8.3 Appendix C. Serviceguard package configuration files ............................................................ 29
8.4 Appendix D. Serviceguard configuration steps ........................................................................ 29
8.5 Appendix E. Serviceguard test commands .............................................................................. 33
8.6 Appendix F. VCS main configuration file (main.cf) ................................................................. 34
8.7 Appendix G. Power cycle test detail using the VERITAS SANPoint Foundation Suite .......... 36
9 For more information .................................................................................................................. 39
1 Introduction

This document describes the results of a project jointly undertaken by TIBCO Software and Hewlett Packard to test various fault tolerant configurations of TIBCO’s Enterprise Message Service (“EMS”) servers. EMS is TIBCO’s high-performance and robust implementation of the Java™ Message Service (JMS), the messaging element of J2EE. During the course of the project various fault tolerant configurations were installed, configured and tested, with the objective of documenting the process and ultimately determining the adequacy of each alternative. Configurations used both EMS built-in fault tolerant features and third party high availability clustering software.

The document has multiple sections. The current section introduces the document. Section 2 gives a brief overview of the EMS product. Section 3 gives an overview of the hardware and software configuration of the testing environment. Section 4 describes specific configurations tested and gives the results of related tests. Section 5 summarizes and compares the whole set of results. Section 6 points the reader to additional resources. The remaining sections provide supplementary information too detailed to be included in the main text.

TIBCO Software Inc. is a leading independent Business Integration software company. TIBCO’s proven Enterprise Backbone, Business Integration and Business Optimization solutions are in use by over 2,000 customers around the world. TIBCO has more than 15 years of experience in creating Enterprise Messaging solutions and TIBCO products are employed in a wider range of industry applications than any other messaging product.

2 TIBCO EMS product overview

This section describes EMS’s main features and functionality and gives a brief overview of product installation.

2.1 EMS features and functionality

Enterprise Messaging is defined as messaging infrastructure, implemented in middleware, that enables flexible, real-time communication among applications and individuals engaged in business process information flows typically associated with integrating diverse and otherwise incompatible enterprise applications.

This type of messaging, driven most recently by standards such as JMS and XML, is becoming the de facto standard communication backbone for enterprises and therefore must have at least the following mission-critical attributes: comprehensive connectivity options and qualities of service, support for transactions, high performance, high scalability, load-balancing and fault-tolerance, and security and manageability.

Many vendors claim messaging capabilities with one or more of the above attributes; the ultimate proof is in success in the marketplace. TIBCO Enterprise Messaging is widely deployed in the most demanding enterprise environments, with an installed base rivaled only by IBM. Currently, more than 1900 organizations use TIBCO messaging as part of their corporate backbone and more than 50 Independent Software Vendors (ISVs) embed TIBCO messaging as part of their solution.

EMS employs a store-and-forward architecture, supporting queue-based and publish/subscribe messaging, local messaging transactions (in which multiple messages may be sent or consumed as an atomic unit of work), message selectors, and more. In the EMS architecture, an EMS server handles message processing (using topics or queues); applications wishing to send and receive messages connect to the server using EMS client libraries, as in Figure 1. Multiple servers can be deployed to balance load and achieve fault tolerance.
EMS is an implementation of the Java Message Service. Generic JMS offers publish-subscribe and request-reply (known in JMS terminology as “point-to-point”) messaging. JMS publishers send messages to “topics” that are sent to an arbitrary number of subscribers. JMS offers a “durable” subscription option that is similar to the Rendezvous certified class of service. Durable subscriptions can receive messages even when the subscriber is not active (the messages are held until the subscriber becomes active).

JMS point-to-point messaging is queue-based. Messages sent to queues remain in queues, until retrieved by a receiver. Receivers can be “exclusive” or “non-exclusive”. In the former, there is only one recipient of a message; in the latter, there may be multiple recipients (these variants correspond to Rendezvous request-reply/certified messaging and distributed queues).

EMS offers a number of extensions to the generic JMS specification. JMS messaging ensures delivery of messages in virtually all circumstances. The costs of ensured delivery are:

- Increased network traffic and latencies, due to the requirement for and associated delays of acknowledgement messages, and
- Increased storage for holding messages at the source until they are acknowledged.

For many applications these costs are undesirable and unnecessary. To avoid them TIBCO offers two extensions to base JMS: Reliable Message Delivery and No-Acknowledgement Message Receipt. These provide a class of service analogous to Rendezvous Reliable mode. Together they eliminate the need for acknowledgement messages and local storage allowing higher message volume, better utilization of system resources, and higher message rates.

EMS allows the message body (not headers and properties) to be compressed by the EMS client library before the message is sent to the EMS server. Message compression is useful when messages will be stored on the server (persistent queue messages, or topics with durable subscribers) and when messages are large. When messages are compressed and then stored, they are handled by the server in the compressed form. Compressed messages consume less space on disk and are handled faster by the EMS server. It is best to use compression when message bodies are large and messages are stored on a server. Because compression requires processing, the time to send and receive compressed messages is generally longer than the time to send the same messages uncompressed.

For further product details see:
http://www.tibco.com/software/messaging/enterprise.messaging.service/
2.2 EMS installation and configuration

EMS requires the Java Runtime Environment (JRE) 1.2. or higher (1.3 or higher recommended). TIBCO Rendezvous 6.6 or higher is required if the Rendezvous routing feature of the server is used. If Rendezvous is already installed on the network, additional licenses are required only if additional hosts (client, server, development subnets, and so on) are being deployed.

The following system requirements are recommended for production systems. The actual requirements depend on the application environment.

- Hardware memory: 256MB of RAM recommended
- Disk space for installation: 60MB
- Disk space for messaging: 256MB minimum

The disk space for messaging is used for the EMS data files. These files contain saved messages, information on client connections and data delivery metadata. These files may be owned by a single EMS server or shared by multiple servers. In the latter case, depending on the configuration, either EMS intrinsic mechanisms or those of third-party clustering systems are used to enforce consistency and coherency of the data. The particular mechanism will be discussed for each configuration reviewed below in Section 4.

EMS can be installed using a variety of modes. To install it using the GUI mode, the following is typed in a terminal window (after, of course, downloading or otherwise obtaining the install package):

```
./TIB_ems-simple_4.1.0_platform.bin
```

Refer to TIBCO Enterprise Message Service Installation for complete installation instructions.

EMS’s main configuration files are:

- `tibemsd.conf` – the main EMS server configuration file
- `topics.conf` – used to configure topic definitions
- `queues.conf` – used to configure queue definitions

3 Test environment overview

This section describes two broad approaches to deploying EMS in a fault tolerant configuration, and the hardware and software lab environment put in place to test specific configurations based on these approaches.

3.1 Fault tolerance approaches

EMS fault tolerant configurations can be achieved using 1) EMS built-in features or 2) those of a third party clustering system.

The built-in EMS fault tolerant features provide a clustering capability enabling primary and backup servers. The primary and backup servers act as a pair, with the primary server accepting client connections and performing the work of handling messages, and the secondary server acting as a backup in case of failure.

The following TIBCO documents describe these techniques in more detail:

TIBCO Enterprise Message Service User’s Guide (Chapter 13, “Making the Server Fault Tolerant”)

Configuring TIBCO EMS: Fault-Tolerant and Load-Balanced
Third party products can ensure EMS fault tolerance. Examples of third party products include HP Serviceguard and Symantec products including the VERITAS Cluster Server and Cluster File System. The vendor documentation listed in Section 6 describes these products completely.

3.2 Hardware environment

This subsection gives an overview of the lab hardware environment. Section 4 gives configuration specific information for each configuration tested.

The diagram in Figure 2 gives a high level view of the testing environment.

---

<table>
<thead>
<tr>
<th>Server hardware</th>
<th>HP 9000 machines</th>
<th>HP Integrity machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>rp3440</td>
<td>rx4640</td>
</tr>
<tr>
<td>CPU</td>
<td>PA8800 – 1GHz</td>
<td>Intel® Itanium® 2 – 1.5GHz</td>
</tr>
<tr>
<td># of CPUs</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Random Access Memory (RAM)</td>
<td>8 GB</td>
<td>64 GB</td>
</tr>
<tr>
<td>Network Cards</td>
<td>2 x 10/100/1000 NIC</td>
<td>2 x 10/100/1000 NIC</td>
</tr>
<tr>
<td>Internal Disks</td>
<td>3 x 36GB 15K RPM</td>
<td>2 x 73GB 15K RPM</td>
</tr>
<tr>
<td>Storage hardware</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>HP StorageWorks EVA3000</td>
<td>2 Controllers, Four 2 Gb FC-AL ports per controller in redundant pairs, 512KB Cache per controller, Redundant Cache, 28 x 72GB @ 15K RPM disks. Transfer Rate: With dual 2 Gb host interfaces per controller, each controller is capable of 8 Gb aggregate data transfer rates</td>
<td></td>
</tr>
<tr>
<td>HP StorageWorks NAS 1200</td>
<td>Microsoft® Windows® Storage Server 2003 models: 320 GB, 640 GB and 1 TB (raw capacity), CPU Intel Pentium® 4 2.4 GHz, Memory: 512-MB 200 MHz PC2100 DDR SDRAM, NICs: Two 10/100/1000 WOL network interface controllers. Hot plug drives: Four 80 GB, 160 GB, or 250 GB 7200 rpm hard disk drives</td>
<td></td>
</tr>
<tr>
<td>HP FC10 Fibre Channel JBOD</td>
<td>DS2405 is the recommended replacement for the discontinued FC10. 2 Gb Fibre Channel technology with speeds up to 300 MB/s. 15 disks (up to 1 TB). Redundant power supplies, fans, and I/O cards. Hot swappable disks, power supplies, fans, and I/O cards</td>
<td></td>
</tr>
<tr>
<td>HP StorageWorks SAN Switch 2/16</td>
<td>16 high performance Auto Sensing 1 and 2 Gb Fibre Channel ports, fully non-blocking, provides 32 Gb switching capacity for uncongested sustained, 2 Gb full duplex throughput, redundant power supplies and cooling</td>
<td></td>
</tr>
</tbody>
</table>

3.3 Software environment

This section describes common software elements; the following section (4) augments this discussion with configuration-specific information for each configuration tested.

