an MSCS cluster must be connected by one or more physically independent networks.

The figure below shows two NT servers having a common SCSI storage.

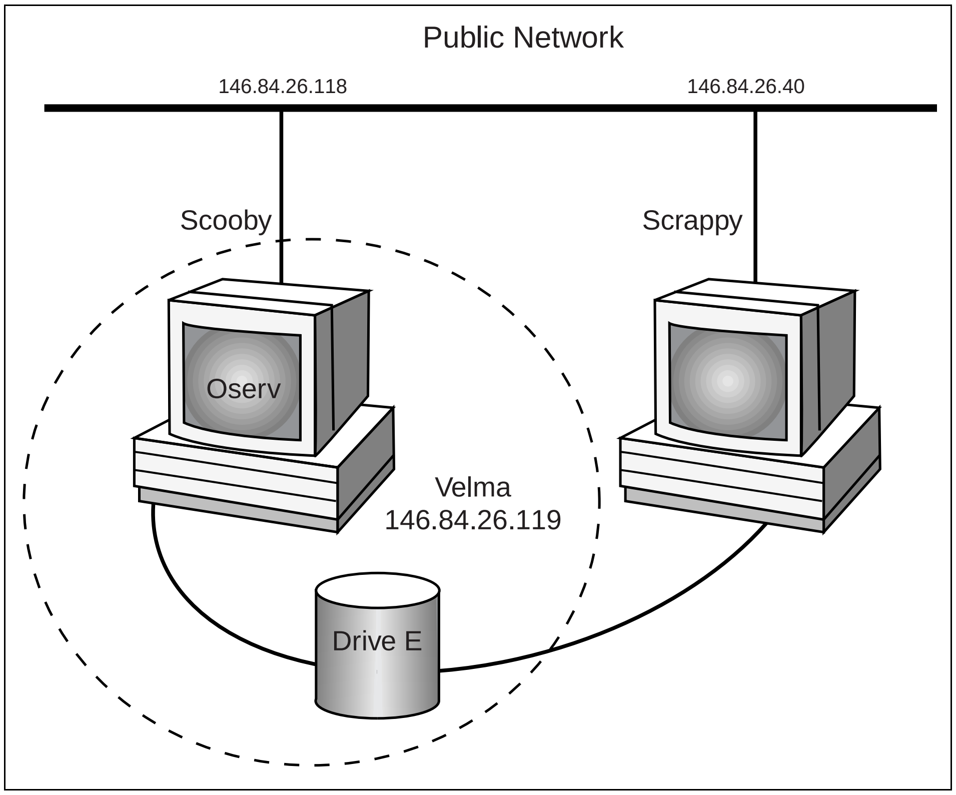


Figure 19. MSCS Environment

We have an MSCS cluster with two nodes scooby and scrappy. A resource group, Tivoli Group, is configured that contains drive E. Node scooby is the owner of this group. So, when both the machines comes up, scooby will own this group.

Important

The MSCS cluster does not support the use of IP addresses assigned from a Dynamic Host Configuration Protocol (DHCP) server for the cluster administration address (which is associated with the cluster name) or any IP address resource. However you can use either static IP addresses, or IP addresses permanently leased from a DHCP server, for the Windows NT network configuration on each node.

When Tivoli is installed on an NT machine, it copies the required files to the designated drive. Also, it creates two NT accounts at installation time. The accounts are the user tmersrvd and the group Tivoli_Admin_Privileges. These accounts are created locally on the machine. It also installs the Tivoli Authentication Package(TAP), which allows set uid methods to work on NT. Other than this, it places the following files under %SYSTEMROOT%.

1. %SYSTEMSROOT%\SYSTEM32

   sis_sh.exe

   TivoliAP.dll

   worldname.exe

2. %SYSTEMroot\system32\drivers\etc\Tivoli

   setup_env.sh

   setup_env.cmd

   tll.conf\arg

   tll.conf\layout

   tll.conf\library

   tll.conf\task

In an HA environment, these files need to be present on the other machine also. There are two ways of doing this.

