HP OpenView Operations for UNIX

Performance Guide

Software Version: A.08.10

Manufacturing Part Number: None
24 March 2005

U.S.A.

Legal Notices

Warranty

Hewlett-Packard makes no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

A copy of the specific warranty terms applicable to your Hewlett-Packard product can be obtained from your local Sales and Service Office.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company
United States of America

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c) (1, 2).

Copyright Notices

Trademark Notices

HP-UX Release 10.20 and later and HP-UX Release 11.00 and later (in both 32 and 64-bit configurations) on all HP 9000 computers are Open Group UNIX 95 branded products.

Intel386, Intel80386, Intel486, and Intel80496 are U.S. trademarks of Intel Corporation.

Intel Itanium™ Logo:Intel, Intel Inside and Itanium are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries and are used under license.

Java™ is a U.S. trademark of Sun Microsystems, Inc.

Microsoft® is a U.S. registered trademark of Microsoft Corporation.

OpenView® is a registered U.S. trademark of Hewlett-Packard Company.

Oracle® is a registered U.S. trademark of Oracle Corporation, Redwood City, California.

OSF, OSF/1, OSF/Motif, Motif, and Open Software Foundation are trademarks of the Open Software Foundation in the U.S. and other countries.

Pentium® is a U.S. registered trademark of Intel Corporation.

Disclaimer

The results published in this document are valid only for the following software versions:

- **OVO/UNIX A.08.10.160**
- **Oracle 9.2.0.2/64**

Table of Contents

Table of Figures

Index of Tables

1 Introduction

With the growing importance of products addressing Integrated Network and Systems management (INSM), HP OpenView Operations for UNIX (OVO/UNIX) will be used more and more in large computing environments. The nature of OVO/UNIX lends itself to be the management tool of choice for regional or enterprise-wide management activities.

The biggest challenge when measuring the performance of distributed network and system management applications such as OVO/UNIX is not the measurement itself, but the determination of the set of parameters having an effect on application performance. Because it is not possible to provide general rules, this guide focuses on the limitations and critical areas.

From a high level point of view there are three different kinds of parameters:

- static parameters (server and/or agent specific) like disc space requirements and kernel configuration
- dynamic parameters (server and/or agent specific) like memory usage of OVO/UNIX processes and CPU utilization
- network-related parameters like network utilization and protocol overhead

In addition, there are other application related parameters:

- number of operators working in parallel
- number of messages per second received by the OVO/UNIX management server
- type of messages to be processed by the OVO/UNIX management server
- number of active and history messages held in the message browser
- number of nodes in the topology/object database that are maintained and monitored by OVO/UNIX
- number of service objects in Service Navigator
- number of custom message attributes in messages

The question is often raised as to "how many OVO/UNIX agent nodes can be managed by a single OVO/UNIX management system". Due to conception aspects it is not possible to give a simple answer to this question. There is no fixed limit for the number of nodes which can be controlled by an OVO/UNIX management server. The capability of an OVO/UNIX management server is based on the number and type of messages which have to be processed. Therefore, the number of nodes which can be managed by a server heavily depends on how the management domain is set up. The more features of the OVO/UNIX intelligent agents like local automatic actions and message filtering are used, the less is the impact on performance of the OVO/UNIX management server.

2 Executive Summary

The main focus of the current OVO/UNIX performance tests was to show the performance of the new OVO HTTPS agents compared to the traditional DCE agents.

In addition, the most interesting tests known from previous versions of the OVO/UNIX Performance Guide were repeated for the new OVO/UNIX 8.10 release.

Some new test areas have been added, e.g. the performance of the SNMP trap interceptor, the performance of the OVO monitor agent and the performance of the opcmon/opcmsg command line interface.

Performance of OVO/UNIX 8.1 vs. OVO/UNIX 7.1

Tests regarding the message throughput have shown a maximum message processing rate on the server of up to 120 messages per second for OVO 7.1.

The current tests have shown a maximum message throughput of up to 120 messages per second, too.

➔ With OVO/UNIX 8.1, there is no loss in message throughput / message processing time on the management server.

Performance of OVO HTTPS agents vs. OVO DCE agents

Comparing the message processing rate on the OVO server based on the agent type (HTTPS vs. DCE), it can be seen that there is no significant difference between these agent types. Although the internals in the agent are implemented in a different way, the messages on the server are processed at a comparable rate.

The memory consumption of the OVO HTTPS agents is higher than the requirements of the DCE agents. The tests have shown that approx. 50% more main memory is needed by the HTTPS agent.

➔ There is no significant difference regarding the message processing rate between HTTPS and DCE agents.

➔ More main memory should be available for the HTTPS agents than for the DCE agents.

Startup time of the Java GUI in OVO/UNIX 8.1 vs. OVO/UNIX 7.1

Looking at the time needed to start the Java GUI in identical situations, it can be seen that there is no difference between the OVO/UNIX 8.1 and the OVO/UNIX 7.1 Java GUI for a low number of active messages (5.000). If a higher number of active messages does exist in the message browser (50.000), the OVO/UNIX 8.1 Java GUI needs significantly less time than the OVO/UNIX 7.1 Java GUI to start (60 seconds vs. 100 seconds).

➔ The OVO/UNIX 8.1 Java GUI starts faster than the OVO/UNIX 7.1 Java GUI the more active messages exist.

3 Environment

3.1 Management Server

3.1.1 Hardware System

Network	Dedicated 1000BT network for the display stations and for the managed nodes.
Machine Type	HP RP4440/8
Processor	8 x 875MHz
Main Memory	12 GB
Disk space	2 x 36GB internal 3 x 36 GB, 15kRPM via fiber channel (DS2405)
OS	HP-UX 11i (11.11) / 64
Swap space	512 MB[1]
OS-Patches	Latest OS patches as listed in the OVO installation guide.

3.1.2 Kernel Configuration

The kernel parameters were checked during the OVO setup – according to the answers given in this phase.

The only kernel parameter which needed to be adjusted was `nproc` – it was set to `16.000` instead of the default value.

During the OVO configuration phase, the script asks for sizing parameters and then computes the requirements for some kernel parameters. As a result of this process and the answers given during the configuration, the kernel parameter `nproc` was proposed to be set to `18560`, which in turn would have forced the dependent parameter `shmseg` to an illegal value. Thus, the highest possible value for `nproc` was configured.

Parameters set during the configuration:

Parameter	Value
Number of Java GUIs	50
Service Navigator GUIs	5
Number of Motif GUIs	10
Number of HTTPS agents	100
Number of DCE agents	10

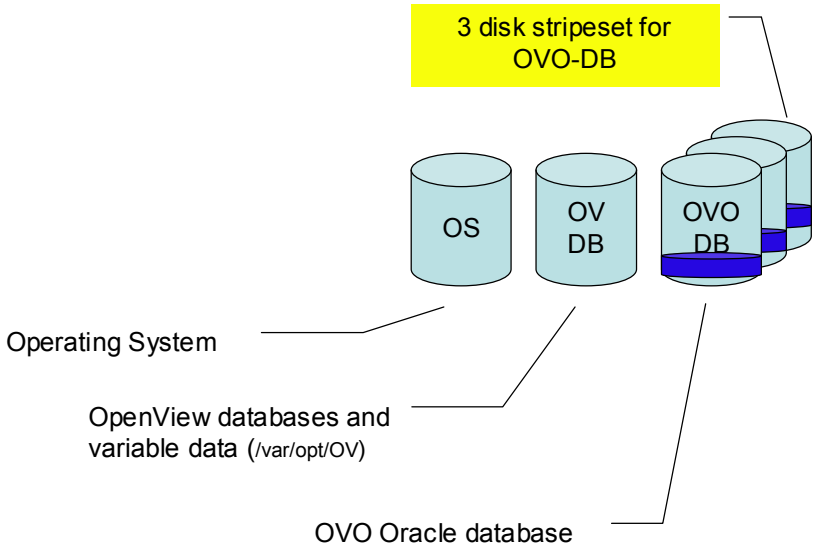
[1] Only a small swap space was reserved because the system was equipped with a significant amount of main memory and configured in such a way that swapping was done entirely in memory. According to the performance specialists, the system should never begin to swap on disks or else the performance test results would be less significant.

3.1.3 Disk Setup

The setup of the disks was fixed for all tests according to the results of previous OVO performance tests:

Five disks were used for the performance tests.

A separate disk for the operating system, a second separate disk for the OpenView databases and variable data, and a 3 disk stripeset for the OVO Oracle database.



3.1.4 OVO/UNIX Installation

- Base Version A.08.10.160
- Linux DCE agent : A.07.25, Linux HTTPS agent A.08.10

3.1.5 RDBMS Installation

- ORACLE 9.2.0.2 / 64 Bit
- Configurable parameters in file *initopenview.ora* [default values in square brackets].

Parameter	Value [Default Value]
db_block_buffers	10000 [550]
shared_pool_size	128M [6000000]
db_files	80 [50]
db_file_multiblock_read_count	16 [8]
sort_area_size	262144 [-]

processes	200 [50]
log_buffer	1572864 [65536]
_attach_count_slack	2000
The redo logs	5 x 30 MB [3 x 20 MB]

Table 1: Oracle Parameters

3.2 Display Stations

3.2.1 HP-UX

Used for the Motif GUIs.

Machine Type	1 x RP3440/4
Main Memory	12 GB
CPU	4 x 750 MHz
OS	HP-UX 11.11

3.2.2 Windows

Used only for the JAVA GUIs.

Machine Type	1 x DL580/4/3.0	1 x DL580/4/3.0
Main Memory	7.5 GB	13.7 GB
CPU	4 x 3.0 GHz	4 x 3.0 GHz
OS	Windows 2003	Windows XP

3.3 Managed Nodes

The real managed nodes (those nodes which were used for the generation of messages) were added manually to the Node Bank, downloaded as configuration **C0** and always reloaded with each of the test configurations (configuration **C0** is loaded via the preparation script and only contains the managed nodes used for message generation).

Machine Type	RP3440 /4	RP3440 /4	RX2600 /2	DL580 /4/1.6	DL580 /4/1.6	DL580 /4/1.6
Main Memory	12 GB	12 GB	12 GB	3.75 GB	3.75 GB	3.75 GB
CPU	4 x 750 MHz	4 x 750 MHz	2 x 750 MHz	4 x 1.6 GHz	4 x 1.6 GHz	4 x 1.6 GHz
OS	HP-UX 11.11	HP-UX 11.11	HP-UX 11.23	W2k3	W2k3	Linux RHAS3.0 _U2
Agent Type	HTTPS	DCE	DCE / HTTPS	DCE	HTTPS	DCE / HTTPS

3.4 The Message Generator

The Message Generator is used to create the test messages. It consists of some programs (using the *opcmsg* API) and a *Message Interface* template.

The managed node generating the messages never turned out to be the critical factor in message throughput. In all situations tested, the management server itself was the element limiting the message throughput.

The Message Generator

- Is started on the real managed nodes running HP-UX.
- May generate messages for a configurable set of nodes (specified through the range of node numbers) and node groups.
- Generates a fixed number of messages per node, and then stops.
- is configurable via a specification file which
  - assigns the real managed nodes to node groups specified on the command line
  - maps the canonical node names node*ng_num* used in message generator to real node names (canonical or systematic node names are used to specify the range of nodes for which messages are generated)
- Each managed node generates messages for the one and only message group.

3.4.1 Miscellaneous

- All nodes in the node groups are defined as ordinary OVO managed nodes of type `OTHER`, with a name, but without an IP address, i.e. these nodes were not reachable via IP.
- Real managed nodes do not generate messages for themselves, but are only used to generate the messages for the *virtual nodes*. The OVO/UNIX agent is installed on the *real managed nodes*.
- The process *opcmsgi* tries to resolve the hostname for nodes given with the *node* parameter of opcmsg. Thus, depending on the `/etc/nsswitch.conf` and the size of the /etc/hosts (if searched), the generation for nodes even of type *Non IP* lasts very long (timeouts of name servers and time used to search the /etc/hosts).

  To avoid these delays, we introduce a very short */etc/hosts* containing only the necessary names and we construct a */etc/nsswitch.conf* which directs the search to the *files* only!

3.4.2 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 list of message types is available:

- Normal messages
- Normal messages, but with flag “on server log only”

- messages with automatic action (`Echo50`)
- see profile AA, but output of automatic action is recorded as annotation (size of annotation for the `Echo50`action is 50 characters)
- see profile AA, but message is automatically acknowledged
- see profile AC, but output of automatic action is recorded as annotation
- messages which are forwarded to the notification service
- messages which are forwarded to the trouble ticket service
- messages with fixed instructions
- a message mix comprised of 40% normal messages, 20% messages with fixed instructions, 20% messages which are forwarded to the notification service, 20% messages with automatic actions

4 Test Results

In this chapter we discuss the results of the tests which were done in the environment described in the previous chapter.

The following areas were covered:

4.1 Message Throughput

4.1.1 Test: Varying the number of active Java GUI's

4.1.1.1 Synopsis

The following metrics are computed in this test.

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 OVO DB
Parameter	• Number of Java GUIs • OVO HTTPS agent versus OVO DCE agent

4.1.1.2 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 OVO DB*".
Message Generator	• 5.000 messages, severity normal • Messages are targeted to 100 nodes in 4 node groups, i.e. 50 messages per node • Operators used are responsible for 400 nodes • Node bank has 2.000 managed nodes • No history messages
Java GUI options	• Different OVO/UNIX user accounts • *show all* messages • refresh interval *5 seconds* [1]

[1] The refresh interval has an impact on the outcome of the bulk transfer of messages to the Java GUI, i.e. 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.

4.1.1.3 Results

Figure 1 shows the message throughput based on the number of running Java GUIs, the message generator is an OVO DCE agent.

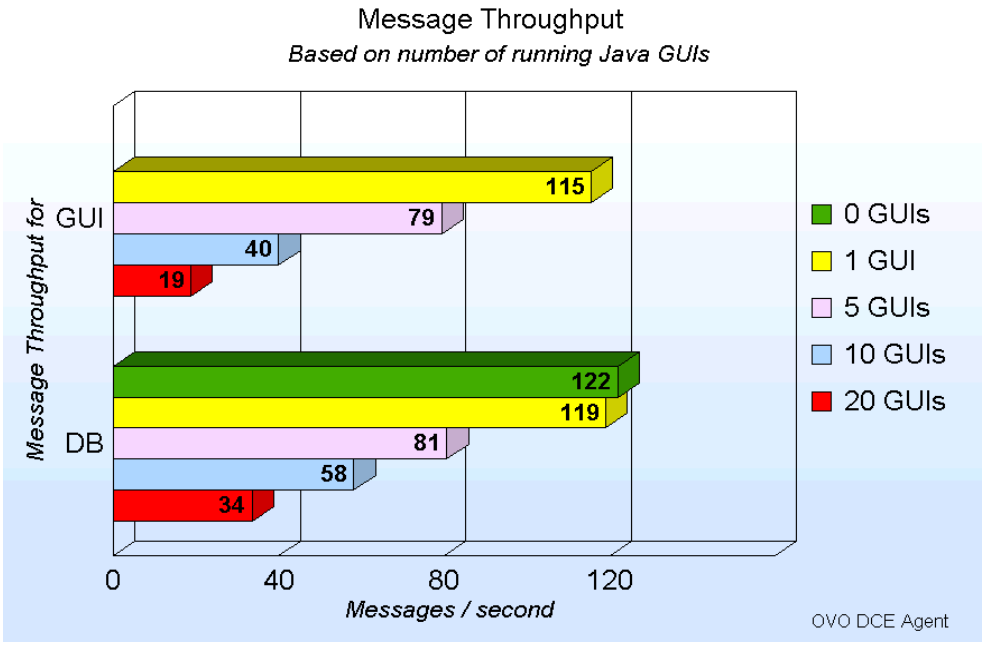


Figure 1: Message Throughput - Number of Java GUI's (DCE agent)

The same metrics are shown in Figure 2 – the message generator is now an OVO HTTPS agent.