HP-UX was installed on all servers. The HP-UX 11.11 Mission Critical Operating Environment (MCOE) was installed on the HP 9000 (PA-RISC based) servers along with applicable current patch bundles. The MCOE includes the Serviceguard software.

The HP-UX 11.23 MCOE, with current patch bundles, was installed on the Integrity (Itanium based) servers.

EMS version 4.1 was installed on all servers.

See the Appendices for further details on installed software.

The root volume group (/dev/vg00) of the Integrity servers was built with HP-UX Logical Volume Manager (LVM). LVM is the legacy storage management product on HP-UX. Included with the operating system, LVM supports the use of MirrorDisk/UX, an add-on product that allows disk mirroring. LVM was also used for the non-root disks on the Integrity servers.

The root volume group on the HP 9000 servers was built with the VERITAS Volume Manager (VxVM) as well as all non-root disks (Note: VERITAS requires that the VERITAS root disk group “rootdg” be configured in addition to the HP-UX root disk on any system using VERITAS Volume Manager products.) VxVM was used to support the disk requirements of the VERITAS clustering products installed.

On all systems the VERITAS File System (VxFS) was used. VxFS provides:

- Extent-based space management that concisely maps files up to a terabyte in size
- Fast recovery from most system crashes using a self-clearing intent log to track recent file system metadata updates
- Online administration capability that allows file systems to be extended and defragmented while they are in use
- Quick I/O features that allow aware database managers to bypass kernel locking by treating files as raw partitions, and enable 32-bit applications to take advantage of a system cache larger than 4 gigabytes
4 Tested configurations and results

This section describes the specific configurations built and tested, and the results of the testing.

Two configurations using EMS built-in capabilities were tested:
- A failover server configuration with both EMS servers on the same system (HP-UX/PA or Itanium)
- A failover server configuration with each EMS server on its own system – with Network Attached Storage and CIFS filesystem (HP-UX/PA, or Itanium)

Three configurations using third party clustering systems were also tested:
- A parallel server configuration, using a Cluster File System, VERITAS’s SANPoint Foundation Suite product on HP-UX/PA and shared FC10 storage
- A failover server configuration, using HP Serviceguard/HP-UX/Itanium and shared HP StorageWorks Enterprise Virtual Array (EVA) storage
- A failover server configuration, using VERITAS Cluster Server/HP-UX/PA and FC10 storage

4.1 Two EMS servers on the same system

Two EMS servers are run on the same system. This is a trivial case of fault tolerance and clearly offers only a degree of protection in the event of a software failure or other problem with the primary EMS server.

4.1.1 Configuration overview

Two EMS servers run on the same server using the same network name but different ports. The server initializing first becomes the active primary – the other becomes a passive standby. The standby will become the active server if the primary fails. The secondary detects a primary failure via loss of heartbeat or loss of a TCP connection it maintains with the primary. The servers each have access to shared EMS data configuration files on the server’s internal disk. The primary server locks this shared state upon initialization and the secondary can only gain access to the data upon failure of the primary.

4.1.2 Tests and results

A client connected to the primary server using the URL: tcp://7222,tcp://7224. When the primary server daemon was killed, the client was reconnected to the standby server listening on port 7224.

4.2 Two EMS servers on separate systems using the VERITAS SANPoint Foundation Suite

VERITAS SANPoint Foundation Suite HA enables coordinated access to shared data in clusters. SPFS/HA integrates VxVM and VxFS with intercommunication technologies from the VERITAS Cluster Server (see Section 4.5) and adds unique technologies of its own (Cluster Volume Manager and Cluster File System), to provide shared volume and shared file system access to the nodes of a cluster.

Like VxVM, Cluster Volume Manager (CVM) is capable of managing both physical disks and the virtual disks exported by hardware RAID array subsystems. CVM adds the following features to Volume Manager:
- Simultaneous access to volumes from multiple servers
- Clusterwide logical device naming
- Consistent logical view of volume state from all servers
- Volume management from any server in the cluster
Volumes that remain accessible by surviving servers after server failures
- Application failover without volume failover

Cluster File System (CFS) extends the features of VxFS to clusters of servers and adds the following:
- Clusterwide freezing of file system state, allowing operations that require a consistent on-disk image of a file system to be performed in a cluster environment
- Both clusterwide and local file system mounting, allowing administrators to share data among cluster nodes or not as application requirements dictate
- Rolling upgrades of Cluster File System itself, so that it can be upgraded node by node
- Lock management. For parallel applications, CFS provides shared data to all application instances concurrently. Applications coordinate their access to this data via intrinsic locking features or via CFS’s Group Lock Manager (GLM). GLM reproduces UNIX® single-host file system semantics in clusters. Applications can never retrieve stale data, or partial results from a previous write

4.2.1 Configuration overview
SANPoint Foundation Suite (SPFS) was installed on rpnode1 and rpnode2 (Figure 3). An HP FC10 Fibre Channel disk enclosure was used as the shared storage device for this configuration.

Figure 3. Two EMS servers with shared local storage

EMS configuration details are included in the following section.

4.2.2 Tests and results
Three categories of testing were performed: failover, performance and startup. The following sections describe the tests and their results.
4.2.2.1 Test set 1: Failover tests

The goal of this test set was to verify that a secondary EMS server will take over operation in case of software or hardware failure. EMS service failover should be transparent to EMS client user applications.

The high availability tests intentionally force failure of the primary EMS server while the topic publisher and subscriber are running. A test passes when the secondary EMS server takes over and the client publisher and subscriber run to completion without message loss. The failover is transparent to the publisher and subscriber.

All EMS Server failover tests passed using the VERITAS Cluster File System on HP-UX 11i v1.

4.2.2.1.1 EMS configuration detail

The test used one topic publisher and one topic subscriber. The topic publisher used PERSISTENT delivery mode sending 1Kbyte messages and the topic subscriber used AUTO_ACKNOWLEDGE mode. The topic subscriber was started before the topic publisher.

The subscriber was started with the following command:

```
java tibjmsMsgConsumerPerf --server <server1URL,server2URL> -topic topic.test --count 100000
```

The publisher was started with the following command:

```
java tibjmsMsgProducerPerf --server <server1URL,server2URL> -topic topic.test --delivery PERSISTENT --count 100000
```

Tibemsd.conf was modified to contain the following settings:

```
store = < directory of the mounted shared disk>
ft_reconnect_timeout = 3600
```

The test uses a failsafe topic destination to avoid data loss in the case of software/hardware failure. Topics.conf was modified to contain the following settings:

```
topic.test failsafe
```

4.2.2.1.2 Test 1: Only one process can lock the file

When a file is locked by one process that is running, no other process on the same or a different machine can obtain the lock on that file.

| Test Setup | 1. Configure two EMS servers on different machines with the same data store located on the shared storage. The second EMS server to start has an “ft_active” parameter configured to a non-existing EMS server URL.  
| | 2. Start the first EMS server; it becomes active.  
| | 3. Start the second EMS server; it tries to become active by locking the data store file. |
| Expected Result | The file-lock attempt by the second server should fail and it should not become active because the first EMS server has already locked the file. |
| Result | Pass |

4.2.2.1.3 Test 2: File lock release upon process failure

The storage system should release the file lock when the owner process has terminated. Killing the primary EMS server process should result in failover to the standby server.
4.2.1.4 Test 3: Maintain file lock during network disconnect
A temporary loss of network communication can cause the secondary EMS Server to attempt to become the primary. The lock owner, the primary EMS server, is still operational. Thus, the correct behavior is that the storage system maintains the file lock of the primary EMS server and prevents the standby EMS server from getting the lock.

| Test Setup | 1. Run two EMS servers as a fault-tolerant primary/standby pair with proper configuration and data store on the shared storage.  
2. Unplug the network cable of the machine on which the primary EMS server is running. |
| Expected Result | The standby EMS server on another machine will try to obtain the file lock after missing the heartbeats. The correct behavior is for the storage file system to maintain the lock even if the primary EMS server becomes unreachable. The standby server will fail to acquire the lock and remain in standby mode. |
| Result | (DID NOT RUN THIS TEST) Using this hardware configuration, this test is equivalent to test 1 above. (PASS) |

4.2.1.5 Test 4: Release file lock in case of hardware failure (/etc/reboot)
The file lock must be released when there is a hardware failure on the machine running the primary EMS server.

| Test Setup | 1. Run two EMS servers as a fault-tolerant primary/standby pair with the proper configuration and data store on the shared storage.  
2. Start an EMS message consumer client expecting N messages on a third machine using FT connection to the EMS FT server pair.  
3. Start an EMS message producing client on the third machine publishing N persistent messages onto the topic listened to by the above consumer, using a FT connection.  
4. Cause an ungraceful shutdown (use /etc/reboot) of the machine on which the primary EMS server is running while the clients (2 and 3) are still running. |
| Expected Result | – The secondary EMS server on another machine acquires the lock and becomes active without waiting for the primary server machine to reboot.  
– The EMS message consumer and producer clients both complete N messages. |
| Result | Pass – Failover time was approximately 20 seconds |

4.2.1.6 Test 5: Release file lock in case of hardware failure (simulated power failure)
The file lock must be released when there is a hardware failure on the machine running the primary EMS server. This test is similar to the previous test. (The method of hardware failure is different)
Test Setup

1. Run two EMS servers as a fault-tolerant primary/standby pair with the proper configuration and data store on the shared storage.
2. Start an EMS message consumer client expecting N messages on a third machine using FT connection to the EMS FT server pair.
3. Start an EMS message producing client on the third machine publishing N persistent messages onto the topic listened to by the above consumer, using a FT connection.
4. Cause an ungraceful shutdown (power cycle) of the machine on which the primary EMS server is running while the clients (2 and 3) are still running.