- We can copy all these files to their respective directories to the desired location from the first node, or
- Instead of copying the files from the first machine to the second machine, we can make a dummy Tivoli Framework installation on the second machine by installing the Tivoli Framework code using the shared disk for the binaries and database and then deleting the binaries and database that we had just installed, which will leave us only with the above mentioned required files in the %SYSTEMSROOT%\SYSTEM32 and %SYSTEMROOT\SYSTEM32\DRIVERS\ETC\TIVOLI directories.

After this, we may need to copy the HOSTS file manually from one machine to other. This file is present in the %SYSTEM%\system32\drivers\etc.

In an NT clustering environment, resource groups move from one machine to the other. The resource group has an IP address associated with it. This is also called a *virtual machine*. In our case this virtual machine is velma. This virtual machine has a physical disk drive, network name, and Tivoli object dispatcher resources along with the IP address resource.

For these resources, we need to define dependencies in the following order.

Physical Disk Drive->IP Address->Network Name->Tivoli Object Dispatcher.

Once we have defined this, the following steps should be carried out.

- Tivoli service defined on the second machine.

  This is done through the Cluster Administrator menu. We only need to change the status of oserv to online. This will automatically start the dependencies first and then the oserv daemon.

- When we install Tivoli on the second machine, it is installed with IP address 146.84.26.118 and machine name scooby.

  We need to have the IP address and hostname of the virtual machine, which, in our case, is velma.

  ```
  odadmin odlist change_ip 1 146.84.26.119 TRUE
  ```

  This command will change the IP address of the dispatcher to IP address of velma.

- Associate an additional hostname to the object dispatcher.

  ```
  odadmin odlist add_hostname_alias 1 146.84.26.119 velma
  ```

- Delete the old host name associated with the object dispatcher.

  ```
  odadmin odlist delete_hostname_alias 1 146.84.26.119 scooby
  ```

- Copy the database from old name to new name.

  ```
  e:>copy \Tivoli\DB\scooby.db velma.db
  ```

Note

We could also use the documented way of changing the database directory as mentioned in the product documentation, *TME10 Framework Planning and Installation Guide.* But for our purposes, the above steps mentioned were sufficient.

- Change the registery entry in the existing machine so that it will point to the new database.

  This can be done using REGEDIT. These entries are available under

  HKEY_LOCAL_MACHINE-> SOFTWARE-> Tivoli PLATFORM-> OSERV94

- Change the label on the managed node representing the TMR server by executing the following command in the BASH environment.

```
MN=‘wlookup -r ManagedNode scooby‘
idlcall $MN _set_label ‘"velma"‘
```

- Change the environment to reflect the database change. Environment setup files define the DBDIR variable that specifies the path of the Database.

  These files are under \WINNT\SYSTEM32\DRIVERS\ETC\Tivoli
- Finally, through the cluster administrator we check dependencies on both machines and ensure that they are set in the following order: Physical Disk Drive->IP Address->Network Name->Tivoli Object Dispatcher
- Also add the following registry replication setting:

  `Software/Tivoli/platform/oserv94`

  Within cluster administrator properties, autostart for oserv should be disabled.

Now we are ready to do a failover. A failover can be done either by switching off the working machine or by moving the resource group to the other machine. Normally it takes less than a minute to failover, after which, oserv starts on the other machine.

6.2 Summary

In this section we tested a hot standby NT TMR server scenario using MSCS Software that has some implementation differences as compared to the Unix scenarios.

Appendix A. A Possible Future for Tivoli HA Implementations

Important

The materials presented here are tentative plans for changes in the Tivoli code in order to simplify HA implementations. These plans *may change* due to new customer requirements or other business reasons. We provide this information on an as is basis and to provide you with some insight to changes being considered by Tivoli.

It is likely that, in the near future, there will be some enhancements to the Tivoli products that will ease the implementation of mutual takeover environments on both Unix and NT Servers. Current official support for multiple oservs on Unix machines will be enhanced to also cover NT servers. The following sections explain the current plans for this implementation.

