Performance and Sizing Guide

HP Operations Manager 9.01 on Linux

Introduction

The biggest challenge when measuring the performance of distributed network and system management applications such as HP Operations Manager (HPOM) on Linux is not measuring the performance itself, but determining the parameters that affect the performance of the application.

The following types of parameters impact the performance:

- Static parameters
    - For example, disk space requirements, kernel configuration, and so on
- Dynamic parameters
    - For example, memory usage of HP Operations Manager on Linux processes and CPU utilization
- Network parameters
    - For example network usage and protocol overhead (for example, jumbo frames)
- Application related parameters
    - Number of operators working in parallel
    - Number of messages per second received by the management server
    - Type and complexity of messages to be processed by the management server
    - Number of active and history messages in the message browser
    - Number of nodes in the topology/object database that are maintained and monitored by HPOM on Linux
    - Number of service objects in Service Navigator
    - Number of custom message attributes in messages

There is no fixed limit for the number of nodes that an HP Operations Manager on Linux management server can manage[1]. The capability of an HP Operations Manager on Linux management server is based on the number and types of messages that must be processed. The number of nodes depends on the setup of the management domain. The more processing HP Operations agents perform on the managed node (for example, local automatic actions or message filtering), the less performance impact can be observed on the management server.

This guide exclusively deals with performance values seen on Linux systems.

Performance Findings

In large-scale HP Operations Manager on Linux environments, many factors determine the optimal setup with the best performance. In the long run, there are trade-offs between performance, security, hardware and software costs, and the total cost of ownership. This document highlights a few key performance findings:

- Various scenarios of HPOM message processing and throughput on the HPOM server
- Startup time of the HPOM Java UI (including Service Navigator)

[1] Though we do not recommend connecting more than 3000 nodes fitted with agents (aka "managed nodes") directly. The limitation is based on the number of connected agents to the system in parallel. External/message allowed nodes can be connected in a much higher number: customer databases have been seen with up to 100.000 external/message allowed nodes – increased history download time when not using the "no orphan" settings have been observed though.

Note:
The HPOM on Linux server is a 64-bit application, running on a 64-bit operating system only and uses a 64-bit Oracle database. This is the first HPOM version ever to use and consist of 64-bit components only.

Summary of Performance Findings

The following HPOM on Linux performance findings have been measured:

- Management server
    - The number of used CPUs depends heavily on the number of concurrent Java GUIs, because each Java UI (with a different user login) is served by its own instance of an opcuiwww process.
    - Depending on the hardware, number, and usage profile of concurrent GUIs and MSI applications, the HPOM server can sustainably process several hundred HPOM messages per second.
    - Although the number of managed nodes has no significant impact on both the Java GUI and the Oracle database message throughput, it does have an impact on some HPOM server processes.
    - If HPOM duplicate message suppression is enabled, message keys should be explicitly set for the messages. To prevent poor throughput a unique message key is generated automatically (to suppress this feature set the variable OPC_CREATE_AUTO_KEYS to FALSE in the opc namespace).
    - Using MSI on the HPOM agent and the HPOM server with more than five simultaneous MSI programs results in decreasing message throughput.
    - Using MSI on the HPOM agent and the HPOM server each with MSI programs in sequence can lead to message delay.
    - HPOM servers can survive massive event storms and then continue normal operations after a short period of time.
- Manager-of-Manager (MoM)
    - With forwarding enabled the data rate into the local database is more than 10% slower than without forwarding enabled.
    - The message rate into the forwarding target's database is around 50% of the sending one.
    - Forwarding only those messages required on the destination HPOM server minimizes performance overhead.
    - The time to synchronize HPOM message acknowledgements is linear and based on the number of acknowledged messages.
- Oracle Database
    - Reducing the bulk of acknowledged messages to less than 5,000 messages avoids disk and network bottlenecks.
    - HPOM history message download speed is about 700 to 800 messages per second.
    - Time required to download history messages depends on the configuration of the Oracle database parameters, particularly the log file groups and buffers.
    - When downloading a large number of HPOM history messages, using a higher bulk commit size than the default improves the performance.