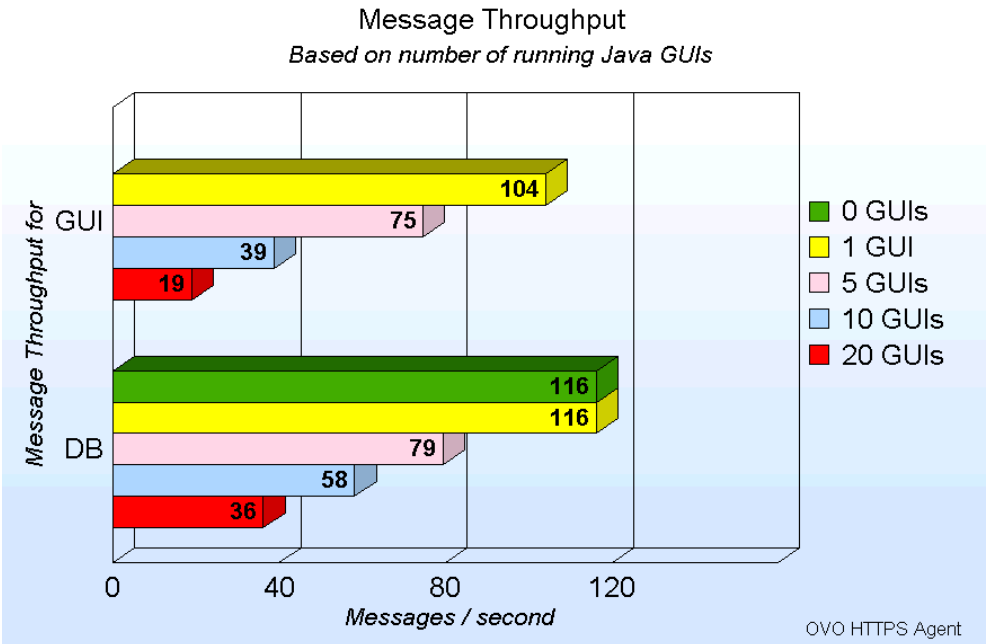


Figure 2: Message Throughput - Number of Java GUIs (HTTPS agent)

4.1.1.4 Interpretation

The following facts may be derived from these results:

- The message throughput decreases linear as more Java GUI's are started and free CPUs are available to server the associated GUI processes. If more GUIs are started and there are no free CPUs anymore, the message throughput will drop faster.
- Message throughput regarding the arrival in the OVO/UNIX database is higher than the throughput regarding the display of messages in the Java message browser.
- The impact of the agent type sending the messages may be neglected. Almost the same throughput is achieved for HTTPS and DCE agent.

4.1.1.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- Only the number of Java GUI's needed at the moment should be started since the OVO/UNIX database message throughput decreases with an increasing number of Java GUI's started.
- Whether a HTTPS or DCE agent is used has almost no impact on the message throughput if multiple Java GUIs are started.

4.1.2 Test: Message duplicates suppression

4.1.2.1 Synopsis

The following metrics are computed in this test.

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

4.1.2.2 Scenario

Measured Value	• Time until the last expected message is shown in the Admin GUI message browser
Message Generator	• 5.000 messages, severity normal, all messages were different (message text contained a unique number) • Messages are targeted to 100 nodes in 4 node groups, i.e. 50 messages per node • Operators used are responsible for 400 nodes • Node bank has 2.000 managed nodes • No history messages
Java GUI options	• No Java GUIs were started
Miscellaneous	• The feature “Add duplications as annotations” was turned off

In the previous performance tests, modifying the number of message keys showed very little impact on the computed metric values, thus for the current tests it has been decided to use only zero and one different metric key.

In addition, there were no significant differences between using an OVO DCE agent vs. using an OVO HTTPS agent as the message generator.

4.1.2.3 Results

The following table lists the message throughput in messages per second.

Legend:

OFF	OVO/UNIX feature *Message Duplicates Counter* was turned **off**
0	*Message Duplicates Counter* turned **on**, no message keys used
1	*Message Duplicates Counter* turned **on**, same key for all messages

Message Keys	OFF	0	1
Msgs/sec	122	46	122

Table 2: Message Throughput - Duplicates Counter

4.1.2.4 Interpretation

The following facts may be derived from these results:

- The loss of performance if message duplicates are counted is small if message keys are used.
- The number of different message keys has a small impact on the performance (the more message keys, the longer the time to process the messages) [Previous tests].
- If duplicates are to be counted without using message keys, the processing time is approx. 3 times of the time needed when using message keys.

4.1.2.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- If message duplicate suppression is enabled, message keys should be attached to the messages – even if every message gets its own key.

4.1.3 Test: Diverting messages to MSI programs

4.1.3.1 Synopsis

The following metrics are computed in this test.

Metric	• Number of messages per second (message throughput) until all messages are arrived in the Admin GUI via the *message stream interface* (MSI)
Parameter	• Number of MSI programs

4.1.3.2 Scenario

Measured Value	• Time until the last expected message is shown in the Admin GUI message browser
Message Generator	• 5.000 messages, severity normal • Messages are targeted to 100 nodes in 4 node groups, i.e. 50 messages per node • Operators used are responsible for 400 nodes • Node bank has 2.000 managed nodes • No history messages
Java GUI options	• No Java GUIs were started

Miscellaneous	• All messages were diverted to one or more MSI programs using the agent MSI. The MSI programs simply wrote the unmodified message back to the message stream. • Because all MSI programs got the same set of messages and each MSI program wrote its messages back to the MSI stream, a multiple of the initial 5.000 messages was seen in the message browser (5.000 messages per MSI program).

4.1.3.3 Results

The following table lists the results of this test.

MSI Progs	None	Agent 1xMSI	Agent 2xMSI	Agent 5xMSI
Messages	5.000	5.000	10.000	25.000
Throughput [Msg/s]	119	119	119	119

Table 3: Message Throughput – MSI (DCE agent)

MSI Programs	None	Agent 1xMSI	Agent 2xMSI	Agent 5xMSI
Messages	5.000	5.000	10.000	25.000
Throughput [Msg/s]	113	116	116	116

Table 4: Message Throughput – MSI (HTTPS agent)

Legend:

MSI Progs: None	No MSI program was active, messages were processed directly without diverting
MSI Progs: Agent *n*xMSI	*n* MSI programs were active using the *agent* message stream interface; thus, *n* times 5.000 messages were actually stored in the OVO database.

4.1.3.4 Interpretation

The following facts may be derived from these results:

- Using the *agent* message stream interface to feed OVO/UNIX with *diverted* messages results in no performance penalty if the messages are sent from a DCE agent.
- If a HTTPS agent is used as the message generator, the message throughput is slightly lower than for the DCE agent (3%). In addition, if more MSI programs are used the message throughput for the DCE sender is more stable than for the HTTPS sender (HTTPS: 2.5% difference).
- Registering more MSI programs did not have a significant impact on the message throughput.

4.1.3.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- Using the message stream interface results in no performance loss.

4.1.4 Test: Diverting messages to ECS circuits

4.1.4.1 Synopsis

The following metrics are computed in this test.

Metric	• Number of messages per second (message throughput) until all messages are arrived in the Admin GUI via the *event correlation system (ECS)*
Parameter	• OVO DCE agent vs. OVO HTTPS agent

4.1.4.2 Scenario

Measured Value	• Time until the last expected message is shown in the Admin GUI message browser
Message Generator	• 5.000 messages, severity normal • Messages are targeted to 100 nodes in 4 node groups, i.e. 50 messages per node • Operators used are responsible for 400 nodes • Node bank has 2.000 managed nodes • No history messages
Java GUI options	• No Java GUIs were started
Miscellaneous	• All messages were diverted to one ECS circuit running on the agent. The ECS circuit simply wrote the unmodified message back to the message stream.

4.1.4.3 Results

The following tables list the results of this test.

ECS circuits	0	1
Messages	5.000	5.000
Throughput [Msg/s]	119	116

Table 5: Message Throughput – ECS (DCE agent)

ECS circuits	0	1
Messages	5.000	5.000
Throughput [Msg/s]	116	119

Table 6: Message Throughput – ECS (HTTPS agent)

4.1.4.4 Interpretation

The following facts may be derived from these results:

- Using the *agent* ECS interface to feed OVO/UNIX with *diverted* messages results in no performance penalty.
- The agent type (DCE vs. HTTPS) had no significant impact on the message throughput (3% difference).

4.1.4.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- Using the ECS interface results in no performance loss.

4.1.5 Test: HTTPS vs. DCE agent as message sender

4.1.5.1 Synopsis

The following metrics are computed in this test.

Metric	• Number of messages in the OVO message manager queue on the OVO server over time in case a large number of messages is generated on the managed nodes
Parameter	• Number of OVO agents • OVO DCE agent vs. OVO HTTPS agent • Platform of OVO agents

4.1.5.2 Scenario

Measured Value	• Number of messages in the OVO message manager queue over time until the queue is empty
Message Generator	• 100.000 messages, severity normal • Messages are targeted to 100 nodes in 4 node groups, i.e. 50 messages per node • Operators used are responsible for 400 nodes • Node bank has 2.000 managed nodes • No history messages
Java GUI options	• No Java GUIs were started
Miscellaneous	• n/a

4.1.5.3 Results

The graph for the number of elements in the queue of process *opcmsgm* over time (in seconds) beginning from the start of the message generator is shown in the following diagrams.

The x-axis shows the time in seconds, the left y-axis shows the number of messages in the queue and the right y-axis shows the message change rate in messages per second.

Legend:

#Items	The number of messages for this point of time
Rate	The message rate in messages per second computed using the current and the previous data point

Figure 3 shows the queue length for HP PA agents. One test was done with one DCE agent sending 100.000 messages, the next text was done with one HTTPS agent sending 100.000 messages, and the last test was done using one DCE and one HTTPS agent sending 50.000 messages each.

The left y-axis shows the total number of items in the OVO server message manager queue over time, the right y-axis shows the change rate of items in the message manager queue over time, in items per second.

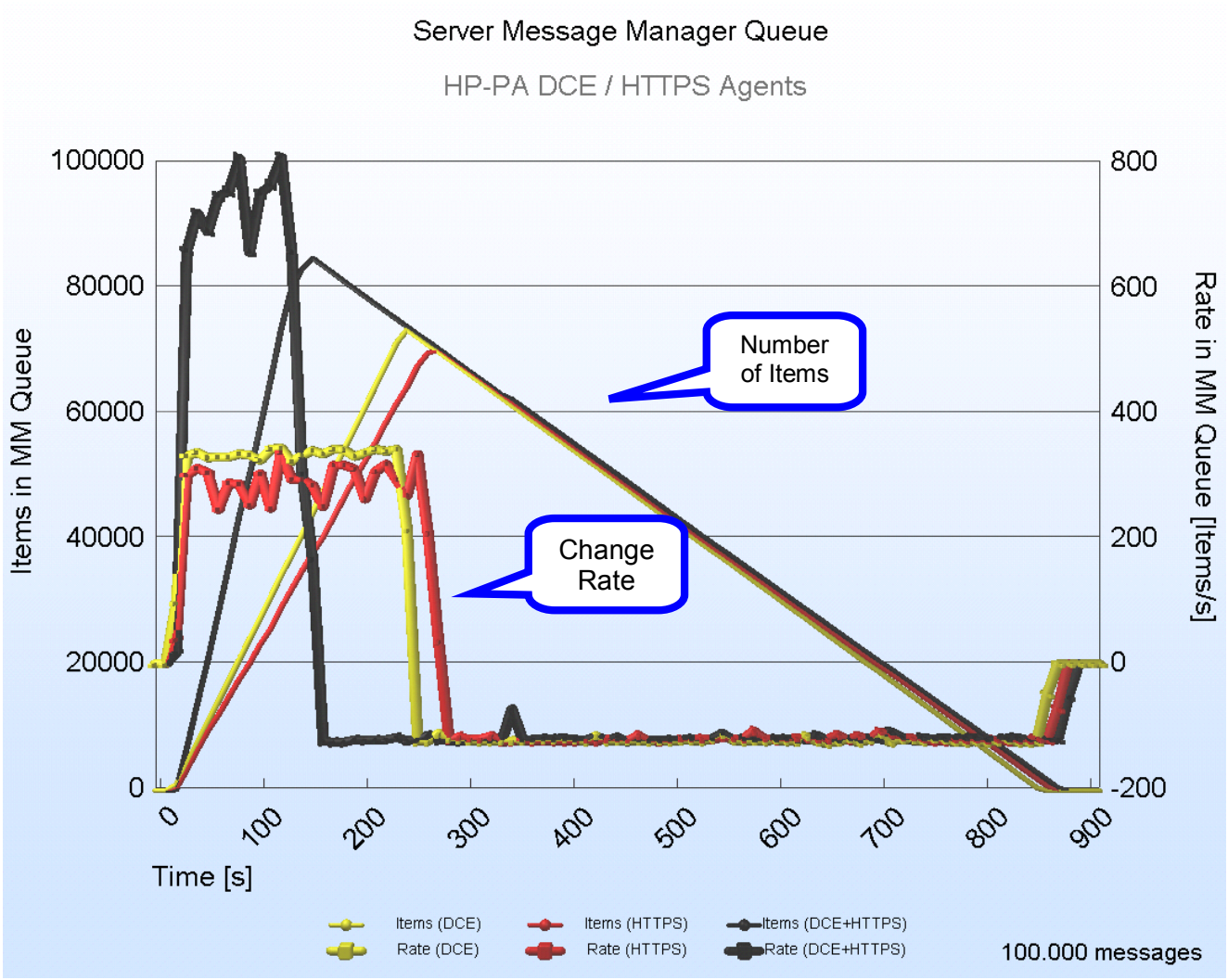


Figure 3: Message Manager queue length: HP PA DCE + HTTPS agents

Interpretation

- The rate by which the messages are processed once they are in the message manager queue is independent of the agent type.
- The DCE agent is able to send messages faster than the HTTPS agent.
- If both a HTTPS agent and a DCE agent are sending messages, the message reception rate in the server is more than the sum of the individual scenario (DCE sending alone and HTTPS sending alone).
- The DCE agent is able to send messages at a constant rate, where the HTTPS agent rate is not constant. The rate of the single HTTPS agent does not exceed the rate of the single DCE agent.

The next Figure 4 shows the queue length on the server in case HP Itanium agents are used. One test was done with one DCE agent running on a HP Itanium system sending 100.000 messages; the next test was done with once HTTPS agent running on a HP Itanium system sending 100.000 messages.

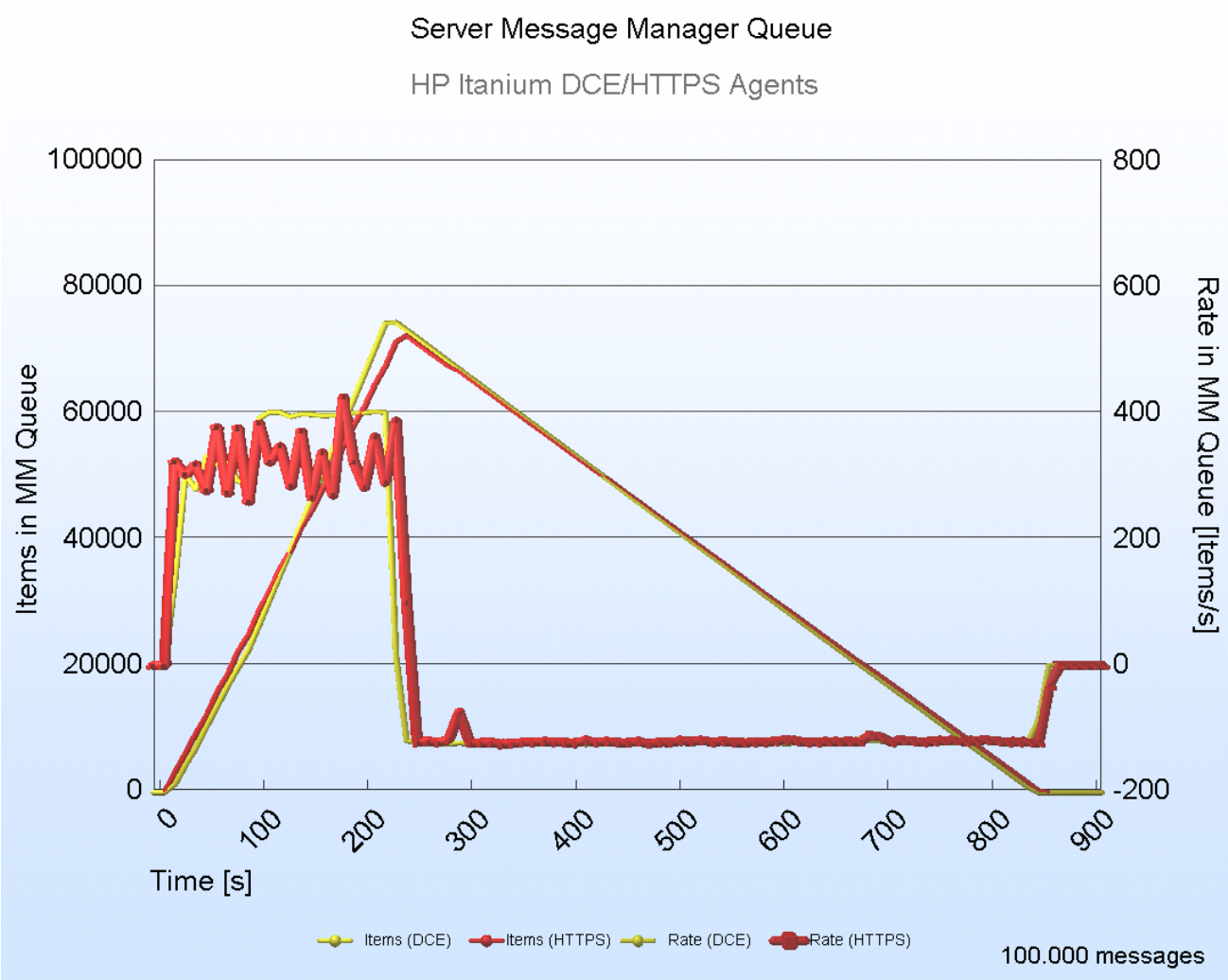


Figure 4: Message Manager queue length: HP Itanium DCE + HTTPS agents

Interpretation

- The rate by which the messages are processed once they are in the message manager queue is independent of the agent type.
- The DCE agent is able to send messages faster than the HTTPS agent.
- The DCE agent is able to send messages at a constant rate, where the HTTPS agent rate is not constant. The rate of the single HTTPS agent does not exceed the rate of the single DCE agent.
- The HP Itanium agents are able to send their messages faster than the HP PA agents (14%).