A.1 Background

Enabling the Tivoli framework to run on the Microsoft Cluster Server (MSCS) is Tivoli's next step in its support of high-availability (HA) environments. Adding support for MSCS requires a redesign of the current generic HA support as well as the addition of some MSCS-specific functionality. To assist the end-user in consistently using Tivoli in an HA environment, great measures have been taken to merge the differing characteristics of Unix and Windows NT. File and directory structure rearrangement and new command line interface tools are just two of the areas modified to achieve consistent environments on both Unix and Windows NT.

The model that the Tivoli framework uses to support MSCS is that all possible application resources are kept on a shared drive. Only one cluster member can access the shared drive at a given time. As control moves from one node to another, the control of the disk resource moves. Since the database files and binaries are installed on this hard drive, they become available to the machine that is taking over for a failed machine.

Changes in Tivoli's generic HA support include modifying the installation and communications code to enable multiple oservs on a machine. In most cases, users will want to use MSCS to provide more reliable access to a particular application or resource. For management of the machines themselves, the user will have a framework installation on each node. Then, the user will have their critical application set up to bounce between those physical nodes on a virtual host. There will be another Tivoli framework installation that will follow that application. This means that there will be multiple oservs on a machine, all using the same port. This brings out several limitations of the product.

A.2 Problems

Currently, the oservs on a given machine can be distinguished using the port number on which they listen. On NT, oserv services are named based upon their listening port. For local communication between a client program and an object dispatcher on the same machine the port number is used to select the appropriate communications channel (a FileMapping on NT and a pipe on Unix).

Since only one process can be listening on a given port at a time, this ensures that each oserv could be uniquely identified by its port number. This will no longer be the case. With the advent of support for clustering solutions and multi-homed hosts, the listening port is not enough to distinguish one oserv from another. The oserv must be able to listen on a given port and a particular network interface. If only the port number is used, it is easy to see that naming collisions will occur for oservs that are listening on the same port but on different interfaces.

On both NT and Unix, a method for selecting one of many network interfaces existed, but required you to choose a network interface for all oservs running on the machine. Changes are system-wide. For HA support, this is not sufficient. A requirement is that different oservs be able to listen on different network interfaces simultaneously. When HA support was added for Unix (set_force_bind parameter), this issue was addressed. However, NT still contained no support for multiple oservs running on different interfaces.

With respect to MSCS specifically, there is a significant amount of machine configuration done during installation that can not be maintained on the shared drive. This brings about a problem when a machine in the cluster fails and another machine tries to take over. Even though the new machine has access to the database files, the libraries and the binaries the oserv will fail to initialize due to the absence of the special user and group resources, values that were written into the registry, the Tivoli Authentication Package, and system DLLs.

A.3 Solutions

Our goal is to add support for multiple oservs running on multiple network interfaces and multiple ports simultaneously and to make the support consistent between NT and Unix. To that end, we plan to separate the notion of oserv identification from the listening port of the oserv. Each oserv installed on a machine will have an ID, which is simply a text string. This string is completely arbitrary in that it is not parsed for parameters (as was previously done on NT to find the port number) but is simply used as a unique key to identify that oserv.

This ID will be used in a number of places. Instead of client programs requiring a port number (`$o_dispatch`) that is used to find the local communication channel, they will require an oserv ID (`$OSERV`) that is used for the same purpose. The oserv will also use its own ID (taken from the service name that is used to start it on NT and a new command line flag on Unix) to look up its parameters, such as database directory, listening port, and network interface, in the registry.

To overcome the problem of machine specific configuration data outside of the shared drive, those data must be explicitly copied from a Tivoli installed cluster member to the others before the failover occurs. To make this copying easier on the user, a set of scripts will be provided that will automatically copy the necessary configuration items to another cluster member. Therefore, the user must install the Tivoli products that they would like to run on the cluster onto one of the nodes and then run a script that will copy these resources to the other cluster members.

A.4 Effects

A.4.1 oserv ID Generation

Currently, there is no notion of an ID on a Unix oserv, and the service name was the analogous concept in the NT world. The service name is generated using the port number. If there is no port number specified, the service name is simply oserv, and if a port is specified, the name is oserv-port.

After the modifications, the ID of the first oserv on a machine will be oserv unless otherwise specified. The next installation gets the ID oserv1 unless otherwise specified. This is followed by oserv2, oserv3, and so on. This same scheme will be used for both NT and Unix. The ID can be changed after the installation of the oserv using the `oregister-update` command.