- Java GUI (including Service Navigator)
    - As more Java GUIs are started, message throughput decreases in a linear fashion, as long as there are still CPUs available to serve the associated GUI processes. If no more CPUs are available, the message throughput decreases faster.
    - HTTPS is the default communication method to the Java GUI. It is faster and more secure than HTTP.
    - Number of managed nodes has no significant impact on the start-up time of the Java GUI.
    - A significant impact on the start-up time of the Java GUI is observed with 50,000 and more active messages to be loaded. If there are too many messages to be loaded, the frontend can run into a timeout. You can tweak this parameter[2] on the frontend to allow a longer load time.
    - Keeping the number of active messages as low as possible (by using duplicate suppression, event correlation, and so on) reduces the start-up time for the Java GUI.
    - Java GUI preference "Show All Messages" should be used only if really needed.
    - Only the number of Java GUIs needed at the moment should be started and running.
    - When using CMAs, the additional performance degradation of the Java UI start-up time and the message throughput is low. It is recommended that you keep the number of CMAs at or below 10 CMAs per message.
    - To increase the usability of the Java GUI, it is recommended that acknowledged messages should be marked only (by setting the HPOM server configuration variable OPC_DIRECT_ACKN_LIMIT to 0). Using this option will ensure that the acknowledged messages get moved to the History table automatically at a later point of time.
    - If deep Service Navigator graphs are used (for example, service graphs with more than five levels), it is recommended that you use the feature "Service Load on Demand" to avoid long delays in the start-up time of the Java GUI.

- HTTPS agent
    - Depending on the hardware and other competing applications, one HTTPS agent is able to send up to 4000 messages per second (using opcle).
    - Maximum transfer rate from the agents to the server is faster than the message processing rate on the server.

- Server Pooling
    - HPOM on Linux Server Pooling provides an easy-to-administer method for managing large-scale environments combined with the capability to quickly switch the workload to different HPOM servers if an HPOM server system is not reachable or is in maintenance mode. For details, see the *High Availability Through HPOM Server Pooling* white paper. HPOM on Linux Server Pooling is also an attractive alternative to hardware-based high-availability setups[3].

[2] "reconnect_timeout" which can be found in the local itooprc of the GUI node

[3] As seen during tests the forwarding rate is around 50% compared to the one without forwarding into the DB. With enabled forwarding the data rate into the local database is 10% slower than without forwarding enabled.

Unmeasured Performance Findings

The following HPOM on Linux performance findings have *not* been measured:

- HTTPS agent performance

  You can find some HPOM HTTPS agent performance-related measures in the *HP Operations Manager for UNIX 8.10 Performance and Configuration Guide*.

- Local or remote Oracle database

  Running a remote Oracle database offloads the HPOM server with CPU and main memory usage. However, remote SQL statements may also have performance and security-related limitations.

- Physical system or virtual machine

  HPOM on Linux can be run on a physical system or on a virtual machine (VM). Although running HPOM on Linux on a VM can decrease performance somewhat, the ease of administration (for example, setting up multiple HPOM servers using VM images, separating "competing" applications in different VMs, and so on) makes it an option worth considering[4].

- Speed of name resolution on the management server

  During the startup of the HPOM server, all node names currently recognized by HPOM on Linux are resolved to fill the corresponding HPOM server process caches for fast processing. Depending on the speed of Domain Name Service (DNS), this name resolution could take a considerable amount of time. Best answer time has been experienced using the local /etc/hosts file.

Executive Summary

The tests show how HPOM 9.01 on Linux (64 bit) performs. For details about the HPOM 9.00 on HP-UX performance tests, see the corresponding Performance and Sizing Guide.

Note:
The core components of the HP Operations agent can send messages at a higher rate than an HPOM management server on any platform can process. (This was verified with the logfile encapsulator process opcle, which uses the fewest resources.) This document therefore does not cover the agent performance.

The tests are described in detail in the section "Test Results" on page 10. For your reference, the description of each test includes the metrics that were measured, the parameters that were varied, the test scenario, the test results, and any conclusions that can be drawn from the test results.

Hardware Recommendations for Optimal Message Throughput

CPU[5]

- 3 cores for the base system
- 1 core for message forwarding

[4] From other tests done on virtual machine we expect a performance decrease of more than 30% by running in a virtual environment compared to the one seen on the same physical hardware

[5] If the number of CPUs is lower than the calculated one, the system will run anyway without problems in a normal production environment. The peak performance mentioned in this document will not be reached though.

- 1 core for each starting Java GUI[6]
- 1 core per 50 nodes (sending messages in parallel – currently "silent" nodes do not count)
- 1 core per active Admin UI
- 0.1 cores per running Java GUI

In general, it is recommended to use cores that are faster rather than many. (The process that represents the bottleneck (opcmsgm) is single threaded.)

Memory[7]

- Oracle 11g default memory depends on the number of CPU cores in your system – see Oracle documentation for further information
- 1.2GB for the Admin UI
- 256MB per opened Admin UI
- 3GB for other HPOM processes (4GB for more than 200,000 messages in the database)
- More than 256MB per Java GUI (depends on the number of messages in the database and the size of the service tree)
- 1GB for base system

Disks

- 20 to 30GB for the database, fast disks, non-journaling[8,9]
- More than 4GB for /var/opt/OV (depends on the number of downloads to the default location)
- 4GB for /opt/OV
- 500MB for /etc/opt/OV

Peak Performance of HPOM on Linux

The tests with HPOM on Linux show a maximum message throughput of up to 2,800 messages per second (in bulked mode from the agent) on a dedicated system. This system was only available for basic throughput tests – all given values have been measured with the machine mentioned.