Figure 5 shows the queue length on the server in case Linux agents are used. One test was done with one DCE agent running on a Linux system sending 100.000 messages; the next test was done with once HTTPS agent running on a Linux system sending 100.000 messages.

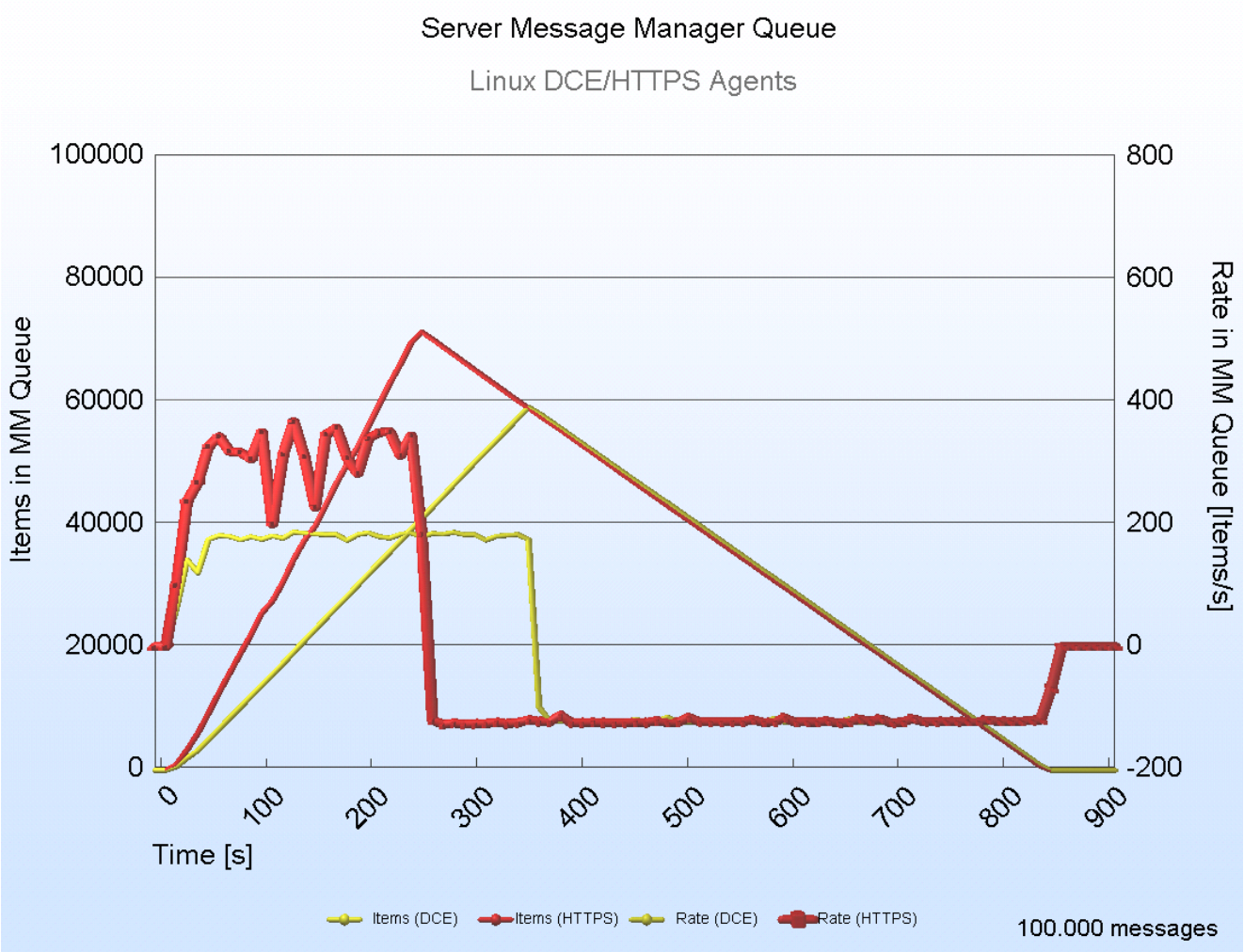


Figure 5: Message Manager queue length: Linux DCE + HTTPS agents

Interpretation

- The rate by which the messages are processed once they are in the message manager queue is independent of the agent type.
- The HTTPS agent is able to send messages faster than the DCE agent (66%).
- The DCE agent is able to send messages at a constant rate, where the HTTPS agent rate is not constant. The rate of the single HTTPS agent does not exceed the rate of the single DCE agent.
- The Linux HTTPS agent is able to send messages nearly as fast as HP-PA and HP Itanium agents. The Linux DCE agent is significantly slower.

Figure 6 shows the queue length on the server where one HP PA HTTPS agent was sending 100.000 messages. One test was performed using the default OVO parameters, for the other test, the OVO parameter `MAX_MESSAGE_BULK` was set to the value 1, which effectively turns the bulk mode off. This value controls the number of messages which are authenticated together.

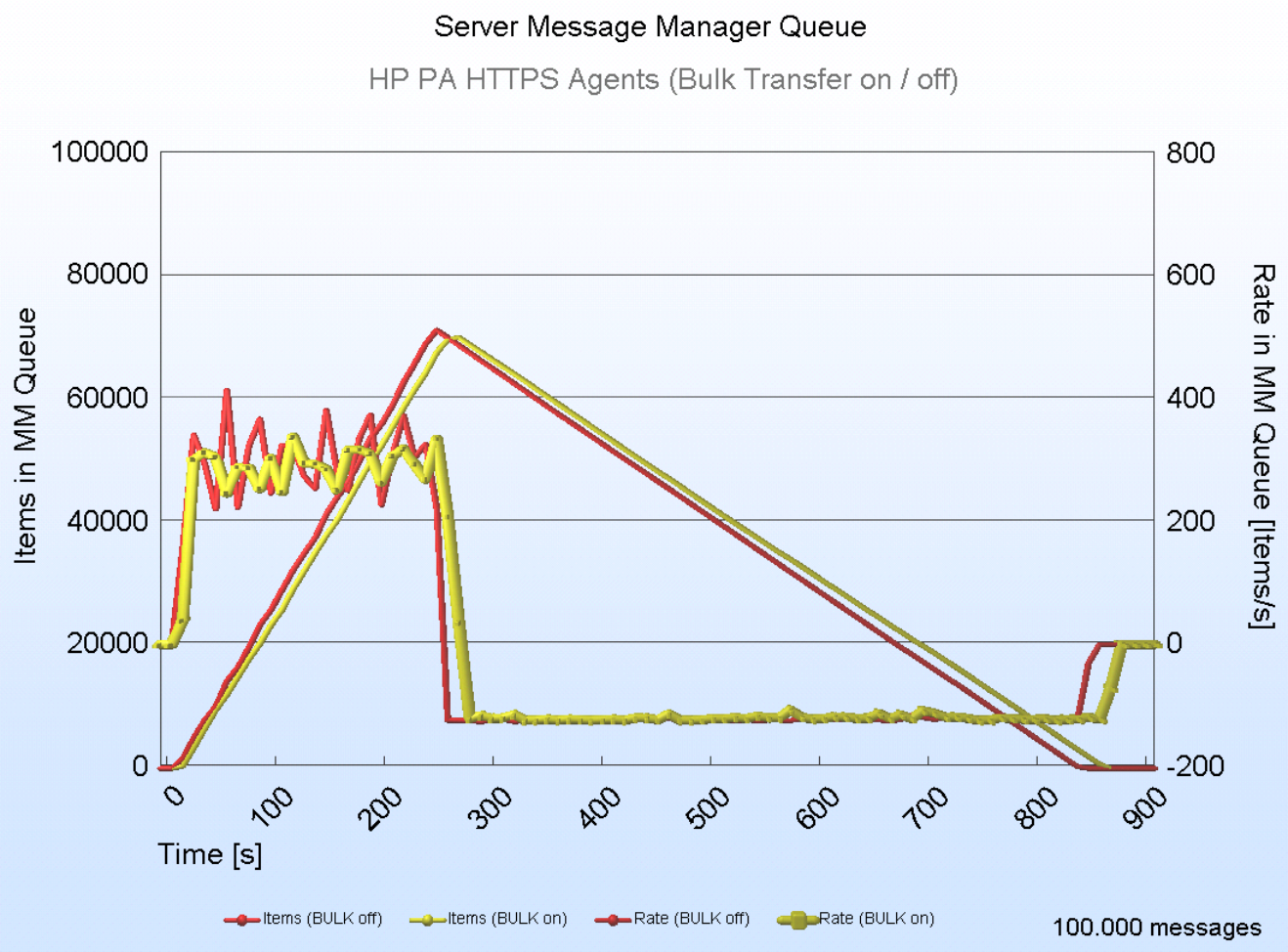


Figure 6: Message manager queue length: HP PA HTTPS agents (BULK on/off)

Interpretation

- The parameter MAX_MESSAGE_BULK had almost no impact on the message throughput.

4.1.5.4 Interpretation

The following facts may be derived from these results:

- There is no significant difference in the message sending rate for HTTPS and DCE agents – besides Linux. For Linux, the DCE agent is significantly slower than the HTTPS agent.
- If HTTPS and DCE agent were both sending messages, the message reception rate was higher than in case the single agents were sending messages.
- More message manager processes were able to process messages faster than a single process.

4.1.5.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- Use HTTPS and DCE agents at the same time ...
- If message bursts are not avoidable, ensure that the time of *silence* is long enough to process all queued messages.

In Table 7, the *sending rate* of HP PA agents are listed. This is the number of messages per second the agent is able to send to the OVO server. As can be seen from the table, there is no real significant difference between the agent types.

Agent Type	DCE	HTTPS
Sending Rate [Msgs/s]	450	410

Table 7: Message sending rate of HP PA agents

4.2 Java GUI Startup Time

4.2.1 Test: Varying the number of managed nodes and messages

4.2.1.1 Synopsis

The following metrics are computed in this test.

Metric	• Startup time of two Java GUIs • Startup time of two Motif GUIs
Parameter	• Number of managed nodes the users are responsible for • Number of active messages

4.2.1.2 Scenario

Measured Value	• Time until the last GUI of a type 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 4 node groups, i.e. 50 and 500 messages per node • Operators used are responsible for 500 and 2.000 nodes • Node bank has 10.000 managed nodes • No history messages
Java GUI options	• Different OVO/UNIX user accounts • *show all* messages • refresh interval *5 seconds*

4.2.1.3 Results

The Table 8 lists the results of this test: the time in seconds needed to start two Java and two Motif GUIs with different OVO/UNIX user accounts. The tests for the different GUI types were done separately.

For the Motif GUIs, the startup time for the 2nd start was taken, thus the time for building the map cache was not included.

Responsible Nodes / Active Messages	500 Motif	500 Java	2.000 Motif	2.000 Java
5.000	45	13	155	15
50.000	57	60	167	61

Table 8: GUI Startup Time - Managed Nodes / Active Messages

4.2.1.4 Interpretation

Figure 7 shows the bar charts for the tests, with a varying number of managed nodes the OVO/UNIX user is responsible for and a varying number of active messages.

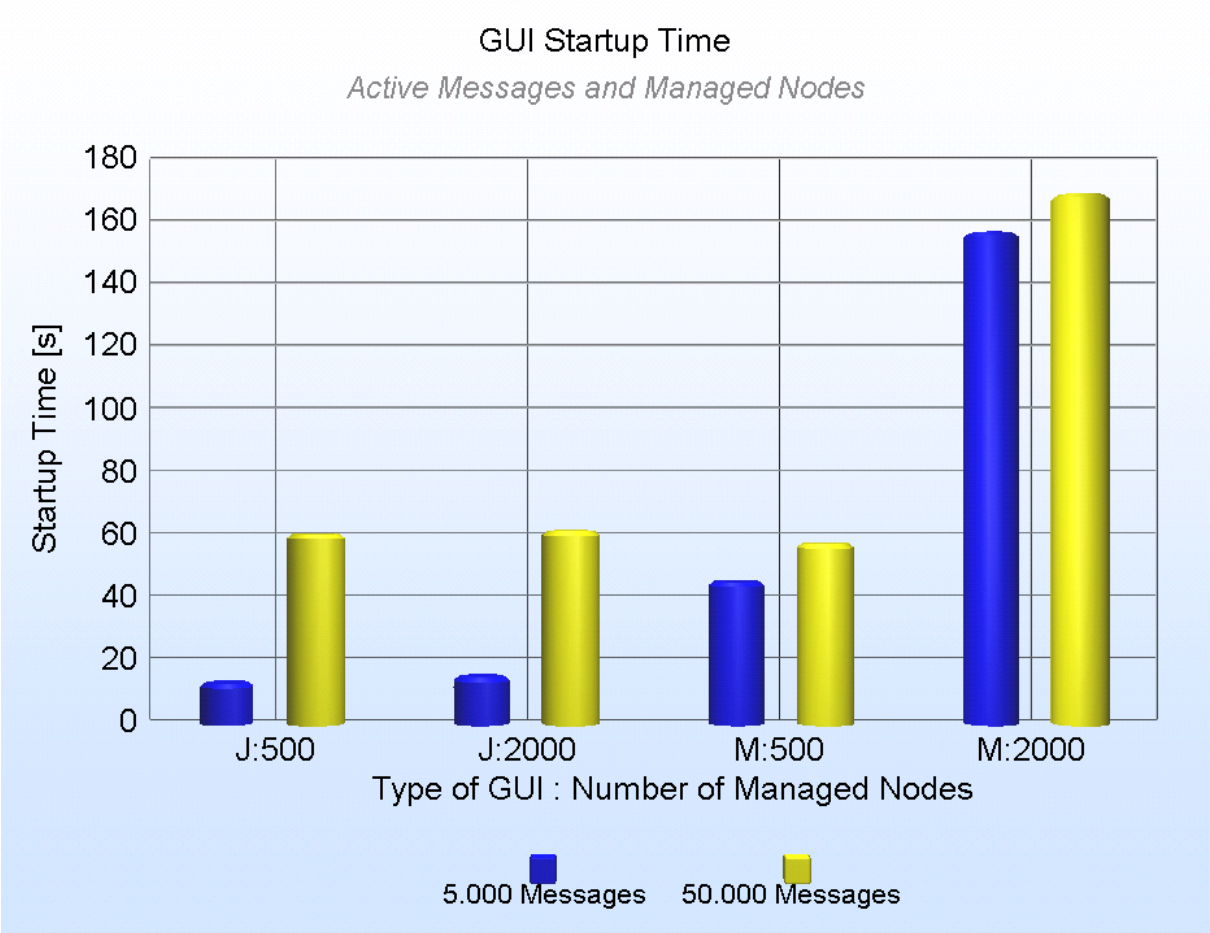


Figure 7: Java GUI Startup Time - Managed Nodes and Active Messages

The following facts may be derived from these results:

- The number of managed nodes in the OVO/UNIX user's realm has a significant impact on the startup time of the Motif GUI. There is 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.