A.4.2 Setup Scripts Directory

The setup scripts directory currently used is always set to /etc/Tivoli during a Unix installation and to %SystemRoot%\system32\drivers\etc\Tivoli-port on NT. Either of these can be overridden by the EtcTivoli environment variable during installation. Once again, there is an assumption that the port is enough to distinguish an oserv on NT and an assumption that there will be only one oserv per machine on Unix.

The default directory will be changed on both platforms. On Unix, the default location will be /etc/Tivoli/oserv_ID. On NT, %SystemRoot%\system32\drivers\etc\Tivoli\oserv_ID will be the default location. There will still be some files that reside in `/etc/Tivoli` on Unix, namely the bin directory and other files that are used by Tivoli's scripts. There will also be the new instances file that contains all of the parameters for all installed oservs, and the start and stop scripts that can start an oserv without the environment set up.

These default directories will still be overridden during installation.

A.4.3 Instances File (Unix only)

The /etc/Tivoli/instances file will contain all of the information regarding installed oservs on Unix. The parameters are used by the oserv if they are not specified on the command line.

The instances file will consists of several lines following the format:

>D:DBDIR:ETCDIR:PORT:HOSTNAME

where ID is the oserv ID that corresponds to all of the parameters on this line, DBDIR is the location of the database directory, ETCDIR is the location of the setup scripts, PORT is the port number, and HOSTNAME is a host name associated with the interface on which the oserv should listen.

Any of the values might be missing, in which case, the default values will be used.

A.4.4 Registry Keys (NT only)

The registry contains all of the information regarding the parameters of the oserv on NT. The database directory is always stored in the registry but under the SOFTWARE key. It will be stored with the Service Manager's data and the rest of the parameters under the SYSTEM key. Each oserv that is installed will have a service key associated with it. Under this key, a subkey named Parameters will be created that contains all of the data contained in the instances file on Unix. So, the Parameters key looks like:

```
(Default)"oserv"
```

```
dbdir"C:\Tivoli\db\jpatterson.db"
```

```
etcdir"C:\WINNT\system32\drivers\etc\Tivoli\oserv"

hostname"jpatterson"

port8101
```

As with the instances file on Unix, any of these values might be missing, which would cause the defaults to be used except the DBDIR, which has no default value.

A.5 Procedures

A.5.1 Fresh Install

During a fresh installation, several things will be done differently to support the new multiple dispatcher scheme. First, the setup scripts will be placed in a different directory by default. The previous default directory is /etc./Tivoli on Unix (which made it too easy to overwrite another oserv's setup scripts) and %SystemRoot%\system32\drivers\etc\Tivoli-port on NT (which won't be sufficiently unique since two oservs can be installed on the same port but using different interfaces). In the new scheme, a Tivoli directory will be created in the *etc* directory (this is `/etc` on Unix and `%SystemRoot%\system32\drivers\etc` on NT). Underneath this directory, the installer will create a directory named after the ID of the oserv for each framework installation. This will be the default location. The user will specify any location that they want for the installation-specific setup files.

On Unix, some additional files will be created in the /etc/Tivoli directory. The install script will create a file named instances that takes the place of the registry on Unix. It will hold the parameters for each of the registered oservs. Two scripts named start and stop that start or stop an oserv based on its ID will be generated. Lastly, the tll.conf directory and bin and lib symbolic links will be created for use by the framework PERL scripts. Since these files will be generic to all oservs on the machine, they will be created by the first oserv installed on that machine. If these files already exists when an oserv is installed, it will leave them alone.   The user will not be able to select another directory in which to install these files.

The default setup_env scripts will set the $OSERV environment variable to the ID of the target oserv instead of setting $o_dispatch (and $WLOCALHOST on Unix). This will be the only information that client programs will need to contact the correct oserv through the local communications channel. The default oserv.rc script on Unix will be modified to pass an ID on the command line instead of specifying the port, host name, and database directory on the command line.