Two different systems were tested: 4Core Xeon at 2.83GHz with 8GB memory (~1,400 msg/s peak) and 32Core Opteron at 2.9GHz with 256GB memory (~2,800 msg/s). The performance boost observed with the latter system is due to better IO architecture of the Opteron CPUs.

Startup time of the Java GUI

The startup time of the HPOM on Linux Java GUI is faster than on any HP-UX systems seen. Though the speedup (less than 20%) is not as significant as the increased message throughput rate.

Test Environment

Management Server Test Setup

Hardware System

Network	Dedicated 1GigaBit network for the display stations and for the managed nodes.
Machine Type	HP dl460c

[6] Each CPU spent on an Admin UI and Java GUI requires an additional one for the DB server (independently if the DB is local or remote).
[7] The memory can be lower than the calculated one. If given memory is too low, swap will be heavily used and – in case of a message storm – will slow down the system additionally due to its increased memory consumption.
[8] The default DB can only extend up to 8GB at max – contact your DBA for your current settings
[9] Spreading the tablespaces over a number of disks does not give a huge boost

Processor	1 x Intel Xeon E5440 (4core@2.83Ghz)
Main Memory	8 GB
Disk space	fiber channel (EVA 4000)
OS	RedHat Enterprise Linux 5.3
Swap space	10 GB [10]
OS-Patches	none

Kernel Configuration

The kernel parameters were checked during the HPOM setup according to the answers given in this phase:

Parameters used for the configuration:

Configuration Parameter	Value
Number of Java GUIs[11]	50
Service Navigator GUIs	5
Number of HTTPS agents	10000

HPOM on Linux Version

- Base Version A.09.01

[10] Swap space was actually never used since the system never required more memory than RAM existed. There always was RAM available for caching.

[11] On RHEL5.3, the number of simultaneous xinetd-started processes had to be extended for this test.

RDBMS Installation

- ORACLE 11.1.0.7
- Configurable parameters in file initopenview.ora [default values in brackets].

Parameter	Value [Default Value]
memory_target	512M [0]
db_files	80 [50]
db_file_multiblock_read_count	32 [16]
log_buffer	1572864 [65536]
The redo logs	5 x 100 MB [3 x 20 MB] 5 log groups with one log file each.

- Tablespaces in Oracle were created as unlimited bigfiles.

Display Stations Test Setup

Windows

Used for JAVA GUIs.

Machine Type	2 x dl460c
Main Memory	6 GB
CPU	Xeon E5450 (Quad, 3GHz)
OS	Windows 2003 R2

Managed Nodes

No physical nodes were connected (except for the management server itself). The database was loaded with real IP nodes (proxied by the existing server).

Message Generation

HP LoadRunner 9.50 was used to send messages. To test bulking, the logfile encapsulator (opcle) was used. CPU load showed that the logfile encapsulator is faster than the message agent (opcmsga), which in turn is faster than the message manager (opcmsgm) on the server. So any agent can fully load the server with a simple logfile policy.
When there are lots of messages generated on a node the sending part of the agent will go into bulk mode; that is, it will send the messages not one by one but multiple of them in one large package (similar to jumbo frames in networking). Using this bulking mode speeds up database throughput by 100% (HP LoadRunner versus bulking agent) because of easier preprocessing on message reception.

Miscellaneous

- All nodes in the node groups are defined as ordinary managed nodes of type `WIN32`, with a name and IP address (these nodes were not reachable via IP though). Since the agents on these nodes cannot directly send messages, they are referred to as *proxied* nodes. All IP addresses were resolved via `/etc/hosts` to achieve faster IP and name resolution (`/etc/nsswitch.conf` was changed accordingly).

- Real managed nodes (that also generate a few messages for themselves) are used to generate the messages for the *proxied nodes*. The HP Operations agent is installed on the *real managed nodes*.
- Tests have shown that there are differences in the message throughput regarding *real nodes* versus *proxied* nodes (probably due to increased lookups for assignments). From these results, a better performance in message throughput may be expected for *real nodes*.

The Message Profiles

The Message Generator is able to create messages of various types. These types are flagged with a special code in the message text which triggers a dedicated condition in the assigned *Message Interface* template.