4.2.1.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- Up to 2.000 managed nodes and 50.000 active messages, the startup times of the Java GUI are approx. 60 seconds. Thus, too many active messages should be prevented (use duplicate suppression, event correlation, etc.).
- The number of managed nodes in the OVO users realm should be kept as small as possible, if the Motif operator GUI is used

4.2.2 Test: Varying the number of CMA's

4.2.2.1 Synopsis

The following metrics are computed in this test.

Metric	• Startup time of two Java GUIs displaying all CMAs attached to the OVO messages.
Parameter	• Number of CMAs attached to the messages • Number of active and history messages

4.2.2.2 Scenario

Measured Value	• The time until the last Java GUI is up and functional, i.e. 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	• 1.000 and 10.000 messages, severity normal, were 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 4 node groups, i.e. 10 and 100 messages per node • Operators used are responsible for 500 nodes • Node bank has 10.000 managed nodes • No history messages for the first test • No active messages for the second test
Java GUI options	• Different OVO/UNIX user accounts • *show all* messages • refresh interval *5 seconds*

4.2.2.3 Results

Table 9 lists the results of this test: the time in seconds needed to start 2 Java GUI's with different OVO/UNIX user accounts, with a varying number of active messages (and CMAs) and no history messages.

Number of CMA's Active messages	0	1	5	10	50
1.000	9	9	11	12	16
10.000	18	21	25	32	84

Table 9: Java GUI Startup Time - CMA's in active messages

Table 10 lists the result where no active messages were present, but a varying number of history messages.

Number of CMA's History messages	0	1	5	10	50
1.000	9	8	9	9	9
10.000	9	9	9	9	9

Table 10: Java GUI Startup Time - CMA's in history messages

4.2.2.4 Interpretation

Figure 8 shows a visualization of the results of this test. No graph is shown for the test with history messages only.

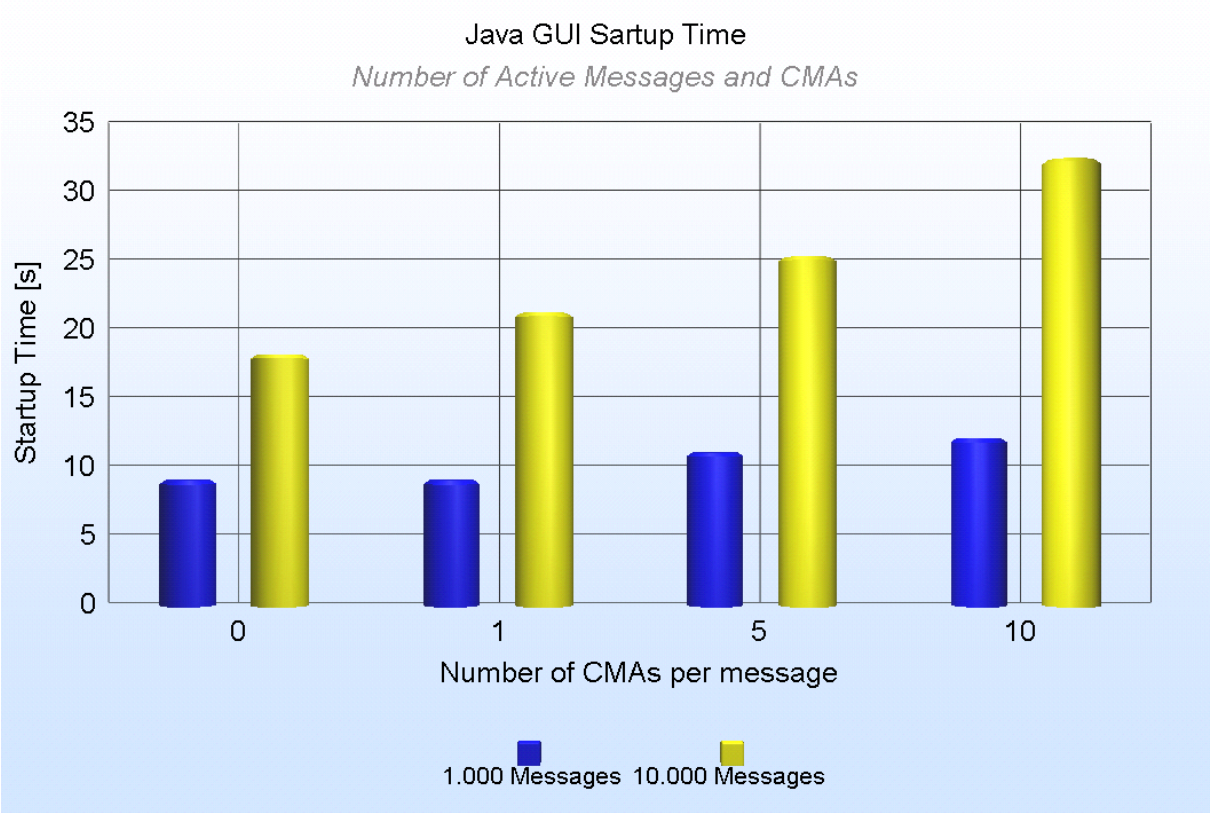


Figure 8: Java GUI Startup Time - CMA's

The following facts may be derived from these results:

- The number of customer message attributes per message has an impact on the startup time of the Java GUI – although this impact is very small.
- The delay in the startup time of the Java GUI depends on the number of CMA's per message in a linear function.

4.2.2.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- Using CMA's imposes little penalty on the startup time of the Java GUI.

4.3 Message Acknowledgement

4.3.1 Test: Varying the number of messages and acknowledge type

4.3.1.1 Synopsis

The following metrics are computed in this test.

Metric	• This test measures the time needed to acknowledge multiple messages using the Java GUI.
Parameter	• The number of Java GUIs • The number of acknowledged messages in one operation • The internal handling of the acknowledgement (messages are moved vs. messages are marked)

4.3.1.2 Scenario

Measured Value	• The time needed to acknowledge multiple messages using the Java GUI. Messages were acknowledged on one Java GUI, the time until the last message disappeared on all started Java GUI's was taken as the result of this test.
Message Generator	• 50.000 messages, severity normal, were generated before the test started. No messages were generated during the acknowledgement. • 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 4 node groups, i.e. 500 messages per node • Operators used are responsible for 500 nodes • Node bank has 10.000 managed nodes • No history messages did exist prior to the test

Java GUI options	• Different OVO/UNIX user accounts – but same responsibility • *show all* messages • refresh interval *5 seconds*
Environment	• For the first part of the test, acknowledged messages were moved from the active to the history database table. Since the acknowledgement was started from the Java GUI, and the Java GUI acknowledges only single messages, each acknowledged message is moved from the act to the history table this way if the standard OVO/UNIX parameters are used. • For the second part of the test, acknowledged messages were only marked as acknowledged in the active message database table. If the OVO/UNIX parameter `OPC_DIRECT_ACKN_LIMIT` is set to `0` using the `ovconfchg`[1] tool, then even if the single acknowledgements are started from the Java GUI, the messages are only marked, not moved. • For both tests, the OVO/UNIX message mover process was disabled (OVO/UNIX parameter `OPC_ACK_MOVE_INTERVAL 0` using the ovconfchg[2,3,4] tool.

4.3.1.3 Results

The following Table 11 lists the results of this test. For this first part of the test, the acknowledged messages were moved from the active messages table to the history messages table. Shown is the time in seconds until the last message disappeared from the last Java GUI message browser.

Number of ackn. Msgs / Number of Java GUI's	1.000	5.000	10.000
1	31	126	316
5	28	153	370

Table 11: Message Acknowledgement – Moved Messages

[1] The parameter was set with: `ovconfchg -ovrg server -ns opc -set OPC_DIRECT_ACKN_LIMIT 0`
[2] The parameter was set with: `ovconfchg -ovrg server -ns opc -set OPC_ACK_MOVE_INTERVAL 0`
[3] In a production environment where acknowledged messages are downloaded quickly on a regular basis (e.g. every 24 hours), the message mover process could be disabled at all. Thus, all acknowledged messages would only be marked and never moved in the OVO database tables until they are finally downloaded.
[4] See "*OVO message acknowledge internals*" on page 65 for more details.

The following Table 12 lists the results of the second part of this test. In the second part, the messages were only marked, not moved.

Number of ackn. Msgs	1.000	5.000	10.000
Number of Java GUI's			
1	18	72	144
5	22	94	187

Table 12: Message Acknowledgement – Marked Messages

Upon special request, we repeated the marked message test with the Motif GUI. Table 13 shows the results for this test.

Number of ackn. Msgs	1.000	5.000	10.000
Number of Motif GUI's			
1	2	10	20
5	2	10	20

Table 13: Message Acknowledgement – Marked Messages (Motif GUI)

4.3.1.4 Interpretation

Figure 9 visualizes the times needed for one Java GUI.

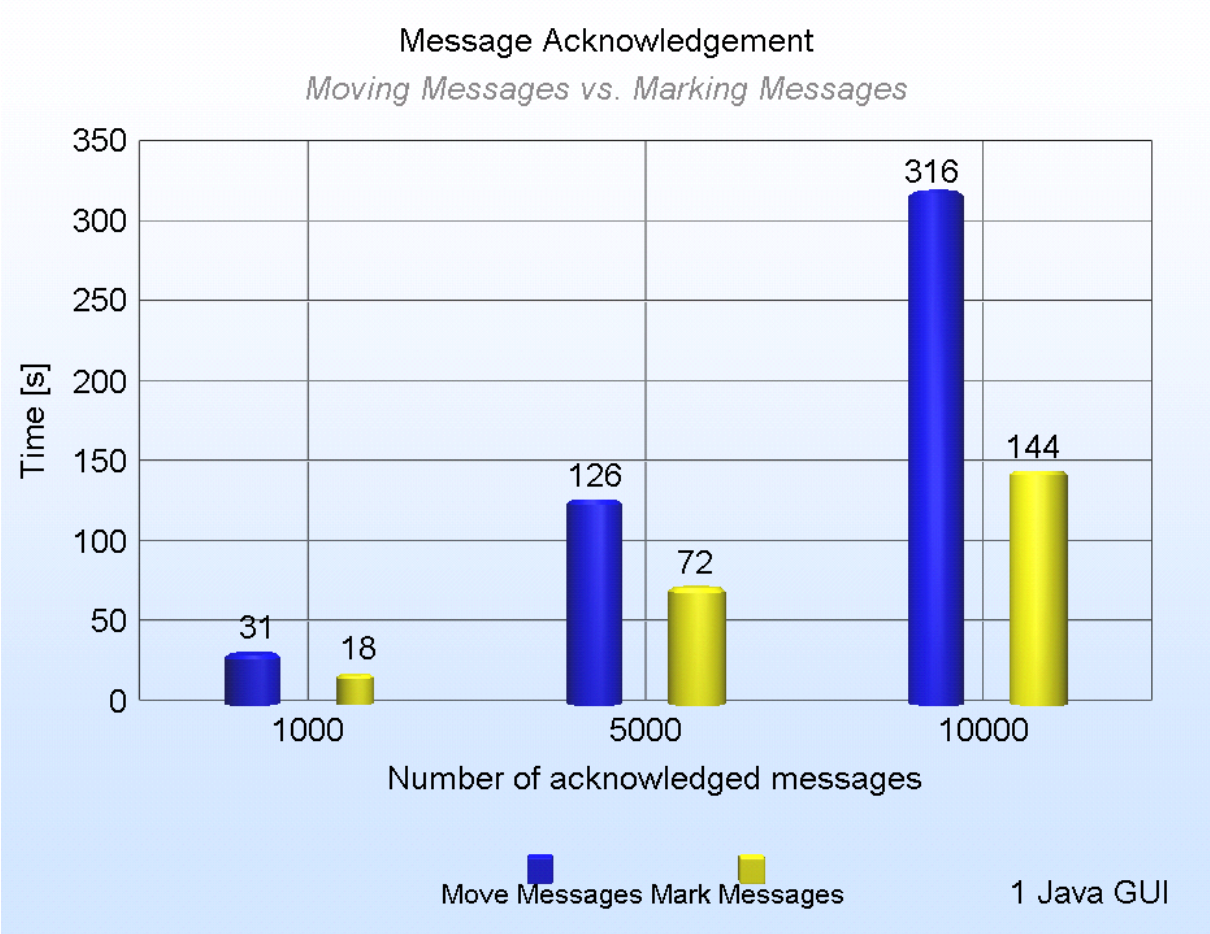


Figure 9: Message Acknowledgement – Time for "Move" and "Mark"

The following facts may be derived from these results:

- Marking the messages on acknowledgement is faster than moving the acknowledged messages from the active to the history database table.
- The time loss when messages are moved between the database tables is constant (7 – 9 seconds per 1.000 messages).
- Message acknowledgement takes longer if more Java GUI's are displaying those messages.

4.3.1.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- For optimizing the usability of the Java GUI, acknowledged messages should be marked only, and later – automatically – moved to the history table.

4.4 Service Navigator Startup Time

For these tests, the shape and the size of the service tree is varied.

We used **trees** with the following attributes:

- Varying depth and width
- No *Use Factor* or *Links*, i.e. associations from services to their *nephews* (children of siblings)

The number of services at level k is $L_k = m^k$ where $k = 0$ for the root level.

The total number of services in the tree: $\frac{m^{n+1} - 1}{m - 1}$ where

- *m* is the breadth (number of children for each service)
- *n* is the depth (number of levels below the top level; see Figure 10)

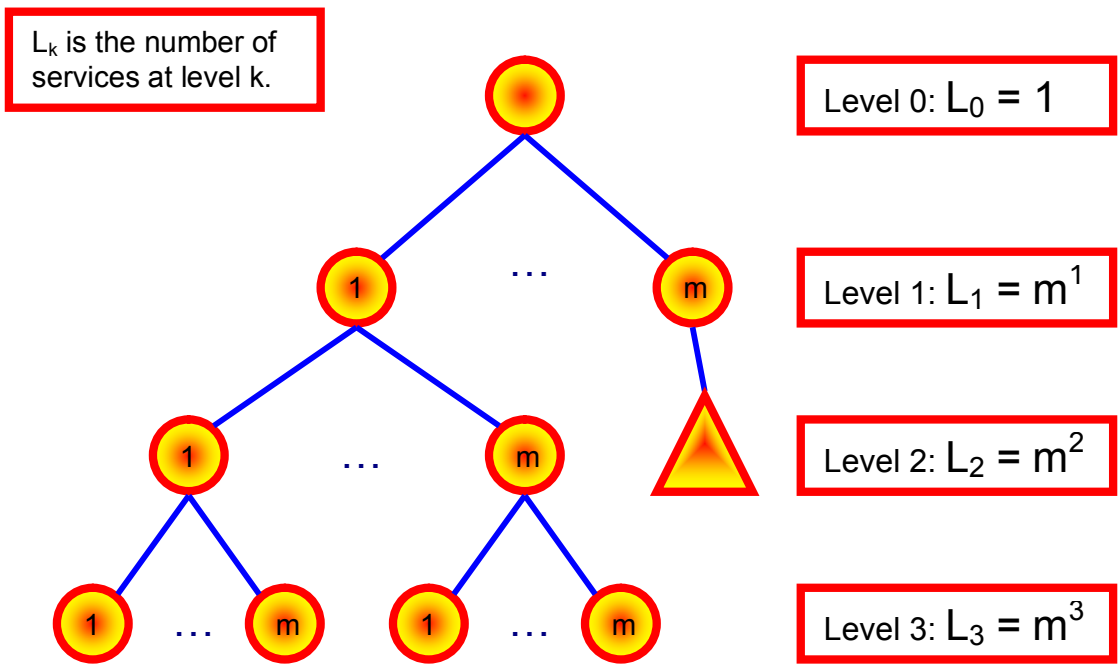


Figure 10 : The service tree model

4.4.1 Test: Varying the shape and size of the service tree

4.4.1.1 Synopsis

The following metrics are computed in this test.

Metric	• The time needed to start one and two Java GUIs for different OVO/UNIX users which have the same service tree assigned.
Parameter	• The number of levels in the service tree (depth). • The number of children for each intermediate service in the tree (breadth). • The feature “Service Load on Demand” was turned off and turned on.

4.4.1.2 Scenario

Measured Value	• This test measures the startup time of the Java GUI in situations where a service tree is assigned to the OVO/UNIX user starting the GUI – thus the “startup time of the service navigator”.
Message Generator	• No messages were generated and used for this test.
Java GUI options	• Different OVO/UNIX user accounts – but same service tree

4.4.1.3 Results

The Table 14 lists the total number of services in the tree, depending on the number of levels and the breadth of the tree. Configurations shown in orange (B/D = 10/10, B/D = 40/5, B/D = 40/10) were not tested, the configurations in yellow (B/D = 3/10, B/D = 40/3) are used to compare the impact of the shape of the tree for two trees with a nearly identical total number of services.