To cycle the power from the console:

- Login to the console.
- Type CM at the MP> prompt to enter Command Mode.
- At the CM> prompt type PC (Power Control)
- Select OFF to Turn Power Off.

Expected Result

– The secondary EMS server on another machine acquires the lock and becomes active without waiting for the primary server machine to reboot.
– The EMS message consumer and producer clients both complete N messages.

Result

Pass – Failover time was approximately 20 seconds for the secondary to take over, but the clients took 10 minutes to reconnect to the secondary server. The initial configuration of the EMS server had to be changed and the test rerun. After rerunning the test with the modified configuration, the test passed. (PASS). See appendix G for test details.

4.2.2.2 Test set 2: Performance tests

The goal of this test set was to measure throughput of the EMS system when using the VERITAS Cluster File System software. The tests measured the message rate of producers and consumers when varying message size, number of threads, and failsafe vs. non-failsafe destinations. A failsafe destination uses synchronous I/O in EMS. A non-failsafe destination uses asynchronous I/O in EMS.

Disk performance is usually the primary factor affecting EMS server throughput (especially when using failsafe destinations). The “diskperf” tool was used to measure disk performance. The diskperf tool uses one thread to write to disk.

$ ./diskperf -block <blocksize> -count 10000 -<fsync> -file /ems/diskperf.test -keep –prealloc

Table 1. HP FC10 Fibre Channel Disk I/O Write Rate (Writes/Sec - Bytes/Sec)

<table>
<thead>
<tr>
<th>Blocksize</th>
<th>102 bytes</th>
<th>1K</th>
<th>10K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write (fsync)</td>
<td>99.86 - 10186</td>
<td>109.07 - 111689</td>
<td>91.08 - 932690</td>
</tr>
<tr>
<td>Write (no fsync)</td>
<td>163.19 - 167105</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$ ./diskperf -block 1024 -count 10000 -file /ems/diskperf.test -keep –read

Table 2. HP FC10 Fibre Channel Disk I/O Read Rate (Read/Sec – Bytes/Sec)

<table>
<thead>
<tr>
<th>Blocksize</th>
<th>1K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>109890.11 - 112527472</td>
</tr>
</tbody>
</table>

The disk used for these EMS tests was somewhat slow. For comparison, the disk performance numbers for similar tests using HP StorageWorks EVA3000 are listed below.
Table 3. HP StorageWorks EVA3000 Disk I/O Rate (IO/Sec – Bytes/Sec)

<table>
<thead>
<tr>
<th></th>
<th>1K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write (fsync)</td>
<td>913.66 - 935587</td>
</tr>
<tr>
<td>Write (no fsync)</td>
<td>3941.66 - 4036263</td>
</tr>
<tr>
<td>Read</td>
<td>212765.95 - 217872336</td>
</tr>
</tbody>
</table>

4.2.2.2.1 EMS configuration detail

The Java client programs tibjmsMsgProducerPerf and tibjmsMsgConsumerPerf (included as sample programs in the TIBCO EMS release) were used as client applications. Producers sent messages using PERSISTENT delivery mode and consumers used AUTO_ACKNOWLEDGE acknowledgement mode. The same number of consumers and producers were used so that messages would not accumulate at the EMS server, which could affect the performance.

The consumer was started with the following command:

```
java tibjmsMsgConsumerPerf –server <serverURL> -queue queue.<fs/nfs> –count <message count> -threads <number of consumers> -connections <number of connections>
```

The producer was started with the following command:

```
java tibjmsMsgProducerPerf –server <serverURL> -queue queue.<fs/nfs> –delivery PERSISTENT –count <message count> –size <message size> -threads <number of producers> -connections <number of connections>
```

Tibemsd.conf was modified to contain the following settings:

```
store_minimum_sync = 512MB
store_truncate = enabled
store_crc = disabled
max_msg_memory = 1GB
msg_swapping = enabled
```

The destination used in the tests is either failsafe or non-failsafe. Queues.conf was modified to contain the following settings for the failsafe and non-failsafe queues respectively:

```
queue.fs  failsafe
queue.nfs
```

Consumers used the default prefetch value for queues.

4.2.2.2.2 Test 1: Varying number of producers/consumers

In this test the message size was (1Kbytes). The number of producers and consumers varied. The number of producers/consumers corresponds to the values of the “-threads” and “-connections” parameters.

- Message Size = 1024 Bytes
- Message Count = 10000
- # Producers/Consumers = 1, 10, 20
Table 4. Producer/Consumer Rate (Msg/Sec)

<table>
<thead>
<tr>
<th>Failsafe</th>
<th>1</th>
<th>10</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>82 / 82</td>
<td>718 / 720</td>
<td>714 / 717</td>
</tr>
<tr>
<td>N</td>
<td>2707 / 2643</td>
<td>4046 / 3957</td>
<td>3644 / 3739</td>
</tr>
</tbody>
</table>

- Message Size = 1024 Bytes
- Message Count = 100000
- # Producers/Consumers = 1, 10, 20

Table 5. Producer/Consumer Rate (Msg/Sec)

<table>
<thead>
<tr>
<th>Failsafe</th>
<th>1</th>
<th>10</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>83 / 83</td>
<td>749 / 749</td>
<td>1243 / 1243</td>
</tr>
<tr>
<td>N</td>
<td>2684 / 2320</td>
<td>4072 / 4073</td>
<td>4321 / 4322</td>
</tr>
</tbody>
</table>

4.2.2.2.3 Test 2: Varying message size

In this test the number of producers and consumers was 1 each, but the message size varied. The message size corresponds to the value of “-size” parameter.
- Message Size = 100, 1K, 10K, 100K, 1M
- Number of Producer/Consumer = 1
- Message Count = 10000

Table 6. Producer/Consumer Rate (Msg/Sec)

<table>
<thead>
<tr>
<th>Failsafe</th>
<th>100 bytes</th>
<th>1K</th>
<th>10K</th>
<th>100K</th>
<th>1M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>82 / 82</td>
<td>82 / 82</td>
<td>76 / 76</td>
<td>47 / 47</td>
<td>15 / 15</td>
</tr>
<tr>
<td>N</td>
<td>3466/3466</td>
<td>2511/2289</td>
<td>1661/1562</td>
<td>395/395</td>
<td>38 / 38</td>
</tr>
</tbody>
</table>

4.3 Two EMS servers on separate systems using CIFS and NAS

The Common Internet File System (CIFS) is a protocol for filesystem operations between computers on a network. Originally developed during the 1980s by IBM as the Server Message Block protocol (SMB), it has been enhanced and expanded by Microsoft and others. The HP CIFS Client enables HP-UX users to mount as UNIX file systems directories shared from CIFS file servers such as Windows servers – including those implemented on specialized Network Attached Storage (NAS) devices – and HP-UX machines running HP CIFS Server.

4.3.1 Configuration overview

An HP StorageWorks NAS 1200 was used as a shared storage device for the EMS configuration files (Figure 4). A single copy of the files is placed on an exported filesystem (/ems-nas-share) on this device. Both systems mount the filesystem as /ems_nas. As with the single system configuration the primary server locks this shared state upon initialization and the secondary can only gain access to the data upon failure of the primary.
The Remote Desktop Protocol client was used to access the NAS device running Windows Storage Server. The following command was issued from one of the HP-UX servers:

```
rdesktop -g 1024x800 -u administrator ems-nas -X ; where ems-nas is the name of the device
```

Once logged into the Storage Server the HP StorageWorks NAS Management Console was launched. This web-based GUI was used to create and assign attributes to a folder (ems-nas-share) and export it for use by clients.