The following message types are available:

NO	Normal messages
OL	Normal messages, but with flag "on server log only"
AA	messages with automatic action (Echo50)
AN	see profile AA, but output of automatic action is recorded as annotation (size of annotation for the Echo50action is 50 characters)
AC	see profile AA, but message is automatically acknowledged
ACN	see profile AC, but output of automatic action is recorded as annotation
NT	messages which are forwarded to the notification service
TT	messages which are forwarded to the trouble ticket service
IN	messages with fixed instructions

Test Results

The following sections describe the executed tests in more detail.

Message Throughput: Varying the Number of Active Java GUIs

Table 1 Metrics and Parameters

Metric	Number of messages per second (message throughput), until all messages are shown in all Java GUIs. Number of messages per second (message throughput), until all messages are stored in the HPOM database.
Parameter	Number of Java GUIs.

Table 2 Test Scenario

Measured Value	Time until the last expected message is shown in all message browsers is taken as the result of this test. Time on server to receive all messages (Receive Time of message) divided by number of messages is taken as the "Rate of messages stored in the HPOM database".
Message Generator	5,000 messages, severity normal. Messages are targeted to 100 nodes in four node groups, that is 50 messages per node. Operators used are responsible for 400 nodes. Node Bank has 2,000 managed nodes. No history messages.
Java GUI Options	Same HPOM user account.[12] Show all messages. Refresh interval 5 seconds. [13]

Test Results

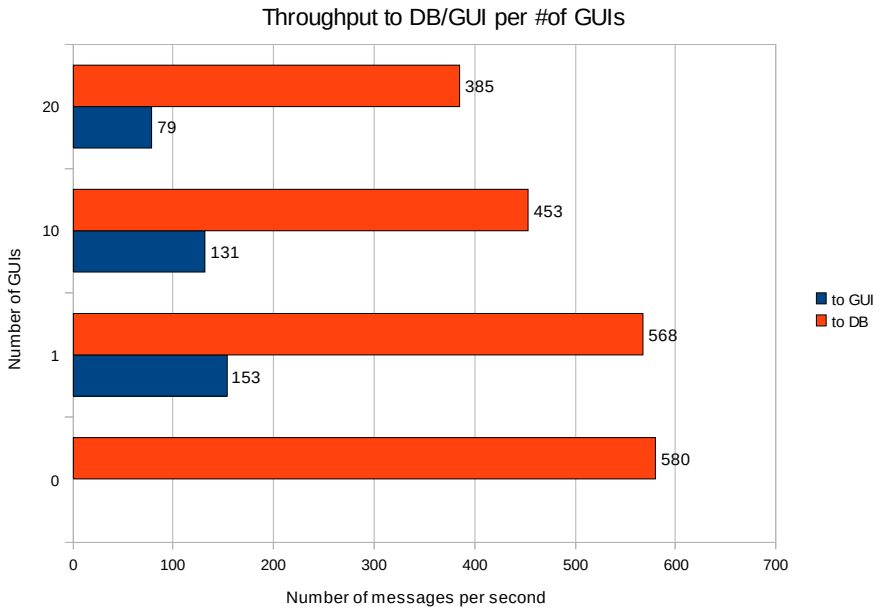
"Figure 1 Message Throughput - Number of Java GUIs" shows the message throughput based on the number of running Java GUIs. The message generator is the local HP Operations agent.

Compared to the HPOM for HP-UX tests, a performance decrease can be observed already for the first Java GUI. This is because the opcuiwww process uses most CPU power, which is already at its limit with only four CPUs. (This test could not be carried out on the dedicated, more powerful test system.)

[12] Given that there are enough CPUs, different user account won't slow down the system very much – but keep in mind that there is some additional overhead in the server though (opcdispm).

[13] The refresh interval has an impact on the outcome of the bulk transfer of messages to the Java GUI: the smaller the refresh interval, the smaller the performance gain of the bulk transfer. For most of the Java GUI tests, the smaller refresh value was used to get more timely results. The recommendation for production use is still 30 seconds – which is the default in the Java GUI.

Figure 1 Message Throughput - Number of Java GUIs



The impact of the network communication protocol of the Java GUI (HTTP versus HTTPS communication) was not part of this test. Tests with HPOM for HP-UX show a performance penalty of 50% for using HTTP instead of HTTPS.

Message Throughput: Duplicates Message Suppression

Table 3 Metrics and Parameters

Metric	Number of messages per second (message throughput), until all messages are shown in the Java GUI.
Parameter	Message duplicate counter active versus inactive. Number of different message keys (0, 1, 10,000).