In order to provide the scripts that create the setup and start scripts with enough information to fill in all of the oserv parameters, there will be a new advanced dialog during the install that gathers the new information (port, host name, oserv ID, and setup script location). This information will be still be specified through the environment variables ($o_dispatch, $WLOCALHOST, and $OSERV, $EtcTivoli, respectively) before the setup is run, but this only sets the default values in the advanced dialog. The user will still be given a chance to override the environment variables.

If the user is installing into an MSCS environment, they must install the desired Tivoli products as usual onto one of the nodes in the cluster. After completing the installation onto that one node, they will run the copy-cluster.sh script to replicate that configuration onto each other machine in the cluster. Only after this is done, will the other machines be able to successfully run the oserv.

It will be the user’s responsibility to create the necessary resources on their MSCS cluster before installing the Tivoli framework. At a minimum, the user must have done the following before installing the framework:

- Create an IP Address resource for the oserv to use (this IP address should appear in the user's DNS, the host name will be needed during framework installation)
- Create a Computer Name resource that will follow the oserv so that files can be shared from the machine that is running the oserv (this is optional)
- Create a Shared Disk resource on which to install all of the Tivoli framework files

After doing these things, the user is ready to run the Framework installation, specifying whatever ID they choose and a host name that resolves to the virtual IP address they have created. After installation, the user must create a Generic Service resource that corresponds to the oserv (or whatever ID was specified) service. They must also ensure that the dependencies are set properly: The oserv service depends on the IP address and the shared disk.

A.6 Summary

This appendix has provided an overview of changes to the framework that are being considered to make HA implementations much easier. Due to the current implementation and naming schemes on NT, running multiple oserv processes on a single NT system is not supported. With the changes outlined above, multiple oservs will be supported in both NT and Unix environments.

Multiple oservs running on the same system are critical for two reasons. One is for the support of a mutual takeover HA environment. The other is to allow one oserv to manage the physical machine, while another oserv could be used to manage logical resources, such as a critical application that may move between systems in an HA cluster.

Appendix B. Sample Script to Copy Tasks to Shared Filesystem

Following is the script written by Thomas Knueppel in the original Tivoli HA book *Implementing TME 10 in High Availability Environments* (SG24-2032-00) in order to move any created tasks from /usr/local/Tivoli/bin into the shared file system. Be sure to take note that the $REGIONNUMBER shell variable will need to be updated to the value shown by the `odadmin` command for your TMR.

```
#!/bin/ksh -x
# create links for tasks for all interpreter-types
# Version 1.0
# Thomas Knueppel, IBM Germany
# October 7th, 1998

. /etc/Tivoli/setup_env.sh
REGIONNUMBER='1925476444  '
BINPATH='/usr/local/Tivoli/bin/'# Directory, for Tivoli binaries
TASPATH='TAS/TASK_LIBRARY/bin/'# This is a static part of the path
# to the Tivoli tasks
TARGETDIR='/tivoli_tmr/usr/local/Tivoli/bin' # Target directory on shared
disk
if test -d $TARGETDIR
then
  echo "$TARGETDIR already existed"
else
  echo "$TARGETDIR doesn't exit, so I will create it"
  mkdir -p $TARGETDIR
fi

INTERPRETER[0]='aix3-r2'
INTERPRETER[1]='aix4-r1'
INTERPRETER[2]='dgux5-r1'
INTERPRETER[3]='dgux-ix86'
INTERPRETER[4]='generic'
INTERPRETER[5]='generic_unix'
INTERPRETER[6]='hpux10'
INTERPRETER[7]='hpux9'
INTERPRETER[8]='mips-irix'
INTERPRETER[9]='nextstep3-ix86'
INTERPRETER[10]='nw3'
INTERPRETER[11]='nw4'
INTERPRETER[12]='os2'
INTERPRETER[13]='os400-v3r1'
INTERPRETER[14]='os400-v3r6'
INTERPRETER[15]='osf-axp'
```

```
INTERPRETER[16]='solaris2'
INTERPRETER[17]='sunos4'
INTERPRETER[18]='sys4-att'
INTERPRETER[19]='sys4-m88k'
INTERPRETER[20]='u6000_svr4mp'
INTERPRETER[21]='uw2-ix86'
INTERPRETER[22]='w32-ix86'
INTERPRETER[23]='win3x'
INTERPRETER[24]='win95'

for i in ${INTERPRETER[@]}
do
  if test -d $BINPATH/$i/$TASKPATH/$REGIONNUMBER
  then
    echo "$i Directory for tasks alreaday exists"
    echo "So I will move the files to the shared directory"
    mkdir -p $TARGETDIR/$i/$TASKPATH/$REGIONNUMBER
    mv $BINDPATH/$i/$TASKPATH/$REGIONNUMBER/* \
       $TARGETDIR/$i/$TASKPATH/$REGIONNUMBER
    rm -rf $BINPATH/$i/$TASKPATH/$REGIONNUMBER
    ln -s $TARGETDIR/$i/$TASKPATH/$REGIONNUMBER
  else
    echo "There is no Directory for the interpreter $i"
    echo "So I will create it and will link it to the shared directory"
    mkdir -p $BINPATH/$i/$TASKPATH
    mkdir -p $TARGETDIR/$i/$TASKPATH/$REGIONNUMBER
    ln -s $TARGETDIR/$i/$TASKPATH/$REGIONNUMBER \
          $BINPATH/$i/$TASKPATH/$REGIONNUMBER
  fi
done
```

