

Benchmarking Caplin Liberator 4.4

November 2007 ■ Confidential

Contents

1	Preface.....	1
1.1	What this document contains.....	1
1.2	Who should read this document.....	1
1.3	Related documents.....	1
1.4	Feedback.....	1
1.5	Acknowledgments.....	2
2	Overview and headline results.....	3
2.1	About the benchmark tests.....	3
2.2	Headline figures.....	4
2.3	Caplin's benchmark tools.....	4
3	Test results for headline figures.....	5
3.1	Low update rates for large numbers of end users.....	5
3.2	Medium update rates for large numbers of end users.....	6
3.3	High update rates for large numbers of end users.....	7
3.4	Very high and extreme update rates.....	8
3.5	Test details.....	10
4	How to benchmark using Caplin's tools.....	11
4.1	Package contents.....	11
4.2	Architecture.....	12
4.3	Test DataSource application (Benchsrc).....	12
4.4	Test RTTP client application (Benchrttp).....	13
4.5	Benchmark requirements.....	14
4.6	Installation and Setup.....	15
	Installing Benchtools	15
	Benchsrc setup	15
	Benchrttp setup	16
	Liberator installation	16
	Liberator setup	18
5	How Caplin's benchmark tests were conducted.....	20
5.1	Test method.....	20
5.2	Test configurations	21
5.3	Test software	23
5.4	Test hardware.....	24

5.5	The network.....	25
6	Results of additional tests.....	26
6.1	Effect of session threads on performance.....	26
	Low backend data set	27
	Medium backend dataset	29
6.2	Effect of DataSource threads on performance.....	31
	High backend dataset	31
	High backend dataset - adding session threads	33
	Medium backend dataset	35
6.3	Effect of burst configuration (batching) on performance.....	37
	Low backend dataset	37
6.4	Effect of message size on performance.....	39
	Overall size of a message	39
	Number of fields in a message	41
6.5	Effect of failover on performance.....	43
	Latency during failover scenario	43
6.6	Liberator 4.4 versus 4.2.....	45
	Low backend data set	45
	Medium backend data set	47
7	Frequently asked questions.....	49
7.1	What burst configuration should we use?.....	49
7.2	How many threads should we configure?.....	49
7.3	How do message sizes affect performance?.....	50
7.4	How much bandwidth will our Liberator use?.....	50
7.5	How many subscriptions can Liberator handle?.....	50
7.6	How much disk space will our Liberator need?.....	51
7.7	Why are there no CPU usage measurements for some tests?.....	51
8	Glossary of terms and acronyms.....	52

1 Preface

1.1 What this document contains

This document details the results of a set of performance benchmark tests carried out on Caplin Liberator 4.4. It is hoped that the information provided in this report will assist customers in production capacity planning when deploying Liberator 4.4

- ◆ [About the benchmark tests](#)^[3] gives a summary of the tests performed by Caplin Systems
- ◆ [Headline figures](#)^[4] summarizes the main results of the tests relating to typical performance profiles for Caplin Liberator.
- ◆ The detailed results of these tests (with performance graphs) are in [Test results for headline figures](#)^[5].
- ◆ There is a section on [how to use Caplin's Benchtools](#)^[11] to perform these kinds of benchmark tests yourself in your own environment.
- ◆ Detailed information is provided on [how Caplin's benchmark tests were conducted](#)^[20].
- ◆ The remainder of the document contains the [results of additional tests](#)^[26] to show the effect of improvements to how threads are implemented in Liberator 4.4, the effect of message size and failover, and a performance comparison of Liberator 4.4 and Liberator 4.2.
- ◆ There is also a section addressing [frequently asked questions](#)^[49] concerning Liberator benchmarking.

1.2 Who should read this document

This document is intended for anyone who is evaluating Caplin Liberator's performance characteristics, or who is planning to deploy Caplin Liberator. Typical readers would be:

- ◆ Technical Managers
- ◆ Enterprise Architects and System Architects
- ◆ System Administrators

1.3 Related documents

- ◆ **Caplin Platform Overview**
Gives a technical overview of the whole Caplin Platform, including the role that Caplin Liberator plays within the Platform.
- ◆ **Caplin Liberator 4.4 Administration Guide**
A guide to configuring and running Caplin Liberator.

1.4 Feedback

Customer feedback can only improve the quality of our product documentation, and we would welcome any comments, criticisms or suggestions you may have regarding this document.

Please email your thoughts to documentation@caplin.com.

1.5 Acknowledgments

AMD and *Opteron* are trademarks of Advanced Micro Devices, Inc.

Dell and *PowerEdge* are trademarks of Dell Inc in the United States and other countries.

Intel and *Intel Xeon* are registered trademarks of Intel Corporation in the U.S. and other countries.