Table 4 Test Scenario

Measured Value	Time until the last expected message has arrived in the database.
Message Generator	500,000 messages, severity normal, all messages were different (message text contained a unique number) but only 1 respectively 10,000 different message keys were used, bulked. All messages are delivered during the tests (to conform to HPOM for Windows tests). Since the local agent is faster than the opcmsgm (opcle seen at 3125msg/s, opcmsga at 2270msg/s), except for the CPU load, there is no penalty to deliver the messages online. Messages are targeted to 10,000 nodes in four node groups, that is 50 messages per node. 1 CMA per message. Operators used are responsible for 400 nodes. Node Bank has 50,000 managed nodes. No history messages.
Java GUI Options	No Java GUIs were started.
Miscellaneous	The feature "Add duplicates as annotations" was turned off (default).

Because of the delivery speed, it was not possible to monitor the throughput decrease while receiving the 10,000 message keys (less than 9s).

Test Results

The OPC_CREATE_AUTO_KEYS feature reduces performance problems with empty message keys.

Table 5 Message Throughput - Duplicates Counter

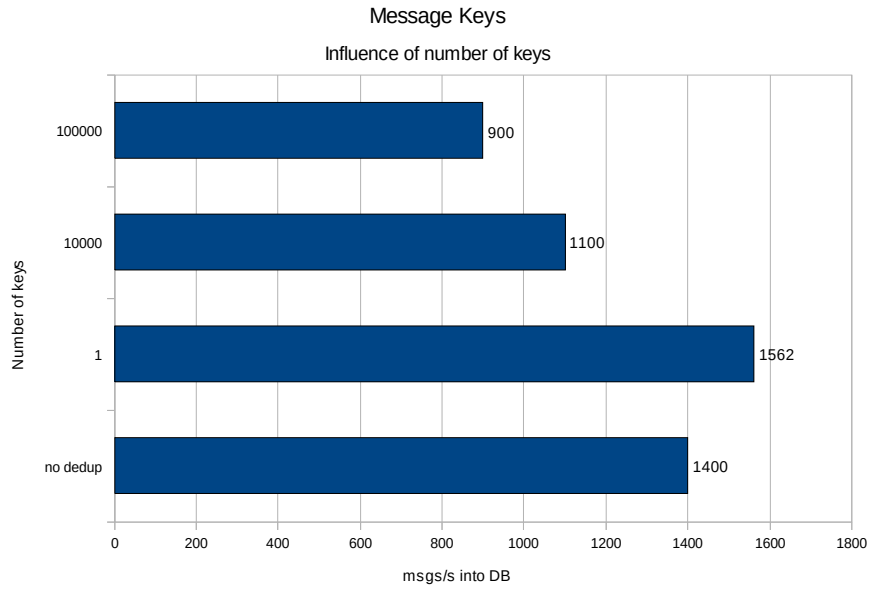
Message Keys	OFF	Deduplication enabled/no message key explicitly set by policy[14] (0)	1	10,000
Msgs/sec	1400	900[15]	1562	1100

OFF	HPOM feature *Message Duplicates Counter* was turned **off**
0	*Message Duplicates Counter* turned **on**, no message keys explicitly set
1	*Message Duplicates Counter* turned **on**, same key for all messages
10,000	*Message Duplicates Counter* turned **on**, using 10,000 different keys

[14] Message keys are automatically created when OPC_CREATE_AUTO_KEYS is set to TRUE. Setting this value to FALSE as seen in previous HPOM performance guides will degrades performance exponentially with the number of active messages!

[15] At 100,000 messages.

Figure 2 Message Throughput - Duplicates Counter



The following facts may be derived from these results:

- The number of different message keys has an impact on the performance: the more message keys there are already in the database, the longer the time to process the messages.

Hints for own message key creation and message key mapping in general:

- The message key should be reasonable short:
  less than 255 characters
- When mapping for pattern use anchoring:
  "^" and/or "$"
- Use pattern instead of multiple possible matches:
  for example, "^UP_<*>" instead of "^UP_foo|^UP_bar|^UP_blah"
- Avoid pattern combinations:
  for example, do not use "some<*><@><*>thing"

Message Throughput: Diverting Messages to MSI Programs

Table 6 Metrics and Parameters

Metric	Number of messages per second (message throughput), into the database via the message stream interface (MSI).
Parameter	Number of MSI programs. MSI active on server.

Table 7 Test Scenario

Measured Value	Throughput into the database.
Message Generator	100,000 messages, severity normal, not bulked. Messages are targeted to 100 nodes in four node groups, that is 50 messages per node. Simple messages (1 CMA, no message mix). Operators used are responsible for 400 nodes. Node Bank has 50,000 managed nodes. No history messages.
Java GUI Options	No Java GUIs were started.

Test Results

The following table shows the results of this test.