Depth of tree / **Breadth of tree**	**3**	**5**	**10**
3	40	364	88573
10	1111	111111	1,1111E+10
40	65641	105025641	1,0755E+16

Table 14: Service Navigator - Number of services in tree

Table 15 and Table 16 show the times needed to start the Java GUI's with "Service Load on Demand" not in effect.

Depth of tree	3	5	10
Breadth of tree			
3	8	9	286
10	12	273	
40	60		

Table 15: Service Navigator – Startup Times with no SLOD – 1 Java GUI

Depth of tree	3	5	10
Breadth of tree			
3	8	10	472
10	14	458	
40	97		

Table 16: Service Navigator – Startup Times with no SLOD – 2 Java GUIs

With "Service Load on Demand" in effect, the startup time of the Java GUIs was constant: 8 seconds. See Table 17 and Table 18.

Depth of tree	3	5	10
Breadth of tree			
3	8	8	8
10	8	8	
40	8		

Table 17: Service Navigator – Startup Times with SLOD active – 1 Java GUI

Depth of tree	3	5	10
Breadth of tree			
3	8	8	8
10	8	8	
40	8		

Table 18: Service Navigator – Startup Times with SLOD active – 2 Java GUIs

4.4.1.4 Interpretation

In Figure 11, the result of this test with one running Java GUI is visualized.

The bars marked with the yellow arrow show the results of the configurations with a nearly identical total number of services, but with different number of levels and different breadth.

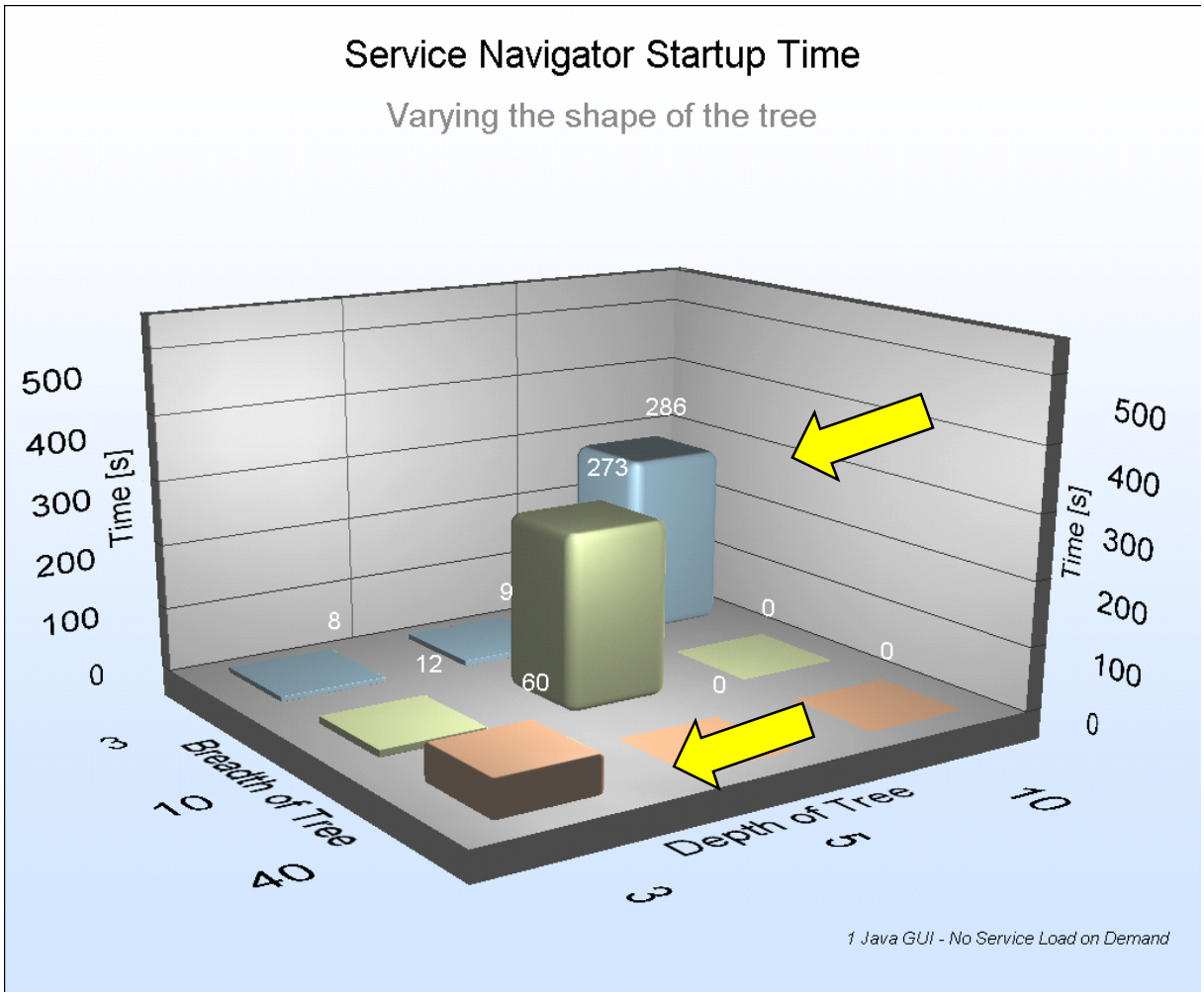


Figure 11: Service Navigator Startup Times – 1 Java GUI

In Figure 12, the result of this test with two running Java GUIs is visualized.

The bars marked with the yellow arrow show the results of the configurations with a nearly identical total number of services, but with different number of levels and different breadth.

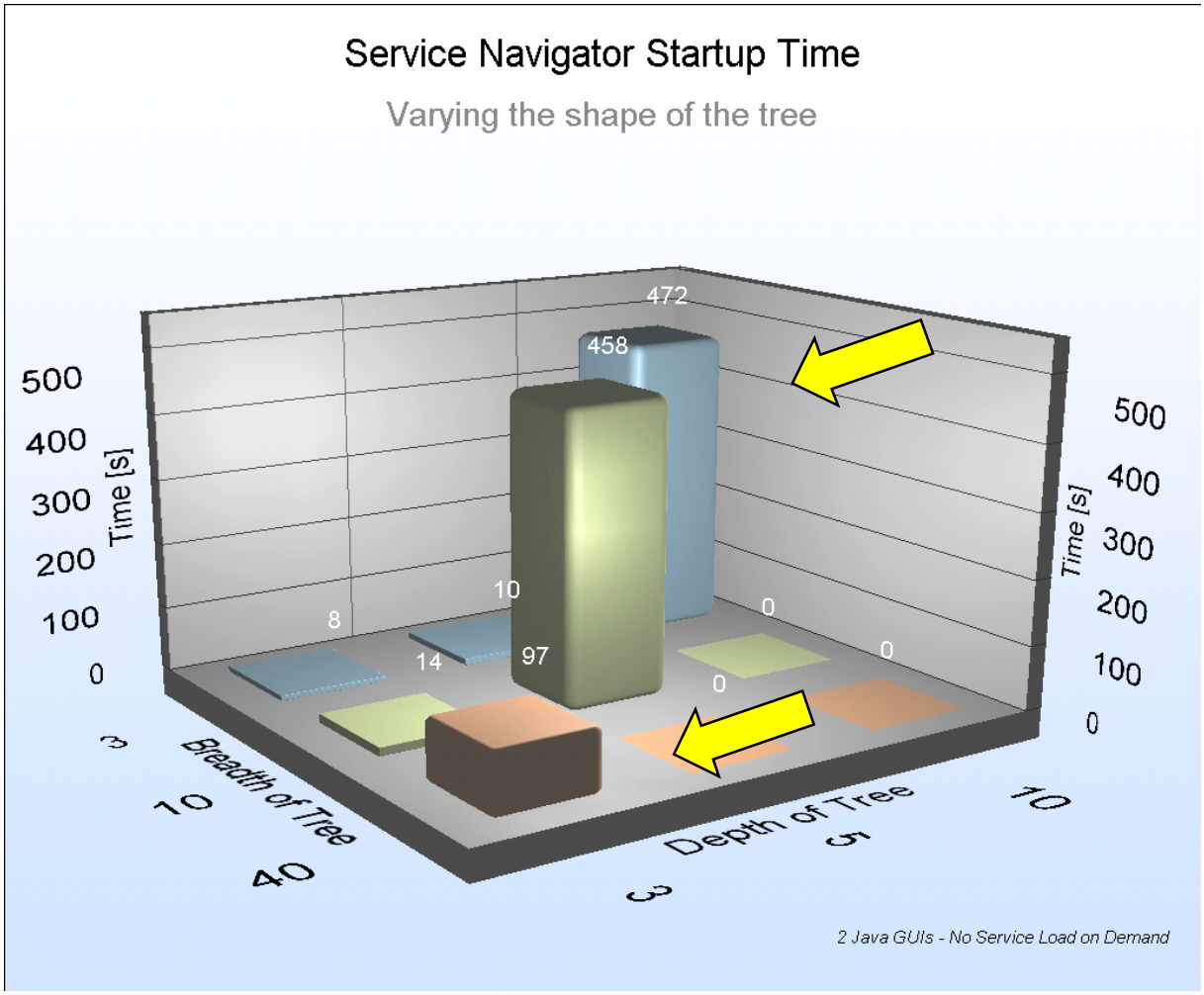


Figure 12: Service Navigator Startup Times – 2 Java GUIs

The following facts may be derived from these results:

- The feature “Service Load on Demand” works as expected: constant startup time for the Java GUI.
- The total number of services is not the only attribute which has an impact on the startup time. Instead, the number of the service levels - the depth of the tree – has the strongest impact on the startup time (see figures for depth 10, breadth 3 vs. depth 3, breadth 40).

4.4.1.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- If *deep* service trees are used – service trees with more than 5 levels – then the feature “Service Load on Demand” should be used to avoid long delays in the startup time of the Java GUI.

4.5 Performance of the History Message Filter

4.5.1 Test: Varying the search criteria

4.5.1.1 Synopsis

The following metrics are computed in this test.

Metric	• The time needed from starting the search in the GUI until the number of expected history messages was shown in the message browser was taken as the result of this test.

Parameter	• The number of Java GUIs
	• The number of history messages in the OVO database
	• The search criteria
	  ○ search for a node
	  ○ search for an object string
	  ○ search for a message pattern
	  ○ search for a CMA

4.5.1.2 Scenario

Measured Value	• This test measures the performance of the filter mechanism used to locate messages in the OVO/UNIX history messages database table, i.e. acknowledged messages. The time needed from starting the search until the number of expected messages was shown in the message browser was taken as the result of this test.
Message Generator	• 50.000 / 100.000 history messages of severity normal were generated before this test • No active messages did exist • No messages were generated during the test • All message had 10 CMAs, 20 characters per CMA • Operators user are responsible for 500 nodes • Node bank has 10.000 managed nodes

Java GUI options	• Different OVO/UNIX user accounts – but same responsibility • show last 50 messages • refresh interval 5 seconds
Environment	• If multiple Java GUIs were used for the test, the other GUIs (other than the one starting the timed search) were performing the “node not found” search. • Automatic download if history messages was disabled • Messages were moved to the opc_hist_messages table by opcdbmsgsmv (message mover process)

4.5.1.3 Results

Table 19 lists the results of this test with 50.000 history messages. Listed are times in seconds from the start of the search until the last expected message was shown in the message browsers. In addition, the numbers of expected messages are shown in parenthesis.

Search time [s]	Motif	1 Java GUI	5 Java GUIs
- Node: found (500)	1	5	5
- Node: not found (0)	25	133	314
- Object: found (500)	1	8	10
- Object: not found (0)	1	1	2
- Message pattern (500)	18	80	284
- Message pattern (0)	18	80	242
- CMA: found (500)	n/a	30	93
- CMA: not found (0)	n/a	4	10

Table 19: Performance of History Filter – 50.000 history messages

In Table 20, the results for 100.000 history messages are shown.

Search time [s]	Motif	1 Java GUI	5 Java GUIs
- Node: found (1000)	1	16	10
- Node: not found (0)	57	244	884
- Object: found (1000)	5	17	19
- Object: not found (0)	1	1	4
- Message pattern (1000)	98	179	713
- Message pattern (0)	40	183	690
- CMA: found (1000)	n/a	139	301
- CMA: not found (0)	n/a	13	39

Table 20: Performance of History Filter – 100.000 history messages

4.5.1.4 Interpretation

In Figure 13, the test results for 50.000 history messages are shown. The number of expected messages in the searches is 500.

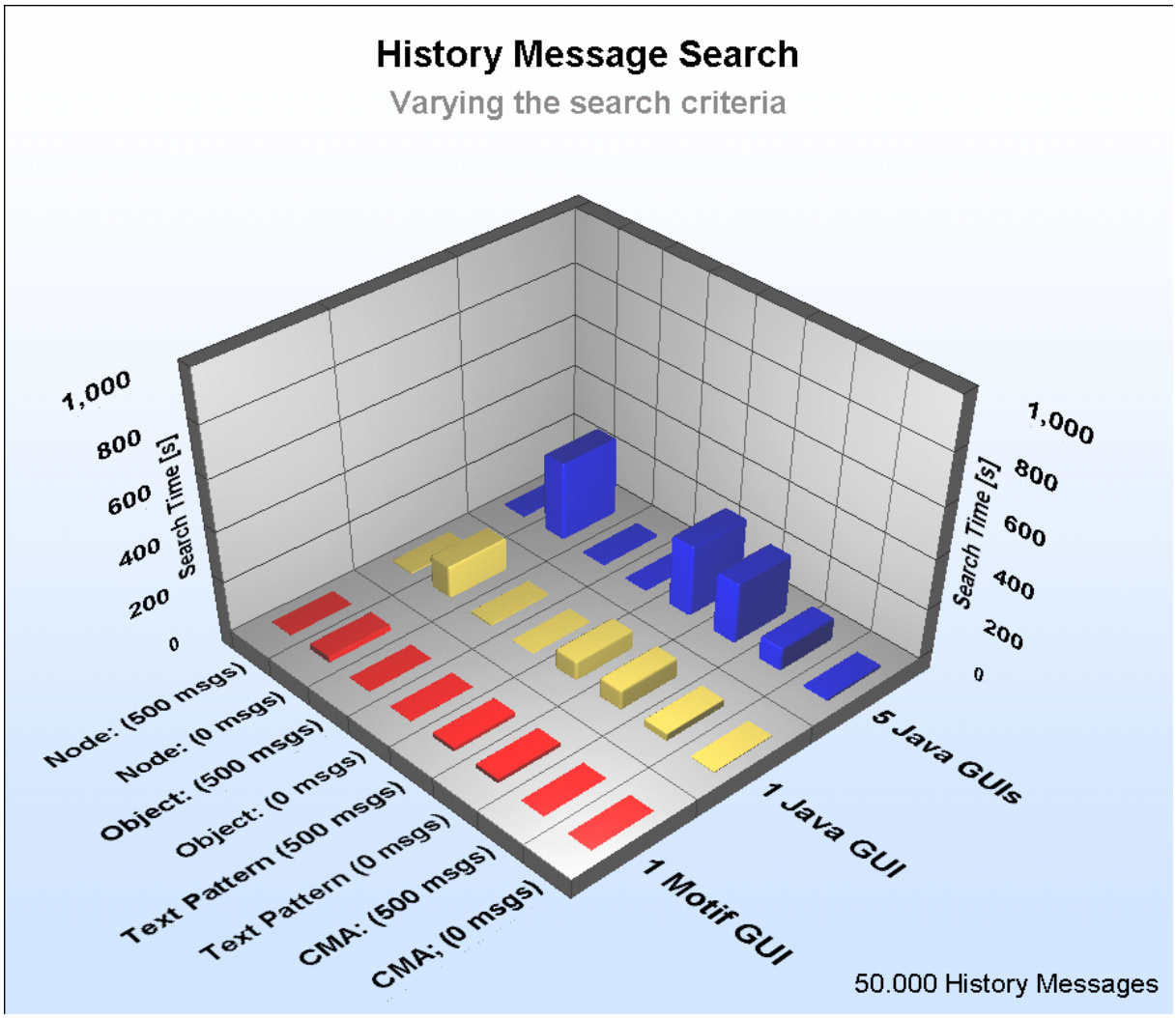


Figure 13: Performance of History Filter – 50.000 history messages

In Figure 14, the test results for 100.000 history messages are shown. The number of expected messages in the searches is 1.000.