In order to mount the shared folder from an HP-UX host the CIFS product must be installed. For this testing, the product filesets were installed on rpnode1 and rpnode2 as part of the HP-UX 11i Mission Critical Operating Environment:

- **HPUX11i-OE-MC.CIFS-Server** A.01.10  HP CIFS Server (Samba) File and Print Services
- **HPUX11i-OE-MC.CIFS-Development** A.01.10  HP CIFS Server Source Code Files
- **HPUX11i-OE-MC.CIFS-Client** A.01.09.01  CIFS Client

The following commands were used to start the CIFS client, mount the share, login (only needed if not logged on as root), unmount and stop the CIFS client:

```
$ cifsclient start
$ cifsmount //ems-nas/ems-nas-share /ems_nas -U Administrator
$ cifsbin ems-nas Administrator cifsumount /ems_nas
$ cifsclient stop
```

After the cifsmount command is issued the HP-UX host will show the share mounted as:

```
localhost:\EMS-NAS\EMS-NAS-SHARE
707693552 71584 707621968 0% /ems_nas
```
4.3.2 Tests and results
The HP CIFS Client is not capable of forwarding lock requests to CIFS servers (in this case Windows Storage Server on the NAS device). The HP CIFS Server is capable of supporting "enforced file locking" from the Windows Client, but since the HP CIFS Client cannot forward the lock request, it will not work from an HP-UX client. EMS is therefore not supported in this configuration.

4.4 One EMS server running on a Serviceguard cluster
Serviceguard enables the creation of high availability clusters of HP servers. High availability clusters ensure that application services remain available in the event of software or hardware failures, such as an operating system panic or the loss of a system processing unit (SPU), disk, or local area network (LAN) component. If one component in the cluster fails, a redundant component takes over. Serviceguard and other high availability subsystems coordinate the transfer between these components to ensure minimal disruption of service. This concept extends to entire systems. In a Serviceguard cluster, application services (individual HP-UX processes) are grouped together in packages that run on a given system in the cluster. If a failure occurs on the system such that the package can no longer run on it, Serviceguard can automatically transfer control of the package to another system within the cluster. Data associated with the package, resident on storage shared amongst the cluster systems, follows the migration of the package.

4.4.1 Configuration overview
Serviceguard version 11.16 was installed on both nodes as part of the Mission Critical Operating Environment (Figure 5). An HP StorageWorks EVA3000 was used as the shared storage device for this configuration. The EMS application configuration files were located in an LVM volume group and VxFS filesystem (/ems) on this device. Serviceguard takes care of making this data available exclusively to the system currently running the package (ems_pkg). The Serviceguard package configuration file is shown in Appendix C.

Figure 5. EMS in a Serviceguard cluster
To ensure robust and reliable access to shared storage, HP StorageWorks Secure Path for HP-UX was installed. Secure Path is multi-path, high availability software that manages and maintains continuous data access to HP storage systems enabling no single point of failure from server to storage. Secure Path for HP-UX is host-resident software that monitors the data paths between server and storage to increase availability of information. In the event that a path failure is detected, Secure Path for HP-UX fails over to an alternative path. When the original path becomes available, Secure Path for HP-UX can automatically fallback to the original path. Secure Path for HP-UX can also balance the workload among available paths to optimize system performance.

4.4.2 Tests and results
EMS failover, performance, and startup tests were performed. A standard repertoire of tests and commands were also exercised on the cluster to verify basic functionality. These tests and commands are listed in Appendix E (section 8.5).

4.4.2.1 Test set 1: Failover tests
The goal of the test set was to verify that the Serviceguard software would transparently failover the EMS service to a functioning cluster node.

4.4.2.1.1 EMS configuration detail
In this test set the topic publisher sent messages in PERSISTENT delivery mode and the topic subscriber used AUTO_ACKNOWLEDGE acknowledgement mode. The topic subscriber started before the topic publisher.

The following command was used to start the subscriber:

```
java tibjmsMsgConsumerPerf -server <server1URL,server2URL> -topic topic.test -count 100000
```

The following command was used to start the publisher (using a message size of 1K):

```
java tibjmsMsgProducerPerf -server <server1URL,server2URL> -topic topic.test -delivery PERSISTENT -count 100000
```

The file tibemsd.conf contained the following settings:

```
store = <mounted directory on shared SAN partition>
```

To avoid data loss in the case of software/hardware failure, a failsafe topic destination was used for the testing.

The file topics.conf contained the following line for the testing destination topic.test:

```
topic.test failsafe
```

4.4.2.1.2 Test 1: Varying failure types
The primary server was failed while the topic publisher and subscriber were running. The following outcomes were then verified:

1. The cluster server software does monitor and perform failover in case of failure.
2. The server failover is transparent to the publishing and subscribing applications and they complete after the failover

Specifically, the following tests were performed:

- EMS Server Process failure, by issuing a kill –9 against the EMS server process
- Hardware failure, by performing a hard shutdown of the machine on which the EMS server is running.
with the following results:

<table>
<thead>
<tr>
<th>Failure Type</th>
<th>EMS Server Failure</th>
<th>Hardware Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failover Time</td>
<td>0</td>
<td>67 sec</td>
</tr>
</tbody>
</table>

4.4.2.2 Test set 2: Performance tests

The performance tests compare performance when using failsafe and non failsafe destinations:

- Failsafe destination: synchronous I/O is used in EMS
- Non-Failsafe destination: asynchronous I/O is used in EMS

This test set ran with clients connecting to a single server without fault tolerant settings.

4.4.2.2.1 EMS configuration detail

The Java client programs, tibjmsMsgProducerPerf and tibjmsMsgConsumerPerf, were used as client applications to connect to the EMS server. The producers sent messages in PERSISTENT delivery mode and consumers used AUTO_ACKNOWLEDGE acknowledgement mode. The same number of consumers and producers were used, so that messages would not accumulate at the EMS server, possibly affecting the performance.

The following command was used to start the consumer:

```
java tibjmsMsgConsumerPerf --server <serverURL> -queue queue.<fs/nfs> -count <message count> -threads <number of producers> -connections <number of connections>
```

The following command was used to start the producer:

```
java tibjmsMsgProducerPerf --server <serverURL> -queue queue.<fs/nfs> --delivery PERSISTENT -count <message count> -size <message size> -threads <number of consumers> -connections <number of connections>
```

The following values were set in tibemsd.conf:

- `store_minimum_sync = 512MB`
- `store_truncate = disabled`
- `store_crc = disabled`
- `max_msg_memory = 1GB`
- `msg_swapping = enabled`

The destination used in the tests was either failsafe or non-failsafe. Queues.conf contained the following lines for the failsafe queue, `queue.fs`, and non-failsafe queue, `queue.nfs`:

```
queue.fs  failsafe
queue.nfs
```

The consumers used the default prefetch value.

4.4.2.2.2 Test 1: Varying number of producers/consumers with 1K message

In the first scenario, the same message size was used but the number of producers and consumers was varied. The number of producer/receivers corresponds to the value of “-threads” and “-connections” parameters.

- Message Size = 1024 Bytes
- Message Count = 100000
• # Producers/Consumers = 1, 10, 20, 50

Table 7. Producer/Consumer Rate (Msg/Sec)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failsafe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>1354/1352</td>
<td>6433/6426</td>
<td>7532/7523</td>
<td>8182/8122</td>
</tr>
<tr>
<td>N</td>
<td>4685/3735</td>
<td>6651/6652</td>
<td>7853/6945</td>
<td>7063/7063</td>
</tr>
</tbody>
</table>

Table 8. Disk I/O Read/Write Rate (Bytes/Sec)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Failsafe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>0/1.2M</td>
<td>0/6.6M</td>
<td>0/7.5M</td>
<td>0/8.0M</td>
</tr>
<tr>
<td>N</td>
<td>173k/753k</td>
<td>0/1.3M</td>
<td>0/1.3M</td>
<td>0/1.3M</td>
</tr>
</tbody>
</table>

4.4.2.2.3 Test 2: Varying message size
• Message Size = 100, 1K, 10K, 100K, 1M
• Number of Producer/Consumer = 1
• Message Count = 100000

Table 9. Producer/Consumer Rate (Msg/Sec)

<table>
<thead>
<tr>
<th></th>
<th>100 bytes</th>
<th>1K</th>
<th>10K</th>
<th>100K</th>
<th>1M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failsafe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>1350/1349</td>
<td>1354/1352</td>
<td>1145/1145</td>
<td>275/275</td>
<td>34/34</td>
</tr>
<tr>
<td>N</td>
<td>3938/3503</td>
<td>4685/3735</td>
<td>2564/2565</td>
<td>506/506</td>
<td>39/39</td>
</tr>
</tbody>
</table>

Table 10. Disk I/O Read/Write Rate (Bytes/Sec)

<table>
<thead>
<tr>
<th></th>
<th>100 bytes</th>
<th>1K</th>
<th>10K</th>
<th>100K</th>
<th>1M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failsafe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>0/1.3M</td>
<td>0/1.2M</td>
<td>0/12.3M</td>
<td>0/26.2M</td>
<td>0/34.2M</td>
</tr>
<tr>
<td>N</td>
<td>508k/1.1M</td>
<td>173k/753k</td>
<td>0/25.6M</td>
<td>0/46.2M</td>
<td>0/39.2M</td>
</tr>
</tbody>
</table>

4.4.2.3 Test set 3: EMS startup tests

The goal of this test was to measure the time it takes for the server, after failover, to restore the state and become active. Storage size and message size were varied in the tests.

Startup time was measured as the interval between the time the server starts the recovery process and the time it becomes active after restoring all state from the shared storage.

4.4.2.3.1 EMS configuration detail

A persistent queue producer was used to fill the EMS server storage before the server was failed over. The message count and message size parameter were varied.

The following command was used to start the producer:

```
java tibjmsMsgProducerPerf -server <serverURL> -queue queue.test.<fs/nfs> -delivery PERSISTENT -count <message count> -size <message size> -threads 50 -connections 50
```
The following parameter settings were made in the tibemsd.conf configuration file:

- store = <data store location>
- store_truncate = enabled
- store_crc = disabled
- msg_swapping = enabled

Note that max_msg_memory and store_minimum_sync were not set.