# of MSIs	Throughput into database [msgs/s]
0	800
1	800
2	800
10	800[16]

The internal mechanism of diverted MSIs in sequence works like “read message from input queue” → “process in MSI1” → “put message back to input queue” → “read message from input queue” → “process in MSI2” and so on.

Under heavy load and very lightweight MSIs, we saw on the HPOM on Linux system a ratio of 1:2 processed to new events in the incoming queue. In the very rare circumstance of 10 MSIs, the ratio of messages into the database versus incoming messages would be $1:2^{10}$. This will lead to a massive growth of the message queue (msgm) and a perceived low incoming rate to the database while the message storm continues. It is therefore strongly advised to use the tools provided to prevent such a message storm.

The following facts may be derived from these results:

- The number of MSI programs does not affect the throughput into the database.
- Too many MSI programs will delay message delivery in case of a message storm.

Conclusions:

- Using a small number of MSI programs results in no performance loss.

[16] Peak rate after all messages have been processed in the MSIs.

Java GUI Startup Time: Varying the Number of Managed Nodes and Messages

Table 8 Metrics and Parameters

Metric	Startup time of two Java GUIs.
Parameter	Number of managed nodes for which users are responsible. Number of active messages.

Table 9 Test Scenario

Measured Value	Time until the last GUI is started and fully functional.
Message Generator	5,000 and 50,000 messages, severity normal, were generated before the test started. Messages are targeted to 100 nodes in four node groups, that is 50 and 500 messages per node. Operators used are responsible for 500 and 2,000 nodes. Node Bank has 50,000 managed nodes. No history messages.
Java GUI Options	Different HPOM user accounts. Show all messages. Refresh interval 5 seconds.

Test Results

"Table 10 GUI Startup Time - Managed Nodes / Active Messages" lists the results of this test: the time in seconds needed to start two GUIs with different HPOM on Linux user accounts.

For the GUIs, we observed a significant difference between the time for the first and the following GUIs where 50,000 active messages were shown. The time for the first GUI is listed first, then the time for the following GUIs. Note that the times seen varied over a wider range – the more GUIs the larger the times varied.

Table 10 GUI Startup Time - Managed Nodes / Active Messages

	Responsible Nodes	
	500	2.000
Active Messages: 5,000	6	10
Active Messages: 50,000	63 / 70	43 / 79

The following facts may be derived from these results:

- The number of managed nodes in the HPOM user's realm has almost no impact on the startup time of the Java GUI.
- The number of active messages has an impact on the startup time of the Java GUI, although a significant increase is seen only in the range of 50,000 messages and above.

Conclusions:

- Up to 2,000 managed nodes and 50,000 active messages, the startup times of the Java GUI are approximately 80 seconds. Thus, too many active messages should be prevented (use duplicate suppression, event correlation, and so on).

Java GUI Startup Time: Varying the Number of CMAs

Table 11 Metrics and Parameters

Metric	Startup time of two Java GUIs displaying all CMAs attached to the HPOM messages.
Parameter	Number of CMAs attached to the messages.

Table 12 Test Scenario

Measured Value	The time until the last Java GUI is up and functional, that is messages are displayed in the message browser. The Java GUI was configured in such a way that all CMAs were shown in the browser.
Message Generator	10,000 messages, severity normal, generated before the test started. A varying number of CMAs was attached to each message. Each CMA had a length of 20 characters. Messages are targeted to 100 nodes in four node groups, that is 10 and 100 messages per node. Operators used are responsible for 500 nodes. Node Bank has 10,000 managed nodes. No history messages for this test.
Java GUI Options	Different HPOM user accounts. Show all messages. Refresh interval 5 seconds.