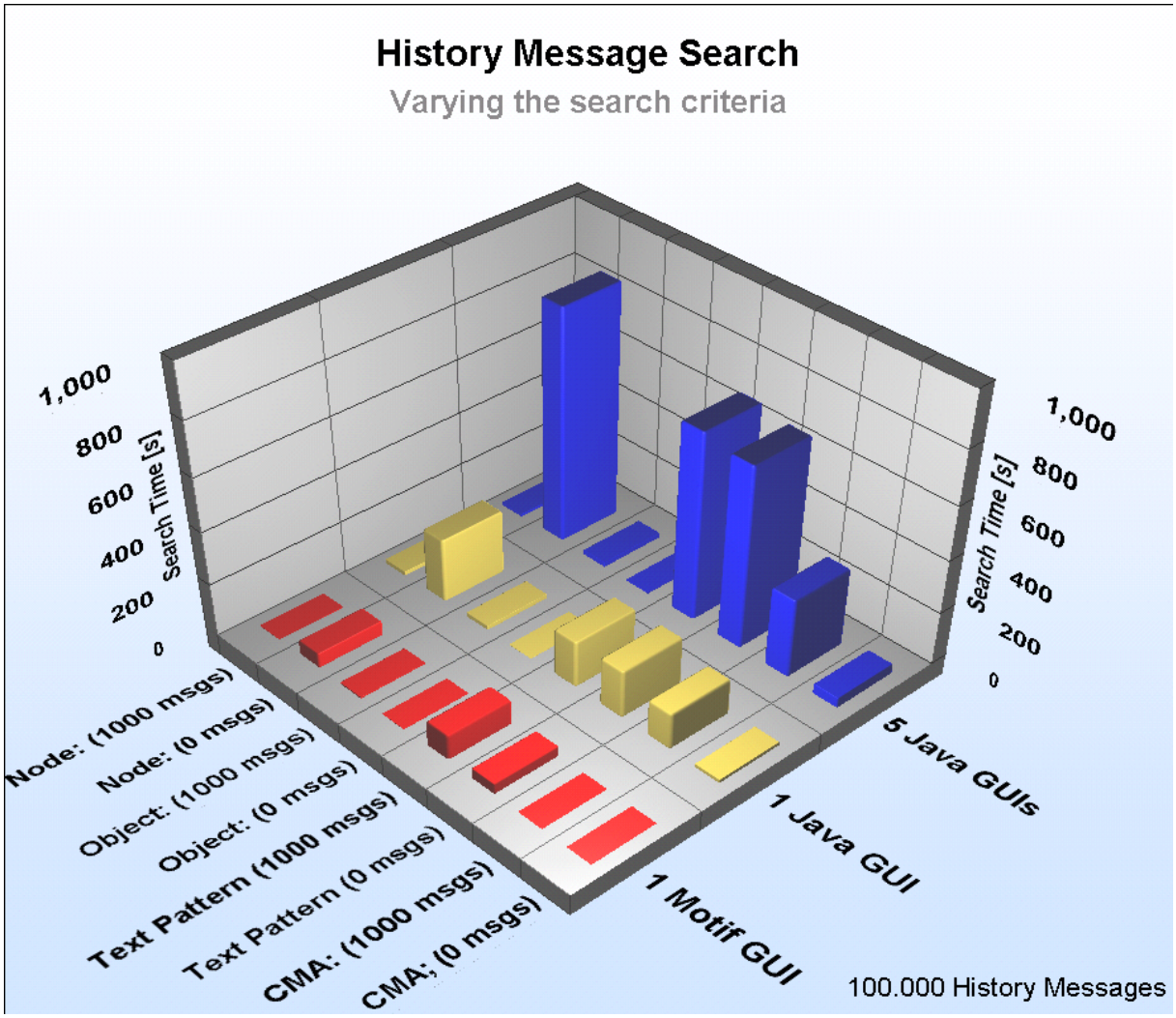


Figure 14: Performance of History Filter - 100.000 history messages

The following facts may be derived from these results:

- Locating history messages when non existing managed nodes are used as the search criteria shows a severe loss of performance.
- Locating history messages using the Motif GUI performs better than using the Java GUI.
- The number of history messages has a linear impact on the time needed for the search.
- The number of Java GUIs performing a parallel search has a direct impact on the time needed for the search.

- Due to the necessary matching algorithm, searching by message text pattern takes longer than searching for a fixed text string.

4.5.1.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- Try to avoid typos when locating history messages via the names of managed nodes. Ensure that only existing node names are used as the search criteria.
- The number of history messages should be kept as small as possible if searches are frequently performed.
- When searching for a message text, use fixed text strings rather than text patterns where appropriate.

4.6 Performance of the History Download

4.6.1 Test: Varying the number of downloaded messages

4.6.1.1 Synopsis

The following metrics are computed in this test.

Metric	• The time to download a varying number of history messages.
Parameter	• The number of history messages in the OVO database

4.6.1.2 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	• 10.000 / 50.000 / 100.000 history messages of severity normal were generated before this test • No active messages did exist • 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 OVO configuration regarding marking/moving acknowledged messages was used. • The OVO message mover process was disabled (OPC_ACK_MOVE_INTERVAL 0).

4.6.1.3 Results

Table 21 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.

Number of messages	10.000	50.000	100.000
Time for download	30,8	144,7	337,3
Time / 1.000 messages	3,1	2,9	3,4
Messages per second	324,7	345,5	296,5

Table 21: Performance of History Download

4.6.1.4 Interpretation

Figure 15 shows the visualization of the results of this test. Shown are the times needed to download the messages and the times to download 1.000 messages.

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

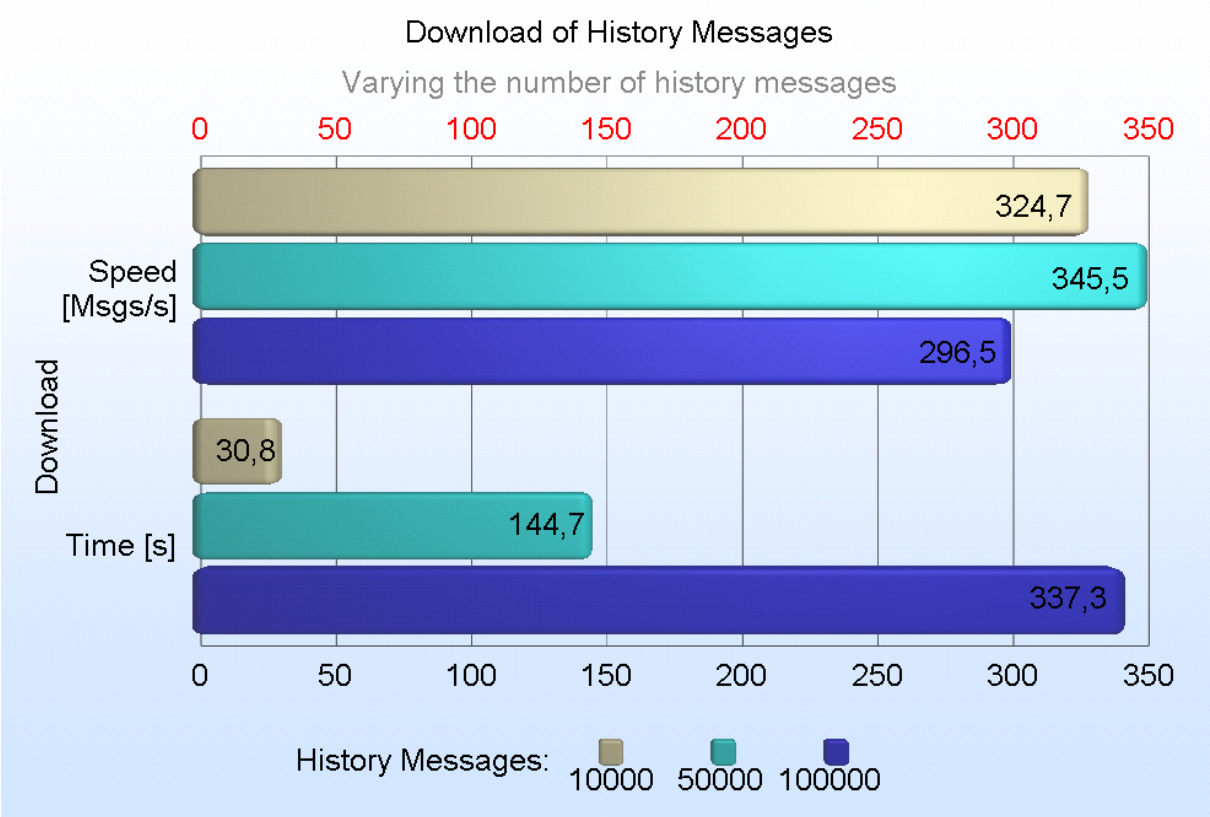


Figure 15: Performance of History Download

The following facts may be derived from these results:

- The download speed varies between 300 and 345 messages per second.
- The time needed to download 1.000 history messages seems to be constant.

4.6.1.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- The download of history can be simply extrapolated once a reference download has been timed.

4.7 Monitor Agent

4.7.1 Test: Performance of monitor agent

4.7.1.1 Synopsis

The following metrics are computed in this test.

Metric	• Time to generate 5.000 and 100.000 messages. • Number of elements in the monitor & message agent queue.
Parameter	• Number of conditions in the monitor policy • Position of firing condition in monitor policy

4.7.1.2 Scenario

Measured Value	• This test measures the time needed to generate 5.000 and 100.000 messages via the OVO monitor agent.
Message Generator	• A varying number of monitor values was generated and sent to an external monitor policy via the opcmon(3) API. • The conditions were ordered according to their threshold, thus the position of the firing condition could be controlled via the value sent to the monitor agent. • Node bank has 10.000 managed nodes
Java GUI options	• No Java GUIs were started for this test
Environment	• One external monitor policy was used with a varying number of conditions. • HP-UX 11.11 HTTPS agent

4.7.1.3 Results

Table 22 lists the results of the test were 100 conditions were used and 5.000 messages were generated.

Matching Condition	Time DB [s]	Rate DB [Msgs/s]
1	38	132
50	39	128
100	38	132

Table 22: Performance of Monitor Agent - 100 conditions and 5.000 messages

Table 23 lists the results of the test were 1.000 conditions were used and 100.000 messages were generated.

Matching Condition	Time DB [s]	Rate DB [Msgs/s]
1	717	139
500	714	140
1000	735	136

Table 23: Performance of Monitor Agent - 1.000 conditions and 100.000 messages

Figure 16 shows the number of items in the message and monitor agent queue for the last test (1.000 conditions and 100.000 messages). The queue lengths were taken in regular intervals (1 second) beginning with the generation of the first message until all expected messages were shown in the message browser.

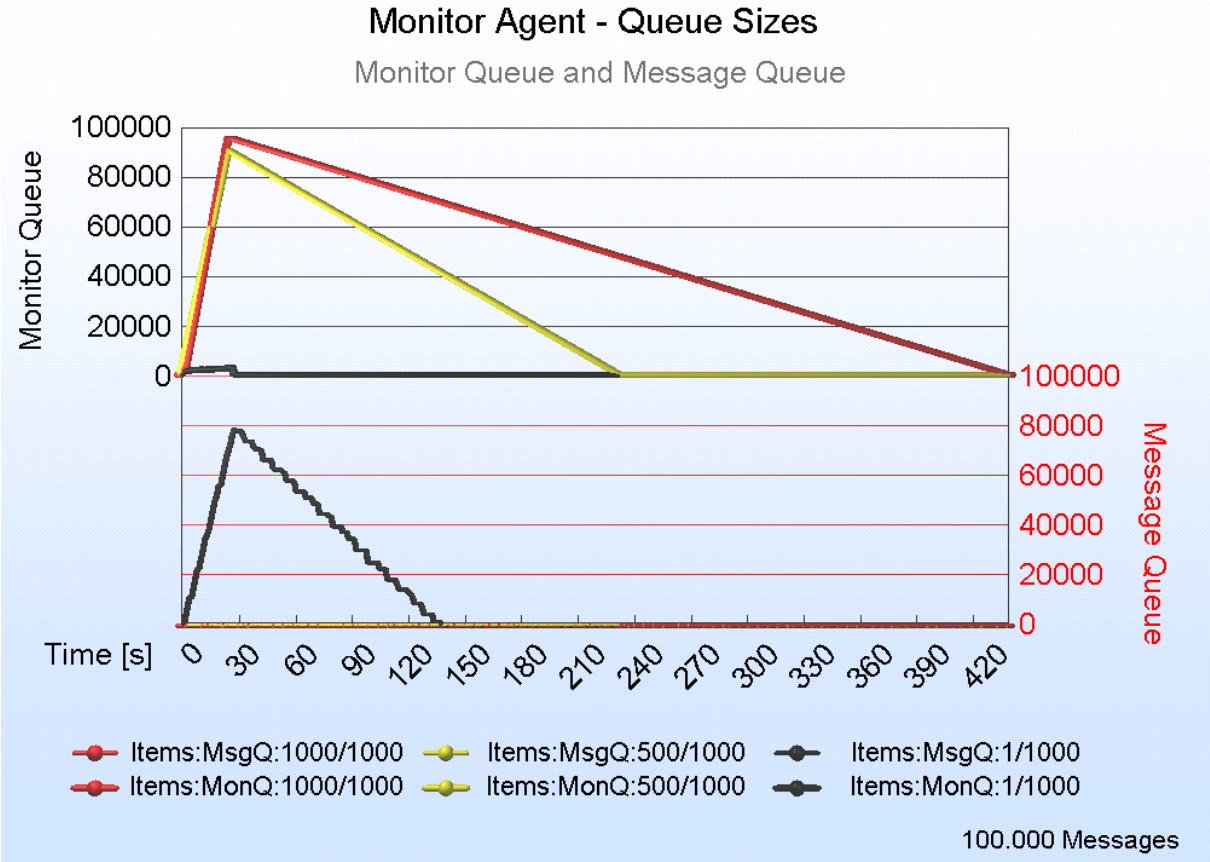


Figure 16: Performance of Monitor Agent – Agent Queue lengths

4.7.1.4 Interpretation

The following facts may be derived from these results:

- The monitor agent can generate messages faster than the server can receive them (agent has finished its tasks after 420 seconds the latest, where the server takes approx. 700 seconds to process the messages).
- The position of the firing monitor condition has no significant impact on the message processing rate on the management server – because the server is the limiting factor.
- The position of the firing monitor condition has a significant impact on the message generation rate on the agent.
  If the 1st condition fires, the message agent part which forwards the data to the server is the limiting part. This can be seen in the graph: monitor queue (which collects the monitor values) does not fill up, but the message agent queue (which collects the messages to be sent to the server) fills up.

If the 1.000th condition matches, the monitor agent part which processes the values is the limiting part. This can be seen in the graph: monitor queue fills up; message agent queue is nearly empty.

- From the data where the 1.000th condition matches, we can see that the rate by which the monitor values are stored in the monitor agent queue is approx. 4.400 values per second, where the rate by which these values are processed is approx. 235 values per second.
- From the data where the 500th condition matches, we can see that the rate by which the monitor values are stored in the monitor agent queue is approx. 4.150 values per second, where the rate by which these values are processed is approx. 445 values per second.
- From the data where the 1st condition matches, we can see that the rate by which the messages are stored in the message agent queue is approx. 4.500 messages per second, where the rate by which these messages are processed is approx. 775 messages per second.
- The processing of the monitor agent queue is faster than the processing of the message agent queue if the first conditions fires. Thus, the limiting factor for sending the messages is the processing of the message agent queue.
- The processing of the monitor agent queue is slower than the processing of the message agent queue if the firing conditions are located at higher list positions (500, 1000). Thus, the limiting factor for sending the messages is the processing of the monitor agent queue.

4.7.1.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- To send monitor messages fast, avoid having long condition lists.
- Have frequently used conditions at the beginning of the condition list.

4.8 SNMP Agent

4.8.1 Test: Performance of SNMP agent

4.8.1.1 Synopsis

The following metrics are computed in this test.