Tests were performed with failsafe and non-failsafe queues specified in the destination configuration file, with the names queue.fs and queue.nfs.

### 4.4.2.3.2 Test 1: Varying storage size

**Message Size = 1024 Bytes**

**Message Count = 0, 10,000 (storage size ~ 10MB), 1,000,000 (storage 1.4GB), 5,000,000 (storage ~ 7.2GB)**

<table>
<thead>
<tr>
<th>Failsafe</th>
<th>0</th>
<th>10,000</th>
<th>1,000,000</th>
<th>5,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>139</td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td></td>
</tr>
</tbody>
</table>

### 4.4.2.3.3 Test 2: Varying size of individual message in storage

**Message Size = 400 Bytes**

**Size of Storage = 0, 1,000,000 (storage ~500MB), 2,500,000 (storage ~ 1GB)**

<table>
<thead>
<tr>
<th>Failsafe</th>
<th>0</th>
<th>1,000,000</th>
<th>2,500,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>0</td>
<td>23</td>
<td>66</td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>0</td>
<td>24</td>
</tr>
</tbody>
</table>

**Message Size = 10k Bytes**

**Size of Storage = 0, 100,000 (storage 1.4GB), 500,000 (storage 7.2GB)**

<table>
<thead>
<tr>
<th>Failsafe</th>
<th>0</th>
<th>100,000</th>
<th>500,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>0</td>
<td>278</td>
<td>372</td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.4.2.4 Test set 3: The effect of stored on partition failover time

The goal of this test was to measure the failover time when using clustering environments where extra tasks such as fsck come into play during failover.

Storage size was varied across the tests. Serviceguard commands were used to initiate the failover.
4.4.2.4.1 EMS configuration detail

A persistent queue producer was used to fill the EMS server to varying storage size before server failover. The “message count” parameter varied to create the storage.

The following command was used to start the producer:

```
java tibjmsMsgProducerPerf -server <serverURL> -queue queue.test -delivery PERSISTENT -
count <message count> -size <message size> -threads 20 -connections 20
```

The following parameters were set in tibemspd.conf:

- store_truncate = enabled
- store_crc = disabled
- msg_swapping = enabled

Tests were performed with failsafe and non-failsafe queues with the name queue.test specified in the destination configuration file.

4.4.2.4.2 Test 1: Varying storage size

Message Size = 1024 Bytes

Size of Storage = 1G, 5G

Table 12. Partition Failover Times (Sec)

<table>
<thead>
<tr>
<th></th>
<th>Failsafe</th>
<th>1G</th>
<th>5G</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAN</td>
<td>Y</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>SAN</td>
<td>N</td>
<td>16</td>
<td>17</td>
</tr>
</tbody>
</table>

4.5 One EMS server running on a VERITAS Cluster Server cluster

Similar to Serviceguard, VERITAS Cluster Server (VCS) is a high-availability solution for cluster configurations. A single VCS cluster consists of multiple systems connected in various combinations to shared storage devices. Shared storage provides multiple systems an access path to the same data. VCS monitors the cluster systems and the application services running on them (represented as ‘service groups’) and restarts services on a different system when hardware or software fails. Client applications continue operation with little or no downtime.

4.5.1 Configuration overview

VERITAS SANPoint Foundation Suite/HA was installed on both nodes (Figure 6). An HP FC10 Fibre Channel disk enclosure was used as the shared storage device for this configuration. The EMS application configuration files were located in a VxVM private disk group and VxFS filesystem (/ems) on this device. VCS takes care of making this data available sequentially and exclusively to the system currently running the VCS EMS service group. The EMS service group was configured using the bundled agents provided with VCS. The cluster configuration file (main.cf) is shown in Appendix F. The tool ‘hagui’ was used to create the EMS service group, the resources, the linkage between resources, and the dependency hierarchy.
4.5.2 Tests and results
EMS failover and performance tests were performed.

4.5.2.1 Test set 1: Failover tests
The goal of the test set was to verify that the VERITAS Cluster Server software would transparently failover the EMS service to a functioning cluster node.

4.5.2.1.1 EMS configuration detail
In this test set the topic publisher sent messages in PERSISTENT delivery mode and the topic subscriber used AUTO_ACKNOWLEDGE acknowledgement mode. The topic subscriber was started before the publisher.

The following command was used to start the subscriber:
```
java –Dtibco.tibjms.reconnect.attempts="100,1000" tibjmsMsgConsumerPerf –server <clusterURL,clusterURL> -topic topic.test –count 10000
```

The following command was used to start the publisher (using a message size of 1K):
```
java –Dtibco.tibjms.reconnect.attempts="100,1000" tibjmsMsgProducerPerf –server <clusterURL,clusterURL> -topic topic.test –delivery PERSISTENT –count 10000
```

The property –Dtibco.tibjms.reconnect.attempts="100,1000" tells the EMS client API to attempt to reconnect 100 times at a 1 second interval when the connection to the EMS server is lost. This setting was used because the VERITAS Cluster Server was using a poll frequency of 60 seconds to detect that a cluster machine was down. The EMS client applications need to attempt reconnect long enough to let the cluster software start the backup EMS server during failover. In a production environment the client application may need to try for 30 minutes if the EMS server has a very large store file to recover. When the reconnect.attempts is too short, the EMS client applications receive the following error during EMS failover:
javax.jms.JMSException: Connection has been terminated by the server

The tibemsd.conf contained the following settings:

store = <mounted directory on shared SAN partition>

To avoid data loss in the case of software/hardware failure, a failsafe topic destination was used for the testing.

The following line was specified in the topics.conf for the testing destination topic.test:

`topic.test failsafe`

4.5.2.1.2 Test 1: Varying failure types

The primary server was failed while the topic publisher and subscriber were running. The following outcomes were then verified:

1. The cluster server software does monitor and perform failover in case of failure.
2. The server failover is transparent to the publishing and subscribing applications and they complete after the failover.

Specifically, the following tests were performed:

- EMS Server Process failure by issuing a kill -9 against the EMS server process
- Hardware failure, by rebooting the machine on which the EMS server is running (/etc/reboot).

With the following results:

<table>
<thead>
<tr>
<th>Failure Type</th>
<th>EMS Server Failure</th>
<th>Hardware Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failover Time</td>
<td>(75 seconds) 60 seconds to detect failure and 15 seconds to start secondary.</td>
<td>(75 seconds) 60 seconds to detect failure and 15 seconds to start secondary.</td>
</tr>
</tbody>
</table>

It is also possible to configure the VERITAS Cluster Server software to restart the primary EMS server on software failure. This configuration also works.

The failover tests PASS.

4.5.2.2 Test Set 2: Performance tests

The goal of this test set was to measure throughput of the EMS system. The tests measured the message rate of producers and consumers when varying message size, number of threads, and failsafe versus non-failsafe destinations. A failsafe destination uses synchronous I/O in EMS. A non-failsafe destination uses asynchronous I/O in EMS.

Disk performance is usually the primary factor affecting EMS server throughput (especially when using failsafe destinations). The performance impact (if any) of using the VERITAS Cluster software was not measured. The “diskperf” tool was used to measure disk performance. The diskperf tool uses one thread to write to disk.

$ ./diskperf -block <blocksize> -count 10000 <-fsync> -file /ems/diskperf.test -keep –prealloc

Table 13. HP FC10 Fibre Channel Disk I/O Write Rate (Writes/Sec – Bytes/Sec)

<table>
<thead>
<tr>
<th></th>
<th>1K</th>
<th>10K</th>
<th>102 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write (fsync)</td>
<td>107.56 - 110143</td>
<td>94.57 - 968395</td>
<td>114.17 - 11646</td>
</tr>
<tr>
<td>Write (no fsync)</td>
<td>162.88 - 166786</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
$ ./diskperf -block 1024 -count 10000 -file /ems/diskperf.test -keep –read

**Table 14.** HP FC10 Fibre Channel Disk I/O Read Rate (Read/Sec – Bytes/Sec)

<table>
<thead>
<tr>
<th>1K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
</tr>
<tr>
<td>11641.44 - 11920838</td>
</tr>
</tbody>
</table>

The disk used for these EMS tests was somewhat slow. For comparison, the disk performance numbers for similar tests using HP StorageWorks EVA3000 are listed below.

**Table 15.** HP StorageWorks EVA3000 Disk I/O Rate (IO/Sec – Bytes/Sec)

<table>
<thead>
<tr>
<th>1K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write (fsync)</td>
</tr>
<tr>
<td>913.66 - 935587</td>
</tr>
<tr>
<td>Write (no fsync)</td>
</tr>
<tr>
<td>3941.66 - 4036263</td>
</tr>
<tr>
<td>read</td>
</tr>
<tr>
<td>212765.95 - 217872336</td>
</tr>
</tbody>
</table>

4.5.2.2.1 EMS configuration detail

The Java client programs tibjmsMsgProducerPerf and tibjmsMsgConsumerPerf (included as sample programs in the TIBCO EMS release) were used as client applications. Producers sent messages using PERSISTENT delivery mode and consumers used AUTO_ACKNOWLEDGE acknowledgement mode. The same number of consumers and producers were used so that messages would not accumulate at the EMS server, possibly affecting the performance.