Test Results

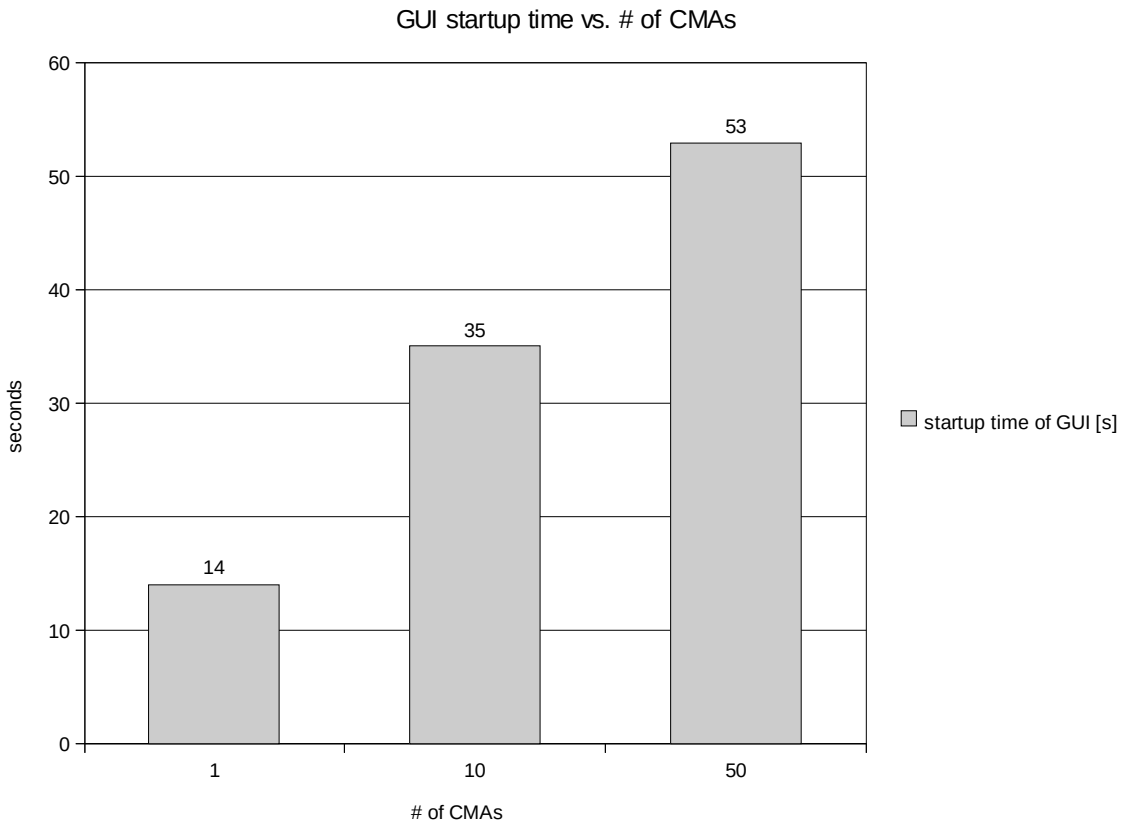
"Table 13 Java GUI Startup Time - CMAs in Active Messages" lists the results of this test: the time in seconds needed to start two Java GUIs with different HPOM user accounts, with a varying number of active messages (and CMAs), and no history messages.

Table 13 Java GUI Startup Time - CMAs in Active Messages

Number of CMAs	1	10	50
Active Messages: 10,000	14	35	53

"Figure 3 Java GUI Startup Time - CMAs" shows the test results. (There is no graph showing tests with history messages only.)

Figure 3 Java GUI Startup Time - CMAs



The following facts may be derived from these results:

- The number of CMAs per message itself has only a small impact on the startup time of the Java GUI.
- The delay in the startup time of the Java GUI depends on the absolute number of different CMAs per message in a non-linear function.

Conclusions:

- CMAs increase the startup time of the Java GUI.[17]

[17] Tests regarding the message flow into the database showed that already the delivery into the database suffers when using many CMAs (throughput with 50 CMAs/message is reduced to nearly a seventh of the maximum peak rate).

Service Navigator

Note:
Service Navigator wasn't part of this performance test. Performance tests for HPOM Service Navigator are planned for a later revision of this document.
Taking the other found values into account we would at least expect the system to perform equally to an HP-UX system.