Metric	• Message generated per second via the SNMP trap interceptor. • Number of lost SNMP traps.
Parameter	• Number of SNMP traps per second sent to the trap interceptor • Location of trap interceptor

4.8.1.2 Scenario

Measured Value	• This test measures the number of SNMP messages generated (and thus the number of SNMP traps processed) and the time needed for this task.
Message Generator	• 1.000 / 10.000 SNMP traps were sent using "`snmptrap -L <count>`". Each SNMP trap had a length of 40 bytes (only SNMP payload). • No active messages did exist • No other messages than SNMP 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	• A single SNMP template was assigned to the trap destination. A simple OVO message was generated for the trap. • HP-UX 11.11 agent

4.8.1.3 Results

First, a test was made to show the number of SNMP traps "`snmptrap`" is able to generate at full speed. Table 24 shows the result of this test. Thus, the SNMP trap generator is fast enough for the actual tests.

Traps	Time [s]	Traps/s
100.000	2,30	43.478
1.000.000	15,00	66.666
5.000.000	81,00	61.728

Table 24: Speed of Trap generator

The different test scenarios are shown in Table 25.

Test	Trap Destination		
3-1a	ovtrapi	DCE	Agent
3-1b	NNM	DCE	Agent
3-1c	NNM	HTTPS	Agent
3-1d	ovtrapi	HTTPS	Agent
3-1e	NNM	HTTPS	Server

Table 25: Test Scenarios – SNMP Agent

Table 26 lists the results of the tests. The SNMP traps were sent in multiple chunks of a varying size, after each chunk a delay of 1 second was inserted. Sending all SNMP traps at once showed that the trap receiver was not able to process all the traps, thus traps were lost (even if NNM was the trap receiver).

In tests were NNM was the trap receiver, the information regarding NNM was added to the result table, too.

Legend:

Traps Sent	Number of total traps sent in this test
Chunk Size	Number of traps sent at once without any delay in between. After each chunk a delay of 1 second was implemented.
Traps Received (NNM)	Number of total traps received by NNM in this test
Time NNM	Time in seconds until all traps were received by NNM.
Rate NNM	Number of traps per second which were received by NNM
Traps Received (OVO)	Number of total traps received by the OVO trap interceptor
Time DB	Times in seconds until all SNMP messages were stored in the OVO database. Note that this is a server based metric.
Rate DB	Number of SNMP messages per seconds which were stored in the OVO database.

Test	Traps Sent	Chunk size 1 sec delay after each chunk	Traps received (NNM)	Time NNM [msg/s]	Rate NNM [msg/s]	Traps received (OVO)	Time DB [s]	Rate DB [msg/s]
3-1a	1.000	1.000	-	-	-	744	7	106
	1.000	500	-	-	-	1000	8	125
	10.000	500	-	-	-	10.000	89	112
3-1b	1.000	1.000	1.000	3	333	1.000	9	111
	1.000	500	1.000	3	333	1.000	9	111
	10.000	10.000	5.972	15	398	5.972	52	115
	10.000	5.000	6.241	15	416	6.242	52	120
	10.000	2.000	6.097	15	406	6.097	51	120
	10.000	1.000	10.000	24	417	10.000	83	120
	10.000	500	10.000	24	417	10.000	84	119
3-1c	1.000	1.000	1.000	2	500	1.000	8	125
	10.000	10.000	5.991	14	428	5.991	51	117
	10.000	1.000	9.978	24	416	9.979	84	119
	10.000	500	10.000	23	435	10.000	84	119
3-1d	1.000	1.000	-	-	-	740	7	106
	1.000	500	-	-	-	1.000	10	100
	10.000	500	-	-	-	10.000	84	119
3-1e	1.000	1.000	1.000	3	333	1.000	9	111
	10.000	10.000	7.542	18	419	6.543	54	121
	10.000	1.000	10.000	24	417	10.000	85	118
	10.000	500	10.000	24	417	10.000	84	119

Table 26: Test Results – SNMP Agent Performance

The last test 3-1e was repeated with a NNM 7.5 installation. Table 27 shows the results.

Test	Traps sent	Chunk size 1 sec delay after each chunk	Traps received (NNM)	Time NNM [msg/s]	Rate NNM [msg/s]	Traps received (OVO)	Time DB [s]	Rate DB [msg/s]
3-1e	1.000	1.000	1.000	3	333	1.000	10	100
	10.000	10.000	5.296	19	279	5.296	52	102
	10.000	5.000	5.173	18	287	5.173	49	106
	10.000	1.000	9.967	35	285	9.967	103	97
	10.000	500	10.000	35	286	10.000	87	115
	98.754	98.754	52.204	180	290	52.204	489	107

Table 27: Test Results – SNMP Agent Performance – NNM 7.5

4.8.1.4 Interpretation

The following facts may be derived from these results:

- NNM is able to process SNMP Trap bursts of up to 1.000 traps per second without missing traps. Beginning with a burst of 2.000 traps/s, NNM starts missing traps. This is independent of the agent type (HTTPS vs. DCE).
- The OVO ovtrapi is able to process SNMP Trap bursts of up to 500 traps per second without missing traps. Starting with bursts of 1.000 traps/s ovtrapi misses traps.
- The message generation rate is nearly independent of which process receives the SNMP traps, NNM or ovtrapi. This rate is approx. 120 messages per second.
- If NNM is used and is able to process the SNMP trap bursts without missing traps, all traps are forwarded to OVO.
- The message generation rate is slightly lower if NNM 7.5 is used.
- NNM 7.5 starts missing traps for bursts of approx. 1.000 traps per second.

4.8.1.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- If there is a need to process SNMP trap bursts of more than 500 traps per second, NNM should be used as the trap receiver – if possible.
- Consider implementing multiple different OVO agents as SNMP trap destinations, i.e. use the distributed approach. Note that in this case one trap should only be sent to one OVO agent to prevent the generation of duplicate messages.

4.9 Miscellaneous Tests

4.9.1 Test: Policy and Instrumentation Deployment

4.9.1.1 Synopsis

The following metrics are computed in this test.

Metric	• Time needed to deploy instrumentation and policies.
Parameter	• Number of policies • Number of conditions in policies • Agent type and platform

4.9.1.2 Scenario

Measured Value	• This test measure the time needed to deploy different sets of policies and instrumentation to different agent types and platforms. • The first time recorded is the total time needed from deployment start (Admin GUI) until the arrival of the completion message in the browser. This time includes the time needed for the policy creation for HTTPS agents. • The second time was taken from the starting the deployment from the command line. This time does not include the policy creation step for HTTPS agents.
Message Generator	• No messages were generated during the test
Java GUI options	• No Java GUIs were started for this test

Environment	• The OSSPI was used for the deployment tests.
	  ○ HP: Quickstart HP-UX, 1 Message, 8 Monitor, 8 Logfile Policies
	  ○ Linux: Quickstart Linux, 1 Message, 5 Monitor, 9 Logfile Policies
	  ○ Windows: Core/Windows 2000: 1 Message, 33 Monitor, 7 Logfile Policies
	• OSSPI instrumentation contained 134 files (15 from CHO tests)
	• Installed server patch A.08.11
	• No tests were performed for HP Itanium DCE agents

4.9.1.3 Results

Table 28 lists the results of this test. Shown are the times in seconds needed to complete the operation. For HTTPS agents, both the total time (policy creation and deployment) and the time needed only for deployment is listed. For DCE agents, only the total time is listed since there is no policy creation step.

Platform	HP PA DCE	HP PA HTTPS		HP IA HTTPS		Win DCE	Win HTTPS		Linux DCE	Linux HTTPS	
Number of policies	**Tot**	**Tot**	**Depl**	**Tot**	**Depl**	**Tot**	**Tot**	**Depl**	**Tot**	**Tot**	**Depl**
Monitor: 1 cond											
1	2	2	2	2	2	2	2	2	2	2	2
50	3	17	2	15	14	3	11	6	3	10	8
100	5	38	30	33	27	15	23	14	3	21	14
Monitor: 1 cond + action											
100	4	32	30	31	26	4	26	14	4	21	14
Monitor: 1000 conds											
1	3	3	2	3	2	3	3	2	3	3	2
50	95	79	23	71	20	80	68	14	70	65	14
100	**187**	**144**	**45**	**135**	**38**	**130**	**131**	**28**	**167**	**126**	**27**
OSSPI policies	3	8	6	7	6	3	9	5	2	5	3
OSSPI instrumentation	3	42	-	36	-	3	12	-	9	20	-

Table 28: Policy and Instrumentation Deployment

4.9.1.4 Interpretation

The following facts may be derived from these results – only the results with the fix are discussed:

- The number of policies has a linear impact on the deployment time.
- The number of policy conditions has a linear impact on the deployment time.
- Deployment to DCE agents is faster than the deployment to HTTPS nodes.
- Adding actions to the policies does not result in additional deployment time.
- Deployment to Linux HTTPS agents is significantly faster than deployment to HP-PA DCE agents (only 67% of HP-PA time).
- Deployment of the OSSPI policies is fairly fast (3 – 9 seconds).

- Deployment of the OSSPI instrumentation is significantly faster for DCE agents.

4.9.1.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- The deployment to HTTPS takes more time than the deployment to the DCE agent (except on Linux).
- Deployment of policies to HTTPS agents take longer for the first time – where the policy creation is needed.

4.9.2 Test: Network Traffic

4.9.2.1 Synopsis

The following metrics are computed in this test.

Metric	• Number of packets and bytes per OVO message sent from the agent to the server.
Parameter	• Number of messages • Size of message text (i.e. size of the message) • Agent type (HTTPS vs. DCE)

4.9.2.2 Scenario

Measured Value	• This test measures the network traffic for messages sent from the OVO agent to the OVO server.
Message Generator	• Messages with severity normal and with a varying length of message text were generated on Windows managed nodes
Java GUI options	• No Java GUIs were started for this test
Environment	• The network traffic was captured and analyzed

4.9.2.3 Results

Table 29 lists the results of this test for a Windows HTTPS node.

The last column lists the number of bytes in the communication between agent and server per message.

HTTPS Agent (Windows)								
	No. Mes-sages	**Length of message text**	**Packets To Srv**	**Packets from Srv**	**Bytes to Srv**	**Bytes from Srv**	**Bytes to Srv per Msg**	**Comm. Bytes per Msg**
	200	50	119	22	141979	2646	710	723
	500	50	292	48	353043	4869	706	716
	1.000	50	574	110	703038	8589	703	712
Average		***50***					***706***	***717***
	200	500	198	61	239176	4986	1196	1221
	500	500	487	146	595816	10749	1192	1213
	1.000	500	964	355	1189458	24065	1189	1214
Average		**500**					***1192***	***1216***

Table 29: Network Traffic – HTTPS node

The results for the same test, but for a DCE agent, are shown in Table 30.

DCE Agent (Windows)								
	Messages	**Length of message text**	**Packets To Srv**	**Packets from Srv**	**Bytes to Srv**	**Bytes from Srv**	**Bytes to Srv per Msg**	**Comm. Bytes per Msg**
	200	50	201	201	190322	26934	952	1086
	500	50	501	501	475622	67134	951	1086
	1.000	50	1001	1001	951122	134134	951	1085
Average		***50***					***951***	***1086***
	200	500	201	201	206322	26934	1032	1166
	500	500	501	501	515622	67134	1031	1166
	1.000	500	1001	1001	1031122	134134	1031	1165
Average		**500**					***1031***	***1166***

Table 30: Network Traffic – DCE node

4.9.2.4 Interpretation

If the average values for the communication bytes are set in relation to the length of the message text, we have the graphs as shown in Figure 17.

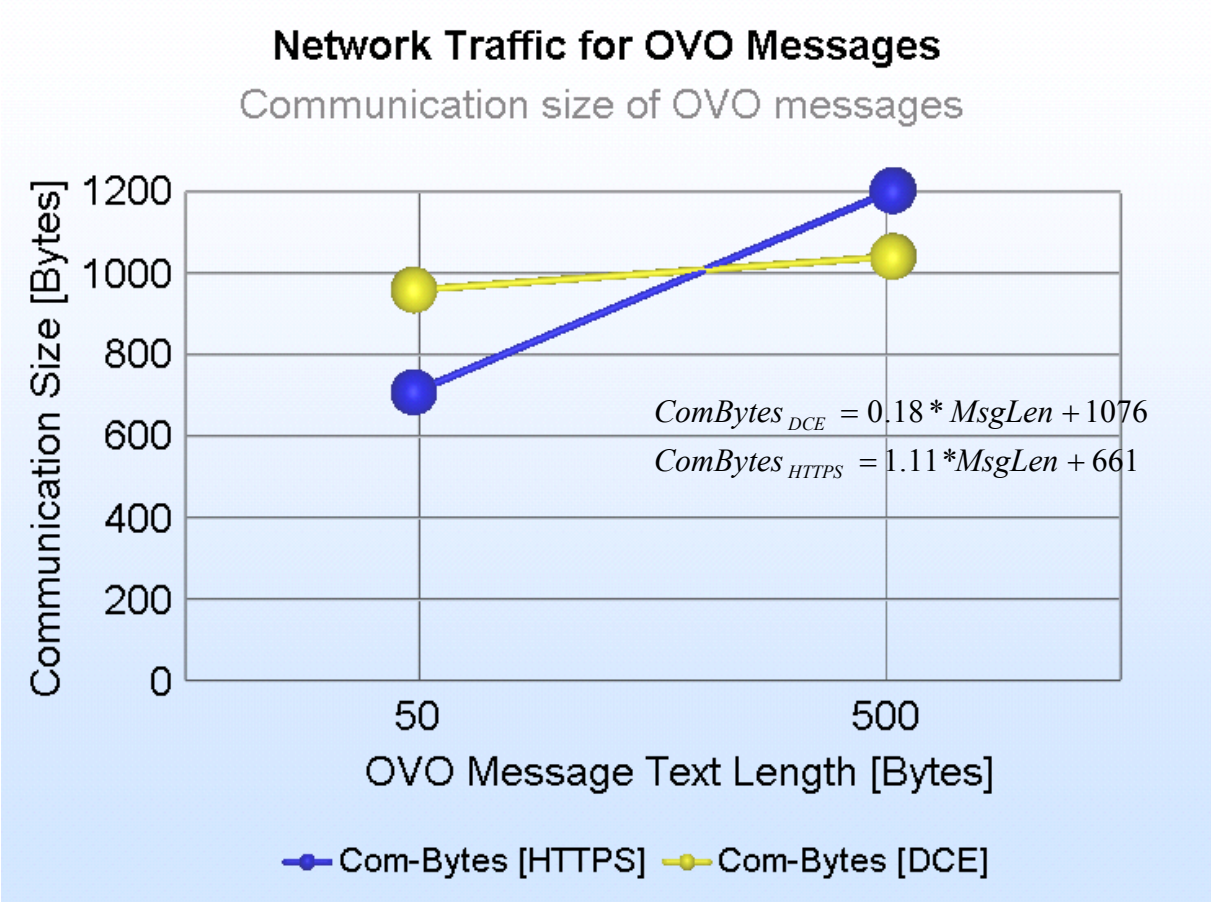


Figure 17: Network Traffic for OVO Messages

The following facts may be derived from these results:

- The amount of network traffic exchanged between agent and server for a message depends linear on the size of the message text.
- For short message texts, fewer bytes are exchanged for HTTPS communication than for DCE communication, i.e. the fixed overhead of DCE communication is higher.
- For longer message texts, the size of the HTTPS messages grows faster than the size of the DCE messages, i.e. the variable overhead of HTTPS communication is higher.

4.9.2.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- Small messages should be used for HTTPS agent – if network traffic is a factor.

4.9.3 Test: Java GUI Resources

4.9.3.1 Synopsis

The following metrics are computed in this test.

Metric	• Memory size required by Java GUIs on display station and on OVO server.
Parameter	• Number of messages • Number of Java GUIs

4.9.3.2 Scenario

Measured Value	• This test measures the free memory available on the Java display station and the OVO sever for different scenarios.
Message Generator	• Message were generated upfront and not during the test
Java GUI options	• Different OVO/UNIX user accounts – but same responsibility. • *show all* messages • refresh interval *5 seconds*
Environment	• n/a

4.9.3.3 Results

Table 31 lists the results for this test.

Active Messages	**0**		**5.000**	
Number of Java GUIs	Memory consumption on server [MB]	Memory consumption on GUI [MB]	Memory consumption on server [MB]	Memory consumption on GUI [MB]
0	0	0	0	0
1	20	41,60	50	47,7
5	80	179,3	210	203,7

Table 31: Java GUI – Memory Requirements

4.9.3.4 Interpretation

Figure 18 shows the visualization of the test results.