The consumer was started with the following command:

```
java tibjmsMsgConsumerPerf --server <serverURL> -queue queue.<fs/nfs> -count <message count> -threads <number of consumers> -connections <number of connections>
```

The producer was started with the following command:

```
java tibjmsMsgProducerPerf --server <serverURL> -queue queue.<fs/nfs> --delivery PERSISTENT -count <message count> --size <message size> -threads <number of producers> -connections <number of connections>
```

Tibemsd.conf was modified to contain the following settings:

- `store_minimum_sync = 512MB`
- `store_truncate = enabled`
- `store_crc = disabled`
- `max_msg_memory = 1GB`
- `msg_swapping = enabled`

The destination used in the tests was either failsafe or non-failsafe. Queues.conf was modified to contain the following settings for the failsafe and non-failsafe queues respectively:

```
queue.fs  failsafe
queue.nfs
```

The consumers used the default prefetch value.
4.5.2.2 Test 1: Varying number of producers/consumers

In this test the message size was (1Kbytes). The number of producers and consumers varied. The number of producers/consumers corresponds to the values of the “-threads” and “-connections” parameters.

- Message Size = 1024 Bytes
- Message Count = 10000
- # Producers/Consumers = 1, 10, 20

<table>
<thead>
<tr>
<th>Table 16. Producer/Consumer Rate (Msg/Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failsafe</td>
</tr>
<tr>
<td>Y</td>
</tr>
<tr>
<td>N</td>
</tr>
</tbody>
</table>

4.5.2.2.3 Test 2: Varying message size

In this test the number of producers and consumers was 1 each, but the message size varied. The message size corresponds to the value of “-size” parameter.

- Message Size = 100, 1K, 10K, 100K, 1M
- Number of Producer/Consumer = 1
- Message Count = 10000

<table>
<thead>
<tr>
<th>Table 17. Producer/Consumer Rate (Msg/Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failsafe</td>
</tr>
<tr>
<td>Y</td>
</tr>
<tr>
<td>N</td>
</tr>
</tbody>
</table>

5 Summary

The tests in this document show that the following configurations may be used to provide fault tolerant EMS servers on the HP-UX platform.

- A failover server configuration with two EMS servers on the same system (providing fault tolerance in case of EMS software failure only).
- A failover server configuration with each EMS server on its own system – with Network Attached Storage and VERITAS Cluster File System.
- A failover EMS server configuration using HP Serviceguard to start EMS
- A failover EMS server configuration using VERITAS Cluster Server to start EMS

The following configuration may NOT be used to provide fault tolerant EMS servers on the HP-UX platform.

- A failover server configuration using Network Attached Storage and HP-UX CIFS client software
6 Resources

6.1 Reference materials
TIBCO Enterprise Message Service User’s Guide
TIBCO Enterprise Message Service Installation
TIBCO Enterprise Message Service Release Notes
VERITAS SANPoint Foundation Suite (tm) and SANPoint Foundation Suite HA 3.5 Release Notes for HP-UX 11i
VERITAS SANPoint Foundation Suite (tm) 3.5 Installation and Configuration Guide for HP-UX 11i
VERITAS Cluster Server 3.5 Release Notes for HP-UX
VERITAS Cluster Server 3.5 Agent Developer’s Guide for HP-UX
VERITAS Cluster Server 3.5 Bundled Agents Reference Guide for HP-UX
VERITAS Cluster Server 3.5 Installation and Configuration Guide for HP-UX
VERITAS Cluster Server 3.5 User’s Guide for HP-UX

Documentation available at:
TIBCO (whitepapers available) http://www.tibco.com
TIBCO User Technical Portal http://power.tibco.com/
TIBCO Download Site (requires ID) http://download.tibco.com/
VERITAS Documentation http://www.symantec.com/enterprise/support/index.jsp
HP Documentation http://www.docs.hp.com
HP Developer Portal http://www.hp.com/go/dspp
HP IT Resource Center http://itrc.hp.com/

7 Revision history
Originally published July 2005
# Initializing...
# Contacting target "rpnode1"...
# Target: rpnode1:/

# Bundle(s):

B6848BA 1.4.gm.46.7  Ximian GNOME 1.4 GTK+ Libraries for HP-UX
B8111AA 1.2.2.15.00  Java 2 RTE for HP-UX (700/800), PA1.1 + PA2.0
Add On
B8465BA A.01.05.08  HP WBEM Services for HP-UX
B9073BA B.06.02  HP-UX icod (Instant Capacity on Demand)
B9098AA 1.2.2.15.00  Java 2 Plugin for HP-UX (700/800)
B9788AA 1.3.1.13.01  Java2 1.3 SDK for HP-UX
B9789AA 1.3.1.13.01  Java2 1.3 RTE for HP-UX
BUNDLE11  B.11.11.0306.1  Required Patch Bundle for HP-UX 11i, June 2003
Base-VXVM  B.03.50.5  Base VERITAS Volume Manager Bundle 3.5 for HP-UX

BaseVxFS  B.03.50.0  VERITAS File System Bundle for HP-UX
CDE-English  B.11.11  English CDE Environment
DevIds11i  B.11.11  HP-UX Device IDs Enablement
ENHAUTO  B.11.11.0402.1  Enhanced AutoFS, February 2004
FDDI-00  B.11.11.02  PCI FDDI; Supptd HW=A3739A/A3739B; SW=J3626AA
FEATURE11-11  B.11.11.0209.5  Feature Enablement Patches for HP-UX 11i, Sept 2002
FibrChanl-00  B.11.11.09  PCI/HSC FibreChannel; Supptd
HW=A6684A,B6685A,A5158A,A6795A
FibrChanl-01  B.11.11.03  PCI-X FibreChannel; Supptd
HW=A6826A,A9782A,A9784A
GOLDFPLS11i  B.11.11.0406.5  Gold Applications Patches for HP-UX 11i v1, June 2004
GOLDBASE11i  B.11.11.0406.5  Gold Base Patches for HP-UX 11i v1, June 2004
GigEther-00  B.11.11.19  PCI/HSC GigEther; Supptd
HW=A4926A/A4929A/A4924A/A4925A; SW=J1642AA
GigEther-01  B.11.11.15  PCI/X GigEther; Supptd
HW=A4926A/A4929A/A4924A/A4925A; SW=J1642AA
HPUX11i-0E-MC  B.11.11.0406  HP-UX Mission Critical Operating Environment Component
HPUXBase4  B.11.11  HP-UX 64-bit Base OS
HPUXBaseAux  B.11.11.0406  HP-UX Base OS Auxiliary
HWEnable11i  B.11.11.0406.4  Hardware Enablement Patches for HP-UX 11i v1, June 2004
IEther-00  B.11.11.05  PCI/PCI-X IEther; Supptd HW=A7011A/A7012A/A37352A
J4258CA  B.06.11  Netscape Directory Server v6 for HP-UX
MOZILLA 1.4.0.01.04  Mozilla 1.4 for HP-UX
MOZILLAsrc 1.4.0.01.04  Mozilla 1.4 Source distribution
OnlineDiag  B.11.11.14.15  HPUX 11.11 Support Tools Bundle, Jun 2004
OpenSSL  A.0.0.0.07-d.002  Secure Network Communications Protocol
PROCSETS  A.0.0.0.06  HP-UX Processor Sets Bundle
RAID-00  B.11.11.01  PCI RAID; Supptd HW=A5856A
T1455AA 1.3.1.13.01  Java2 1.3 Netscape Plugin for HP-UX
T1456AA 1.4.2.02.01  Java2 1.4 SDK for HP-UX
T1456AAddOn 1.4.2.02.01  Java2 1.4 SDK -AA addon for HP-UX
T1457AA 1.4.2.02.01  Java2 1.4 RTE for HP-UX
T1457AAddOn 1.4.2.02.01  Java2 1.4 RTE -AA addon for HP-UX
T1458AA 1.4.2.02.01  Java2 1.4 Netscape Plugin for HP-UX
T1471AA  A.03.61.002  HP-UX Secure Shell
hpuwsApache  A.2.0.49.00  HP-UX Apache-based Web Server
hpuwsTomcat  A.4.1.29.02  HP-UX Tomcat-based Servlet Engine
hpuwsWebmin  A.1.070.00  HP-UX Webmin-based Admin
hpuwsXmL  A.2.00  HP-UX XML Web Server Tools
perl  D.5.0.0.B  Perl Programming Language
scsiU320-00  B.11.11.01  PCI SCSI U320; Supptd HW=A7173A