Appendix C. Determining the Status of the oserv Process

Often in HA environments, one of the challenges is how to determine when an application is no longer performing as expected and should be failed over. It is easy being able to monitor the existence of a particular process, but the existence of the process does not necessarily indicate that the application is healthy. For instance, if an application becomes deadlocked, or otherwise hangs, the process may still show as active.

This appendix provides some insight into what you might do to monitor the status of the TMR server to recognize when you might want to force a failover.

There is no absolute indication that the TMR server or overall TMR is not performing normally. The basis of the TMR is the oserv. The oserv must be running. It is a process that dynamically creates and deletes objects in its database. As it performs these operations, it maintains a `rollback` log file, $DBDIR/odb.log. If you examine the size of the $DBDIR/odb.log file, you should see the file size increase and shrink. If the oserv is inactive for a lengthy period of time, this file size should return to 0. The oserv will perform a purge of its internal buffers that get written to this file after 10 minutes of inactivity. The user can force this same operation by issuing the command:

```
odadmin db_sync
```

If, over time, you see the $DBDIR/odb.log file growing, then there is a concern that a transaction has not completed. Steps should be taken to resolve the problem.

Issuing the command `odstat -c` or `odstat -cv` will list the active methods with their current states, times they were started, and the OID of the method's object. Some of these methods are quite static. They are started when the TMR server is initialized. For example, there is a method for the scheduler, endpoint manager, and gateway, just to name a few.

Other methods that persist for a long period of time are uiserver (the desktop) and check_db. You may see these persist for hours, even days. They should normally show a state of *run.*

The third class of methods are transient. They can last a few seconds, hours, or even days. The state of these methods becomes a concern. If the method is in any state other than run, one may need to determine if the oserv on the managed node is running. If a simple wping hostname of that manage node fails, then you may need to recycle the oserv on that managed node to reestablish the connection with the TMR server and clear the hung methods. (In some cases, you may actually need to recycle the oserv twice).

Understanding what started a method and what a method does helps determine if the method is hung up and will not complete. A runaway method that is not completing as it should, but continues to process (in a loop, for instance), can slowly diminish available resources and eventually cause the oserv to hang. The states most often seen for such methods are rwait, waiting on a managed node, and ali, waiting for the TMR server to authenticate the user of the method. The rwait message is caused by the managed node's oserv not responding. The ali message is caused by the TMR server being to busy to process all of the requests.

If recycling the managed node does not cause the problem of the growing $DBDIR/odb.log file to abate, then examine the output of odstat -c for other methods that have not completed. Are they running on the TMR server? If they are, examine the output of odstat -cv for the process ID (pid). If it is feasible, you might try using a kill -15 (not -9), on the process. If the state of the oserv does not start cleaning up, you may need to recycle the TMR server.