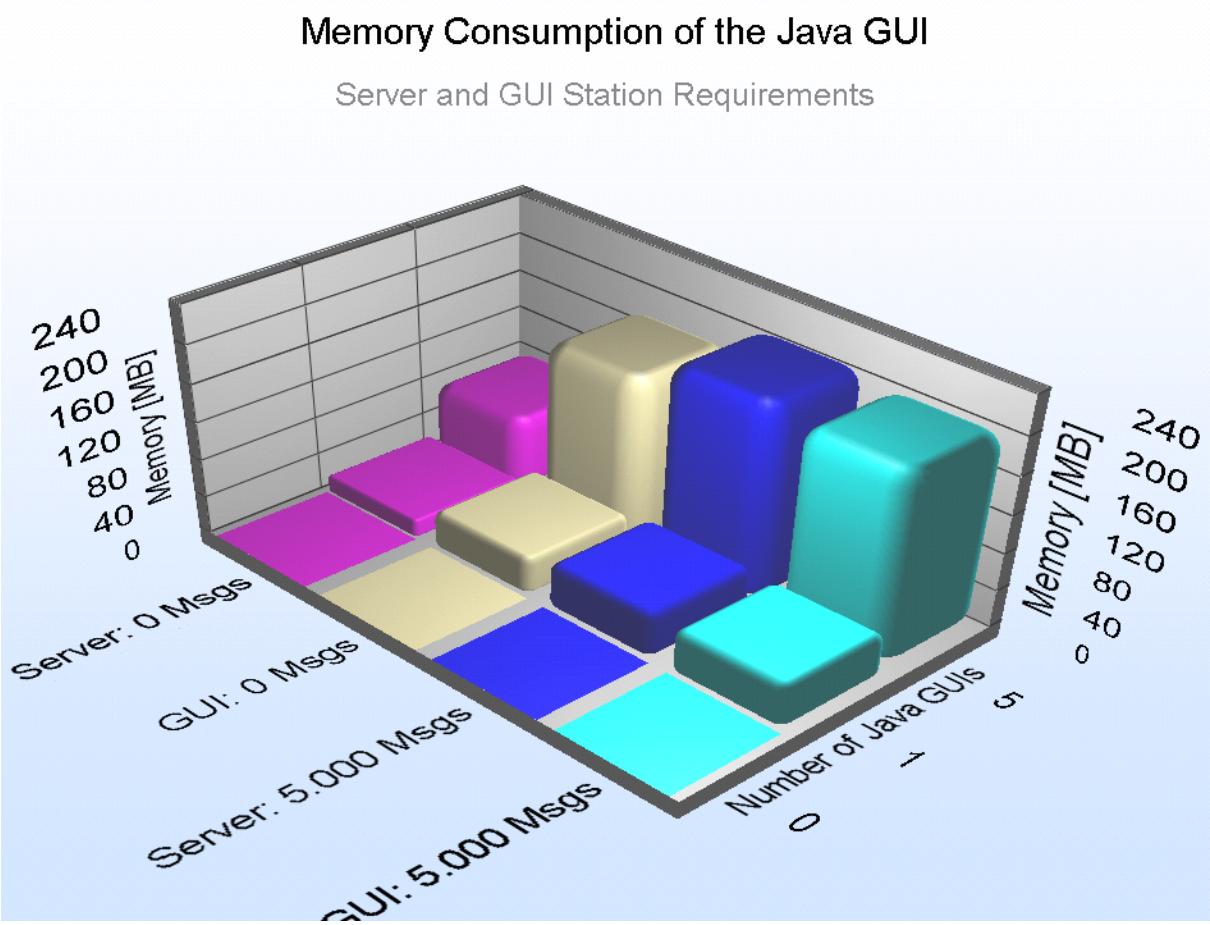


Figure 18: Java GUI – Memory Requirements

The following facts may be derived from these results:

- One Java GUI needs approx 16 – 20MB of main memory on the management server, plus 6 MB per 1.000 active messages.
- One Java GUI needs approx 36 – 42MB of main memory on the display station, plus 1.2 MB per 1.000 active messages.

4.9.3.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- The main memory needed on the server and the display station may be approx. computed.

4.9.4 Test: Memory Consumption on Agent

4.9.4.1 Synopsis

The following metrics are computed in this test.

Metric	• Memory size required by the OVO agent
Parameter	• Number of policies loaded by the agent • Agent type and platform

4.9.4.2 Scenario

Measured Value	• This test measures the memory required by all OVO agent processes
Message Generator	• No messages were generated during this test
Java GUI options	• No Java GUIs were used for this test
Environment	• The OSSPI policies were distributed to the agent • Shared memory is counted only once

4.9.4.3 Results

Table 32 lists the results for this test.

Agent		HP PA DCE	HP PA HTTPS	HP Itanium HTTPS	Linux HTTPS	W2k3 DCE	W2k3 HTTPS
No poli-cies		Mem on Agent [MB]	Mem on Agent [MB]	Mem on Agent [MB]	Mem on Agent [MB]	Mem on Agent [MB]	Mem on Agent [MB]
	User Mem	15,4	14,6	44	18,2	11,5	18,6
	Virtual Mem	28,2	49,7	92,2			
All kind of policies	User Mem	28,5	31,9	60	26,4	23	29,3
	Virtual Mem	46,8	72,9	129,2			
Lcore-Virt-Mem			92,8	90			

Table 32: Agent – Memory Requirements

4.9.4.4 Interpretation

The following facts may be derived from these results:

- The HTTPS agent needs more main memory than the DCE agent. The LCore components are listed separately in the table (HP).
- If policies are deployed to the managed nodes, the growth in main memory is higher for the HTTPS agent than for the DCE agent.
- The difference in main memory requirements between DCE and HTTPS agent is higher for HP-PA agents than for Windows 2003 agent.

4.9.4.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- HTTPS agents need more main memory than DCE agents.

4.9.5 Test: Performance of opcmon/opcmsg CLI

4.9.5.1 Synopsis

The following metrics are computed in this test.

Metric	• Time needed to start 1.000 opcmon/opcmsg commands in sequence
Parameter	• Agent type and platform

4.9.5.2 Scenario

Measured Value	• This test measures time needed to start 1.000 `opcmon` and `opcmsg` commands in sequence
Message Generator	• Messages of severity *normal* were generated for this test.
Java GUI options	• No Java GUIs were used for this test
Environment	• The command line interface was used for this test. • The monitor values were sent to an external monitor policy. • No tests were performed for HP Itanium DCE agents

4.9.5.3 Results

Table 33 lists the results for this test.

Number of Executions	1.000			
Platform	opcmon [s]	opcmsg [s]	opcmon [calls/s]	opcmsg [calls/s]
Windows 2003 x86 HTTPS	95	96	10,5	10,4
Windows 2003 x86 DCE	72	81	13,9	12,3
Linux Red Hat x86 HTTPS	47	47	21,3	21,3
Linux Red Hat x86 DCE	21	21	47,6	47,6
HP-UX 11.23 IA HTTPS	128	129	7,8	7,8
HP-UX 11.23 IA DCE	Not Tested			
HP-UX 11.11 PA HTTPS	310	310	3,2	3,2
HP-UX 11.11 PA DCE	53	53	18,9	18,9

Table 33: Performance of opcmon/opcmsg CLI

Figure 19 shows the visualization of the result.

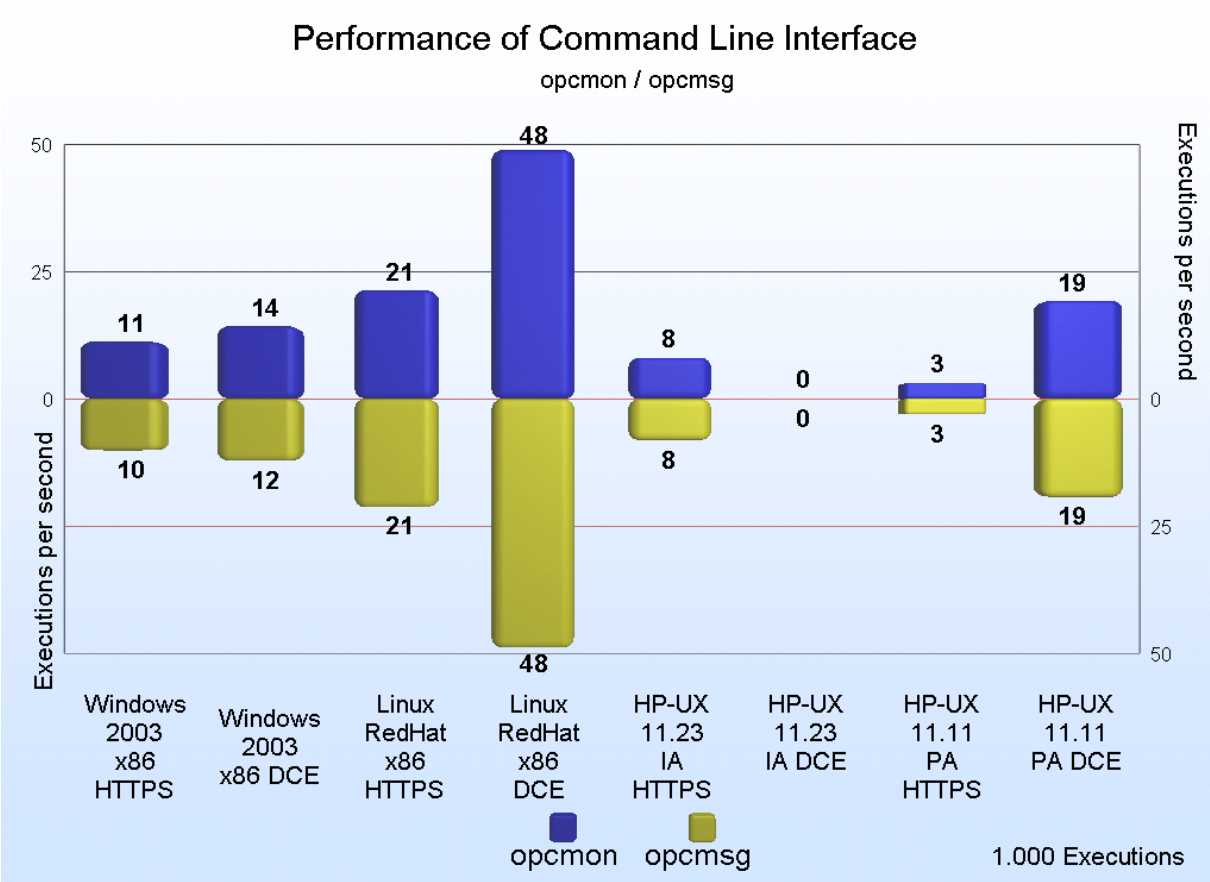


Figure 19: Performance of opcmon/opcmsg CLI

4.9.5.4 Interpretation

The following facts may be derived from these results:

- The time for one `opcmon`/`opcmsg` command to complete is very high for HTTPS agents on HP systems.
- The DCE versions of the tools are always faster than the HTTPS versions – on the same platform.
- Event the Linux HTTPS agent is faster than the HP-PA DCE agent – and 7 times faster than the HP-PA HTTPS agent.

4.9.5.5 Conclusions

Regarding the tests in this chapter, the following conclusions may be drawn:

- If `opcmon`/`opcmsg` needs to be called at a high rate on HP HTTPS agents, an alternative architecture should be considered (API calls, MSI programs, background process which is feed by a light communication channel).

5 Appendix

5.1 OVO message acknowledge internals

Note that the parameters described in this section have to be set in the opcsvinfo file for OVO versions prior to 8.0. Beginning with OVO 8.0, the opcsvinfo file has been substituted by a separate data store, for which the parameters have to be set using

"ovconfchg –ovrg server –ns opc –set <parameter> <value>"

When a message is acknowledged the related rows out of several tables (opc_act_messages, opc_msg_text, opc_orig_msg_text and if applicable opc_annotation and opc_anno_text) get moved into the corresponding history tables.

There is however an exception to this rule:

If more than 50 messages get acknowledged, then these messages are not moved immediately into the corresponding history tables but are just flagged as being acknowledged. A background job (opcdbmsgmv) which is scheduled to run every 2 hours, will move these messages into the history tables at a later point in time thus not blocking the GUI which started the message acknowledge steps. Several parameters allow tuning of all these steps if this is needed:

OPC ACK MOVE INTERVAL

Description:
Interval in seconds in which the opcdbmsgmv process is started by the opcctlm process to move the flagged (acknowledged) messages to the history tables.

Default: 7200 seconds (= 2 hours)

How to use this variable:
If set to 0, the message move is disabled.
The history download will recognize marked messages and download them.

The message move should only be disabled in these cases; otherwise the active tables might grow to sizes which can seriously impact the performance of the GUI:

- for test purposes
- if regular history downloads are run and history messages are not kept very long in the DB.
- opcdbmsgmv is run by other means (cron or an OVO scheduled action at specific times)

Example:
configure the opcdbmsgmv process to run every hour in the configuration:

- OPC_ACK_MOVE_INTERVAL 3600

OPC DIRECT ACKN LIMIT

Description:
This variable helps determine whether the messages to be acknowledged should be moved to the history tables immediately or whether this move can be deferred. If more messages than this limit (default value: 50 messages) are acknowledged then they are only flagged as acknowledged and are moved in the background by the opcdbmsgmv process.

How to use this variable:
It is not expected that normal operation in the browser would require mass acknowledgements of messages.

If regularly 10, 20 or more messages are acknowledged at once in the GUI and this is perceived as slow, then this variable should be set to a value below the number of messages acknowledged at once. So modifying this variable will have a direct impact on the GUI performance.

Note, that the Java GUI uses the APIs and therefore acknowledges one message at a time. If you want that messages acknowledged by the Java GUI are only marked, set OPC_DIRECT_ACKN_LIMIT to 1.

Example:
configure the acknowledgment process so that messages are always marked; add following line to the configuration data:

- OPC_DIRECT_ACKN_LIMIT 1

OPC ACK COMMIT COUNT

Description:
for performance reasons whenever multiple messages need to be acknowledged, these get grouped together and multiple messages are committed in one transaction. This count allows defining the number of messages to be committed in one transaction.

Default : 100 messages

How to use this variable:
Please be careful when increasing this value as this might cause locking conflicts on the DB if set too high! Every commit will release locks on the message tables thus allowing for better concurrency. When would you increase this value?

- if you want to speed up opcdbmsgmv

When would you decrease this value?

- if locking conflicts occurred during message acknowledgement.

Modifying this variable will have an impact on the opcdbmsgmv performance.

Example:
configure the acknowledgement process to always perform a commit after 50 messages were moved to the history tables.
In the configuration data:

```
OPC_ACK_COMMIT_COUNT   50
```

Sample scenarios

The settings of these variables depend on the environment. Therefore, there is not one set of settings that is optimal for all customers. Here are some sample configurations and appropriate settings for these cases:

- Many acknowledgements, OVO is used in the day shift only:

  In this case, it makes sense to always mark messages by adding following info to the configuration data:

  OPC_DIRECT_ACKN_LIMIT 1

  Further it makes sense to run opcdbmsgmv in the night after other maintenance tasks (history download, backup and so on). Since OPC_ACK_MOVE_INTERVAL only allows specifying an interval and not a fix time, you can do following: Disable the automatic scheduling by adding following info to the configuration data:
  OPC_ACK_MOVE_INTERVAL to 0

  Now setup an OVO scheduled action that is run once per night and runs opcdbmsgmv.

- History messages are kept only short time:

  In this case, it makes sense to always mark messages by adding following info to the configuration data:

  OPC_DIRECT_ACKN_LIMIT 1

  It also makes sense to disable opcdbmsgmv and let the history download handle the marked messages. Since history messages are not kept very long, the
  impact of the marked messages in the active message tables on starting OVO GUIs is small. To disable opcdbmsgmv, add following info to the configuration data:

  OPC_ACK_MOVE_INTERVAL to 0

- If you want all messages to be moved immediately:

  Set the variable OPC_DIRECT_ACKN_LIMIT to a very high number (for example 1000000), so that the messages are always moved immediately to the history tables.

- Many lock timeouts caused by opcdbmsgmv

  You see a lot of lock timeout errors like the following in the opcerror file while opcdbmsgmv is running:

  ```
  04/05/00 15:12:03  OpC message acknowledge utility(3095)
  [chk_sqlcode.scp:99]:
  Database: ORA-00054: resource busy and acquire with NOWAIT specified
  (OpC50-15)
  Retry. (OpC51-22)
  ```

OVO does three retries if a lock timeout occurs. If some tasks cannot be finished within this three retries, you may reduce the amount of messages that are processed by opcdbmsgmv within one transaction. This reduces the time that other processes have to wait. You can do this by reducing the value of OPC_ACK_COMMIT_COUNT, for example from the default of 100 to 50.