# Product(s) not contained in a Bundle:
8.2 Appendix B. Swlist of rxnode

# Initializing...
# Contacting target "rxnode1"...
# Target: rxnode1:/

B6848BA 1.4.gm.46.9  Ximian GNOME 1.4 GTK+ Libraries for HP-UX
B6849AA B.02.01.02  Bastille Security Hardening Tool
BB339BA B.03.00.09  servicecontrol manager Server and Agent Bundle
BB465BA A.02.00.04  HP WEM Services for HP-UX
B9073BA B.11.23.06.03  HP-UX iCOD (Instant Capacity on Demand)
B9788AA 1.3.1.13.01  Java2 1.3 SDK for HP-UX
B9789AA 1.3.1.13.01  Java2 1.3 RTE for HP-UX
B9901AA A.03.05.10.02  HP IPFilter 3.5alpha5
BUNDLE11i B.11.23.0409.3 Required Patch Bundle for HP-UX 11i v2 (B.11.23),
September 2004
Base-VXVM B.03.50.1A.014  Base VERITAS Volume Manager Bundle 3.5 for HP-UX
CDE-English B.11.23.0409  English CDE Environment
FDDI-00 B.11.23.01  PCI FDDI; Supptd HW=A3739B; SW=J3626AA
FibrChanl-00 B.11.23.03  PCI FibreChannel; Supptd HW=A6795A, A5158A
FibrChanl-01 B.11.23.02  PCI-X FibreChannel; Supptd
HPUX11i-OE-MC B.11.23.0409  HP-UX Mission Critical Operating Environment Component
HPUXBaseAux B.11.23.0409  HP-UX Base OS Auxiliary
HPXBaseOS B.11.23  HP-UX Base OS
IEther-00 B.11.23.05  PCI/PCI-X iEther; Supptd
J4258CA B.06.04.01  ISEE Platform
Judy B.11.04.04.15  Judy Library - development and runtime
libraries for handling dynamic arrays
MOZILLA 1.4.0.01.04  Mozilla 1.4 for HP-UX
MOZILLAsrc 1.4.0.01.04  Mozilla 1.4 Source distribution
MySQL 3.23.54a.01  MySQL open-source database
NPar B.11.23.01.03.00.06  nPartition Provider for HP-UX
OnlineDiag B.11.23.03.16  HPUX 11.23 Support Tools Bundle, Sep 2004
OpensSL A.00.09.07.d.011  Secure Network Communications Protocol
ParMgy B.11.23.02.00.03.04  Partition Manager - HP-UX
RAID-01 B.11.23.02  RAID SA; Supptd HW=A7143A/A9890A/A9891A
Sec00Tools B.01.02.00  Install-Time security infrastructure.
SecPatchCk B.02.02  HP-UX Security Patch Check Tool
T1456AA A.14.2.03.04  Java2 1.4 SDK for HP-UX
T1456AAaddon 1.4.2.03.04  Java2 1.4 SDK - AA add-on for HP-UX
T1457AA 1.4.2.03.04  Java2 1.4 RTE for HP-UX
T1457AAdaddon 1.4.2.03.04  Java2 1.4 RTE - AA add-on for HP-UX
T1471AA A.03.71.007  HP-UX Secure Shell
USB-00 B.11.23.02  Object Oriented USB Driver
WBEPM-LAN-00 B.11.23.01  LAN Provider for Ethernet LAN interfaces.
8.3 Appendix C. Serviceguard package configuration files

diff of “template.conf” and “ems.conf”

```plaintext
< PACKAGE_NAME
> # PACKAGE_NAME
> PACKAGE_NAME ems_pkg
< NODE_NAME
> # NODE_NAME
> NODE_NAME rxnode1
> NODE_NAME rxnode2
< LOCAL_LAN_FAILOVER_ALLOWED YES
> LOCAL_LAN_FAILOVER_ALLOWED NO
< RUN_SCRIPT
> RUN_SCRIPT /etc/cmcluster/ems_pkg/ems.cntl
< HALT_SCRIPT
> HALT_SCRIPT /etc/cmcluster/ems_pkg/ems.cntl
< #SERVICE_NAME <service name>
< #SERVICE_FAIL_FAST_ENABLED <YES/NO>
< #SERVICE_HALT_TIMEOUT <number of seconds>
> SERVICE_NAME ems
> SERVICE_FAIL_FAST_ENABLED NO
> SERVICE_HALT_TIMEOUT 10
< #SUBNET
> SUBNET 10.255.177.0
```

diff of “template.cntl” and “ems.cntl”

```plaintext
> VG[0]="/dev/vgems"
> LV[0]="/dev/vgems/lvems"; FS[0]="/ems"; FS_MOUNT_OPT[0]="-o rw,suid,largefiles,delayslog,datalnlog"; FS_UMount_OPT[0]=""; FS_FSCK_OPT[0]=""
> FS_TYPE[0]="vxfs"
< #IP[0]=""
< #SUBNET[0]=""
> IP[0]="10.255.177.15"
> SUBNET[0]="10.255.177.0"
> #IP[0]="10.255.177.11"
> #SUBNET[0]="10.255.177.0"
> #IP[1]="10.255.178.11"
> #SUBNET[1]="10.255.178.0"
< #SERVICE_NAME[0]=""
< #SERVICE_CMD[0]=""
> SERVICE_NAME[0]="ems"
> SERVICE_CMD[0]="/opt/tibco/ems/bin/tibemsd64 -config /ems/sg/tibemsd.conf"
> SERVICE_RESTART[0]="-R"
```

8.4 Appendix D. Serviceguard configuration steps

**Notes:**
- The shared volume group and lock disk must be on the shared storage device

**I. Prepare the systems**

A. Install software
1. Install HP-UX on all systems before cabling disks together
   - Including all necessary patches (see the Release Notes and the ITRC, http://itrc.hp.com for current patches)
   - Mirror boot disk (root logical volumes)
   - Synchronize UNIX config files across all nodes (using, for instance, NIS)
   - Synchronize time across all nodes using NTP
   - Synchronize kernel parameters across all nodes
   - Configure name resolution on all nodes (ensure hacl entries are defined in /etc/inetd.conf and /etc/services on the NIS server if NIS is configured)

2. Install Serviceguard software
   - Verify installation in Software Distributor logs.

B. Ensure all nodes can communicate by configuring /rhosts on all hosts.
   - chmod 400 for security
   - contains all node names (unqualified)
   - alternatively, configure /cmclnodelist (See Manual)

C. Configure shared storage using LVM

Notes:
   - make all owner/group and permissions same on nodes
   - on shared disk device (EVA)
   - create dual paths (SecurePath)
   - use vgdisplay and lvdisplay where appropriate to ensure accuracy of operations
   - VG names and numbers must be unique within the cluster

1. Create shared volume group (/dev/vgems), logical volume (/dev/vgems/lvems), and mount point (/ems) on rxnode1
2. Deactivate shared volume group on rxnode1
   - umount /ems # for all mounted shared LVs (eg., /<appname>)
   - Modify /etc/fstab: remove (comment) references to shared LVs
   - vgchange -a n /dev/vgems

3. Distribute the mapfile (on rxnode1)
   - vgexport -p -s -m <mapfile_name> /dev/vgems
     Record PV names
     copy the mapfile to rxnode2

4. Distribute the volume group (on rxnode2)
   - mkdir /dev/vgems
   - Create /dev/vgems/group with the same minor number
     - (mm) as on the primary system eg., mknode /dev/vgems/group c 64 0xmm0000
   - transpose disk device names if different
Note: the following procedure may be different for SecurePath since pvlinks would not be used

- `vgimport -s -m <mapfile_name> /dev/vgems <pv> <pv> <pv> ...
- `vgchange -a y /dev/vgems` # verify vgimport
- `vgcfgbackup /dev/vgems` # for all imported VG’s

Note: special handling of `/etc/lvmpvg` may be needed for certain types of storage configurations, see the Manual

- create mount points and mount directories
- umount all the directories
- `vgchange -a n /dev/vgems` # for all shared VGs

5. Repeat steps 3 & 4 for each primary/standby pair in the cluster (n/a for a two node cluster)

6. Create cluster lock volume group

   - `/dev/vglock`, on shared storage

D. Configure networking

1. Verify all network connections

2. Identify live & standby links (private LANs)

3. Verify network startup: `/etc/rc.config.d/netconf`

4. Verify IP address and names in name service lookup tables: `/etc/hosts`, DNS, NIS (/etc/nsswitch.conf)
   - service name (EMS) and “floating” IP address must be defined in name service

5. Optionally configure Auto Port Aggregation (trunking) on the links for load balancing and fault tolerance. see:  

II. Configure the cluster

A. Prepare cluster configuration file (cluster.ascii)

1. `vgchange -a y /dev/vgems`

2. `cd /etc/cmcluster` # home dir for config files

3. `cmquerycl -v -C cluster.ascii -n rxnode1 -n rxnode2`

4. edit `cluster.ascii` (see Manual).
5. cmcheckconf -v -C cluster.ascii
   • repair any reported problems and rerun 5

B. Initialize the cluster

1. vgchange -a y /dev/vglock       # initialize lock disk
2. cmapplyconf -v -C cluster.ascii  # distribute binary file
3. cmruncl                          # start the cluster
4. cmviewcl                         # verify start up
5. vgchange -a n /dev/vgems
6. vgchange -c y /dev/vgems
7. cmhaltcl                         # halt the cluster

C. Ensure the shared VGs are not activated at boot, but rather by the package

1. Modify /etc/lvmrc: AUTO_VG_ACTIVATE=0
2. Add non-shared VGs to custom_vg_activation function
3. Modify /etc/fstab: comment-out references to shared LVs  # VGs and filesystems will now have to be manually activated until packages are ready to go

D. Ensure nodes autostart the cluster on boot

1. On each node edit /etc/rc.config.d/cmcluster and set AUTOSTART_CMCLD=1

III. Configure packages

A. Create package configuration (.conf) and control scripts (.cntl) on one node

1. mkdir /etc/cmcluster/ems
2. cd /etc/cmcluster/ems
3. cmmakepkg -p ems.conf       # make template
4. cmmakepkg -s ems.cntl        # make template
5. Edit files using Appendix 2 worksheets and Manual
6. cmcheckconf -v -P ems/ems.conf # verify .conf
B. Distribute files to other nodes