Throughput when forwarding

Table 14 Metrics and Parameters

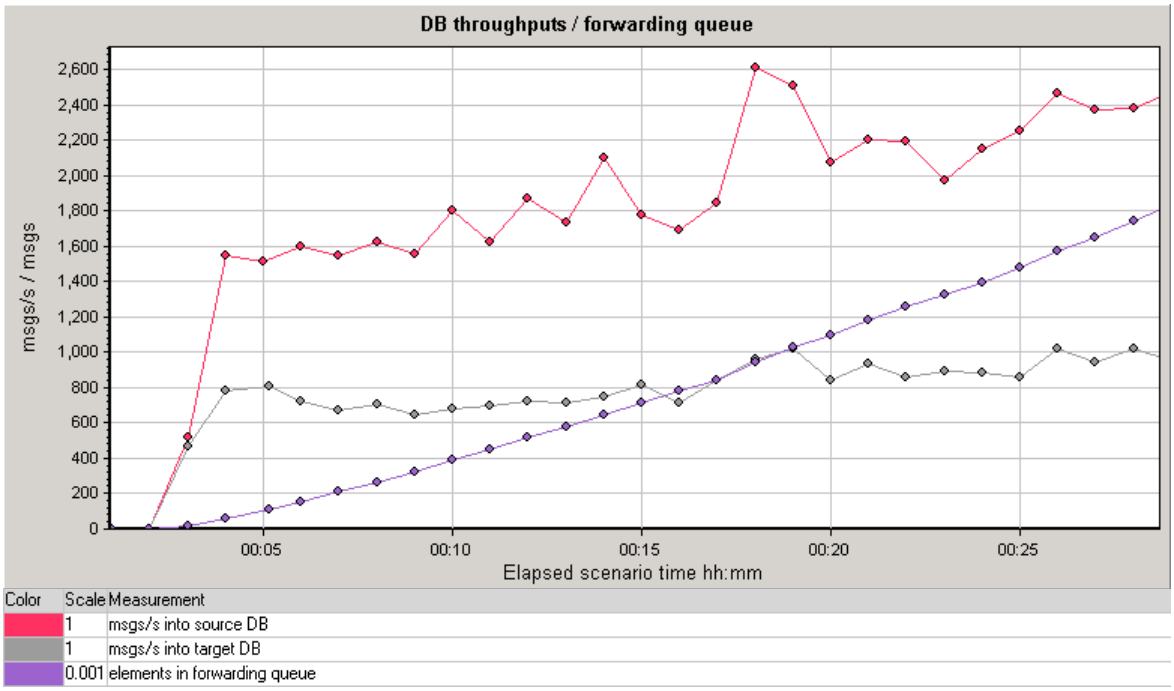
Metric	The message rate into an equally fitted system
Parameter	none

Table 15 Test Scenario

Measured Value	Throughput to local and remote DB Status of queues on both systems
Message Generator	Messages of severity normal are generated by the local agent of the master server No active messages before starting test Node Bank has 50,000 managed nodes
Java GUI Options	No Java GUIs were started for this test
Environment	Two identical machines building server pool Machines are different from those used in the other tests: 32 core Opterons with 256GB of memory

Test Results

Figure 4 DB throughput local and remote MoM



The following facts may be derived from these results:

- Processing is slightly decreased on the source server (not visible in the graph)
- Message rate into target DB is roughly half of the one on the source server
- The queue of the forward manager increases => the forward manager is unable to get rid of the message to the bbc transport mechanism in time
- No queue for https builds up (not shown here) –> https is not a bottleneck

Conclusions:

- In case of using server pooling cut down the throughput expectations into half: the secondary server's rate into the DB is ~50% of the primary one. After the message storm the secondary server needs some time to catch up with the primary one.

Note:

- The machines used are the second ones mentioned in Hardware System and the results have to be cut at least by half to compare them to the other values in this guide

Performance of the History Download: Varying the Number of Downloaded Messages

Table 16 Metrics and Parameters

Metric	The time to download a varying number of history messages.
Parameter	Oracle Log Buffer and Log File Groups.

Table 17 Test Scenario

Measured Value	This test measures the time needed to download a varying number of history messages. The download tool opchistdwn was started on the command line.
Message Generator	100,000 history messages of severity normal were generated before this test No active messages No messages were generated during the test Node Bank has 10,000 managed nodes.
Java GUI Options	No Java GUIs were started for this test.
Environment	Automatic downloading of history messages was disabled. Default HPOM configuration regarding marking and moving acknowledged messages was used. The HPOM message mover process was disabled (OPC_ACK_MOVE_INTERVAL=0).

Test Results

"Table 18 Performance of History Download" lists the results of this test. Shown are the times in seconds beginning with the start of the *opchistdwn* utility until its termination. In addition, the messages downloaded per second are shown, too.

Table 18 Performance of History Download

Configuration	Config A	Config B
Time for download [s]	1,846	183
Time [s] / 1.000 messages	18.46	1.83
Messages per second	54.2	546

Legend:

Config A	Oracle Log Buffer size = 1.5MB 3 Log File Groups, one file per group, 20MB per file
Config B	Oracle Log Buffer size = 1.5MB 6 Log File Groups, one file per group, 100MB per file HPOM configuration variable OPC_UPDWN_COMMIT_COUNT = 1000

The opchistdwn process took approximately 50% of one CPU, the Oracle database disk array was used up to 20%.

The following facts may be derived from these results:

- The download speed can be optimized to reach about 540 messages per second.
- The time needed for the download of history messages depends on the configuration of the Oracle database parameters, especially the log file groups and buffers.

Conclusions:

- The download of history can be simply extrapolated once a reference download has been timed.
- The number of nodes has a huge influence on the orphan search after the download part. Omit the "-no_orphan" option only from time to time

Appendix

Ext2 File System Tuning

- No file system parameters were tuned.
- To find out if the file system has any influence on the performance, all agent and server queues were put on a RAM disk but without any noticeable results.

May 2010