Depending on the size of the $DBDIR/odb.log file, the startup may take some time. The oserv replays, attempts to rollback, all non committed transactions. Be somewhat patient. Examine the size of the $DBDIR/odb.log file before the TMR server gets busy. Did the size go to 0? You may have to stop and start the oserv again. Some transactions depend on others, and the order that they were processed made it impossible to complete the first time the oserv tried to perform a rollback on them.

Another indicator of TMR server health may be the command `tmstat`. The output of this command is not easy to read. The volume of output from this command depends greatly on how active the TMR is. If tmstat produces no output, then there are no transactions and/or locks at the present time.

Transactions and/or locks can last from a few seconds to hours or even days depending upon what processes are running. Perform a simple operation, such as running a task on a managed node, and look at the output of `tmstat`. You will see a section showing the transactions and a section showing the locks. The transaction section will have an entry such as:

```
{2020203456:1,2020203456:5,20:3} Top-T running Yes No running running 1073
```

The value {2020203456:1,2020203456:5,20:3} is the transaction ID. 2020203456:1 is the resource host and its dispatcher number. 2020203456:5 is the resource locker that is managed node with dispatcher 5. (If the 10-digit number does not match the 10-digit number of the resource host, then the resource locker is on a different TMR.)

20:3 is the unique identifier number for this transaction. The Top-T specifies that this is a top-level transaction. Finally, 1073 is the method thread identifier.This can be seen in the output from odstat -c and can tell you what method is running.

In the lock section of `tmstat` output, you can see the resource that is locked and the transaction ID. You will have to look at the transaction section to go back from there.

A few guidelines that will help prevent deadlock situations from occurring include:

- In a multiply connected TMR situation, do not run wchkdb on both TMRs at the same time.
- In a multiply connected TMR situation, do not run wchkdb on 1 TMR and wupdate on the other at the same time.
- In a multiply connected TMR situation, do not run wupdate on both TMRs at the same time.
- Always look for processes that may block other processes, then do no run on them concurrently.

Appendix D. Special Notices

This publication is intended to help Tivoli and High Availability specialists to plan for and implement Tivoli in HA environments. The information in this publication is not intended as the specification of any programming interfaces that are provided by the various Tivoli products and High Availability products we described. See the the specific product announcements for more information about what publications are considered to be 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 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 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

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 107.

- *An Introduction to Tivoli’s TME 10 , SG24-4948*
- *Bullet-Proofing Your Oracle Database with HACMP: A Guide to Implementing AIX Databases with HACMP, SG24-4788*

E.2 Redbooks on CD-ROMs

Redbooks are also available on the following CD-ROMs. **Order a subscription** and receive updates 2-4 times a year.

CD-ROM Title	**Collection Kit Number**
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbook	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037

E.3 Other Publications

The following publications ship with the Tivoli software and contain important and relavant information:

- *TME 10 Framework Planning and Installation Guide*
- *TME 10 Framework User’s Guide*
- *TME 10 Framework Reference Manual*

- *TME 10 Enterprise Console User’s Guide*
- *TME 10 Enterprise Console Reference Manual*

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** `http://www.redbooks.ibm.com/`

  Search for, view, download or order hardcopy/CD-ROM redbooks from the redbooks web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

  Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders via e-mail including information from the redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the "How to Order" section at this site: `http://www.elink.ibmlink.ibm.com/pbl/pbl/`

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the "How to Order" section at this site: `http://www.elink.ibmlink.ibm.com/pbl/pbl/`

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the "How to Order" section at this site: `http://www.elink.ibmlink.ibm.com/pbl/pbl/`

This information was current at the time of publication, but is continually subject to change. The latest information for customer may be found at `http://www.redbooks.ibm.com/` and for IBM employees at `http://w3.itso.ibm.com/`.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at `http://w3.itso.ibm.com/` and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may also view redbook. residency, and workshop announcements at `http://inews.ibm.com/`.

IBM Redbook Fax 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.

Index

ITSO Redbook Evaluation

High Availability Scenarios for Tivoli Software
SG24-2032-01

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 online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
_ **Customer** _ **Business Partner** _ **Solution Developer** _ **IBM employee**
_ **None of the above**

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!)**

SG24-2032-01
Printed in the U.S.A.

High Availability Scenarios for Tivoli Software

SG24-2032-01