1. cd /etc/cmcluster
2. rcp ems/ems.conf rxnode2:/etc/cmcluster/ems/ # copy .conf file to each cluster node
3. rcp ems/ems.cntl rxnode2:/etc/cmcluster/ems/ # copy cntl file to each cluster node
4. cmcheckconf -v -C cluster.ascii -P ems/ems.conf

Note: cmcheckconf and cmapplyconf must be rerun whenever cluster (cluster.ascii) or package configuration (.conf) files are changed

C. Generate the binary config file and distribute it to nodes

1. cmapplyconf -v -C cluster.ascii -P ems/ems.conf

D. Run the cluster and start packages

1. cmruncl -v # start the cluster, will start packages

8.5 Appendix E. Serviceguard test commands

1. Disconnecting one or more of the LAN links to simulate a card or cable failure
2. Disconnecting a power source such that backup power kicks in
3. Fail ‘soft’ system failure by doing a shutdown or reboot
4. Simulating a ‘hard’ system failure by turning the SPU off
5. Simulating one or more disk failures by pulling out hot swap disks
6. Simulate replacement of the failed disk by pushing it back in the hot swap enclosure
7. Killing a selected application process to simulate a software failure
8. Killing the Serviceguard cluster daemon
9. Testing the systems under load or with all packages running on one node
10. Displaying or modifying the cluster with command sequences such as the following:
   a. Start the cluster
      # cmruncl -v
   b. Stop the cluster
      # cmhaltcl -f -v ;the -f option will terminate the service if it is running
   c. Query the cluster
      # cmquerycl -v
   d. View the cluster
      # cmviewcl -v
e. Display networking information including the system affinity of the package ‘floating ip’
   # netstat -i

f. Display mounted filesystems, including those associated with running packages
   # bdf

g. move a package from one cluster node to another
   # cmhaltpkg pkg_name ; halt pkg on ___
   # cmrunpkg -n node_name pkg_name ; run it on new node

h. Enable package switching. (Use if the service has switched from one system to another and
   package switching is currently disabled)
   # cmmodpkg -d pkg_name ; disable first (check this)
   # cmmodpkg -e pkg_name

i. Halt a cluster node
   # cmhaltnode -f -v node_name ; -f is a force, which will halt the service, if it is running on
   this node (service will move to the other node)

j. Restart a node (cluster will reform with both nodes as members)
   # cmrunnode -v node_name

k. Restart the cluster when only one node is available (use, for instance, if the other node is down.
   Cluster will not form with a subset of nodes unless explicitly told to do so with the cmruncl –n
   command)
   # cmruncl -v -n node_name
   # cmmodpkg -e -n node_name pkg_name
   # cmrunpkg -n node_name pkg_name

l. Disable packet switching (until next cluster restart) for a running node.
   # cmmodpkg -d <pkg_name>

8.6 Appendix F. VCS main configuration file (main.cf)

include "types.cf"
include "CFSTypes.cf"
include "CVMTypes.cf"

cluster ems_cluster {
    UserNames = { admin = "cDRpdxPmHpzS." }
    Administrators = { admin }
    CounterInterval = 5
}

system rpnode1 {
}

system rpnode2 {
}

group cvm {
    SystemList = { rpnode1 = 0, rpnode2 = 1 }
    AutoFailOver = 0
    Parallel = 1
    AutoStartList = { rpnode1, rpnode2 }
}

CFSQlogckd qlogckd {
Critical = 0
)

CFSfsckd vxfsckd (
  ActivationMode @rpnode1 = { emsdg = sw }
  ActivationMode @rpnode2 = { emsdg = sw }
)

CVMCluster cvm_clus (
  Critical = 0
  CVMClustName = ems_cluster
  CVMNodeId = { rpnode1 = 0, rpnode2 = 1 }
  CVMTransport = gab
  CVMTimeout = 200
)

vxfsckd requires qlogckd

// resource dependency tree
//
// group cvm
// {
//  CVMCluster cvm_clus
//  CFSfsckd vxfsckd
//  {
//   CFSQlogckd qlogckd
//  }
//}
//

group ems (
  SystemList = { rpnode1 = 1, rpnode2 = 2 }
  AutoStartList = { rpnode1 }
)

DiskGroup emsDiskGroup (  
  DiskGroup = emsdg
)

IP emsIP (  
  Device = lan0
  Address = "10.255.177.15"
  NetMask = "255.255.255.0"
)

Mount emsMount (  
  MountPoint = "/ems"
  BlockDevice = "/dev/vx/dsk/emsdg/emsvol"
  FSType = vxfs
  FsckOpt = "-y"
)

NIC emsNIC (  
  Enabled = 0
  Device = lan0
)

Process emsProcess (  
  PathName = "/opt/tibco/ems/bin/tibemsd64"
  Arguments = "-config /ems/sg/tibemsd.conf"
)

Volume emsVolume (  
  Volume = emsvol
  DiskGroup = emsdg
)

emsIP requires emsNIC
emsMount requires emsVolume
emsProcess requires emsIP
emsProcess requires emsMount
emsVolume requires emsDiskGroup

// resource dependency tree
//
8.7 Appendix G. Power cycle test detail using the VERITAS SANPoint Foundation Suite

The primary node was power-cycled. The secondary took over within 20 seconds, but the secondary server purged the two client connections after 60 seconds (the default setting of the configuration parameter ft_reconnect_timeout). The clients did not notice the lost primary for about 10 minutes. When the clients tried to reconnect, the secondary (backup) server had purged the connections.

Log from the secondary EMS Server:

2005-03-25 14:44:41 Server is now active.
2005-03-25 14:45:42 Purged 2 connections.

2005-03-25 14:54:21 [anonymous@rxnode1]: reconnect failed: connection unknown
2005-03-25 14:54:24 [anonymous@rxnode1]: reconnect failed: connection unknown
2005-03-25 14:54:28 [anonymous@rxnode1]: reconnect failed: connection unknown
2005-03-25 14:54:32 [anonymous@rxnode1]: reconnect failed: connection unknown
2005-03-25 14:56:08 [anonymous@rxnode1]: reconnect failed: connection unknown
2005-03-25 14:56:11 [anonymous@rxnode1]: reconnect failed: connection unknown
2005-03-25 14:56:15 [anonymous@rxnode1]: reconnect failed: connection unknown
2005-03-25 14:56:18 [anonymous@rxnode1]: reconnect failed: connection unknown
CLIENT 1
Server....................... tcp://rpnode1:7222,tcp://rpnode2:7222
User......................... (null)
Destination.................. topic.test
Message Size................ 1024
Count......................... 10000
Production Threads.......... 1
Production Connections...... 1
DeliveryMode............... PERSISTENT

Publishing to destination 'topic.test'
javax.jms.JMSException: Connection has been terminated by the server
  at com.tibco.tibjms.Tibjmsx.buildException(Tibjmsx.java:440)
  at com.tibco.tibjms.TibjmsSession._publish(TibjmsSession.java:1017)
  at com.tibco.tibjms.TibjmsMessageProducer._publish(TibjmsMessageProducer.java:238)
  at com.tibco.tibjms.TibjmsMessageProducer.send(TibjmsMessageProducer.java:348)
  at tibjmsMsgProducerPerf.run(tibjmsMsgProducerPerf.java:232)
  at java.lang.Thread.run(Thread.java:534)

CLIENT 2
Production Threads.......... 1
Production Connections...... 1
Ack Mode..................... AUTO_ACKNOWLEDGE

Subscribing to destination 'topic.test':
javax.jms.JMSException: Connection has been terminated by the server
  at com.tibco.tibjms.Tibjmsx.buildException(Tibjmsx.java:440)
  at com.tibco.tibjms.TibjmsSession._confirmNonTransacted(TibjmsSession.java:2330)
  at com.tibco.tibjms.TibjmsSession._confirm(TibjmsSession.java:2428)
  at com.tibco.tibjms.TibjmsSession._doAutoAcknowledge(TibjmsSession.java:2895)
  at com.tibco.tibjms.TibjmsSession._receive(TibjmsSession.java:1238)
  at com.tibco.tibjms.TibjmsMessageConsumer._receive(TibjmsMessageConsumer.java:202)
at com.tibco.tibjms.TibjmsMessageConsumer.receive(TibjmsMessageConsumer.java:310)
at tibjmsMsgConsumerPerf.run(tibjmsMsgConsumerPerf.java:229)
at java.lang.Thread.run(Thread.java:534)

The power-cycle test was rerun after setting ft_reconnect_timeout to 3600 seconds.
The primary node was power-cycled. The secondary took over within 20 seconds. The clients did not notice the lost primary for about 10 minutes. The clients successfully reconnected to the backup and finished without message loss on the failsafe PERSISTENT topic.

Log from the backup

2005-03-25 15:33:28 Server is now active.
9 For more information

HP servers, [www.hp.com/go/servers](http://www.hp.com/go/servers)
HP storage, [www.hp.com/storage](http://www.hp.com/storage)
HP Serviceguard, [www.hp.com/go/serviceguard](http://www.hp.com/go/serviceguard)
HP-UX, [www.hp.com/go/hpux](http://www.hp.com/go/hpux)
VERITAS Cluster Server,
VERITAS Cluster File System,
TIBCO Enterprise Message Service,

To help us improve our documents, please provide feedback at [www.hp.com/solutions/feedback](http://www.hp.com/solutions/feedback).