Java, *JavaScript* and *JMX* are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. or other countries.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Enterprise Linux is a registered trademark of Red Hat, Inc. in the United States and other countries.

2 Overview and headline results

2.1 About the benchmark tests

The benchmark tests detailed in this document are designed to show how Caplin Liberator will perform on the Linux® platform.

The main factor affecting the overall performance of Liberator is the power of the machine on which it runs. The tests were conducted on servers representing typical commercially available machines that could be used to host Web servers and server applications. A single Liberator instance was run on one of two machines: a 4 CPU Dell™ PowerEdge™ server running at 2.4GHz per CPU ("Test Machine 1") or an 8 CPU HP ProLiant server running at 2.8GHz per CPU ("Alternative Test Machine 1"). The operating system on both servers was Redhat Linux Enterprise Linux 4 – 64 bit. For more details of these machines see [Test hardware](#)^[24].

Message latency versus CPU usage

The key item measured in the tests was the end-to-end message latency against the number of logged in clients, and by implication the number of update messages being sent out to the totality of the connected client base.

Although some of the test results show CPU usage, in practice end-to-end message latency is more significant as a measure of Liberator performance than CPU usage. Message latency has a direct impact on users and may increase long before CPU usage reaches its maximum. The aim of sizing a system incorporating Caplin Liberator should be to achieve a maximum desired message latency for a given maximum update rate.

For more information on this see [Why are there no CPU usage measurements for some tests?](#)^[51]

Liberator 4.4. performance improvement

Benchmark tests were previously performed on Caplin Liberator 3.6. The latest version of Liberator (4.4) contains several performance improvements, most notably the implementation of a separate thread for each DataSource connection, so it was considered appropriate to run a new set of performance tests on Liberator 4.4.

Two of the tests specifically illustrate the performance improvement of Liberator 4.4 relative to Liberator 4.2 – and hence relative to Liberator 3.6, since Liberator 4.2 has the same threading architecture as Liberator 3.6. See [Liberator 4.4 versus 4.2](#)^[45].

Test set up

For detailed information on the test set up used at Caplin Systems see [The test environment](#)^[20].

Note: It is hoped that the information provided in this report will assist customers in production capacity planning. However, while the tests were designed to emulate real-world traffic and user scenarios, they were conducted using specific hardware running in an isolated environment and therefore no guarantees can be made that identical results will be achieved in other environments.

2.2 Headline figures

The following table summarizes the main results of the benchmark tests performed by Caplin Systems on Caplin Liberator 4.4. The tests were run against a single Liberator instance running on the indicated test machine and were set up so that a large proportion of the data received by each user was unique to that user.

The test results are shown in graphical form in [Test results for headline figures](#)⁵, together with more detailed analysis and explanation.

Test profile	Result	See graph and discussion in:	Test machine
Low update rates for large numbers of end users (per-user update rate of 1 update/sec). Users subscribed to a random set of objects out of a pool of 1000 objects.	Liberator performs extremely well, handling 30,000 end users with very low message latency (less than 30 milliseconds end-to-end).	Low update rates for large numbers of end users ⁵	HP ProLiant server: 8 CPUs, 2.8Ghz per CPU
Medium update rates for large numbers of end users (per-user update rate of 10 updates/sec). Users subscribed to a random set of objects out of a pool of 1000 objects.	Liberator performs well, handling 10,000 concurrent end users with low message latency (less than 50 milliseconds end-to-end). With 15,000 concurrent end users the message latency is still only 75 milliseconds.	Medium update rates for large numbers of end users ⁶	HP ProLiant server: 8 CPUs, 2.8Ghz per CPU
High update rates (per-user update rate of 50 updates/sec). Users subscribed to a random set of objects out of a pool of 1000 objects	At this high per-user update rate Liberator can still handle 10,000 end users with less than 150 milliseconds latency.	High update rates for large numbers of end users ⁷	HP ProLiant server: 8 CPUs, 2.8Ghz per CPU
Very high update rates (per-user update rate of 100 updates/sec). Users subscribed to 100 objects out of a pool of 20,000 objects.	At this very high per-user update rate Liberator can handle 10,000 concurrent users with end-to-end message latency of no more than 300 milliseconds.	Very high and extreme update rates ⁸	Dell PowerEdge server: 4 CPUs, 2.4Ghz per CPU

2.3 Caplin's benchmark tools

Benchmarking a streaming server such as Liberator in a realistic manner is a challenge, because of the need to simulate the large numbers of users and high update rates that would be encountered in the real-world business environments where the server is typically deployed. Caplin Systems has produced a suite of tools, the Benchtools, that make such benchmarks easier to set up and run. These tools are available for Caplin's customers to measure the performance of Caplin Liberator in their own environments. There is a guide to using them in the section [How to benchmark using Caplin's tools](#)¹¹.

The Benchtools suite was used to run the benchmark tests described in this document.

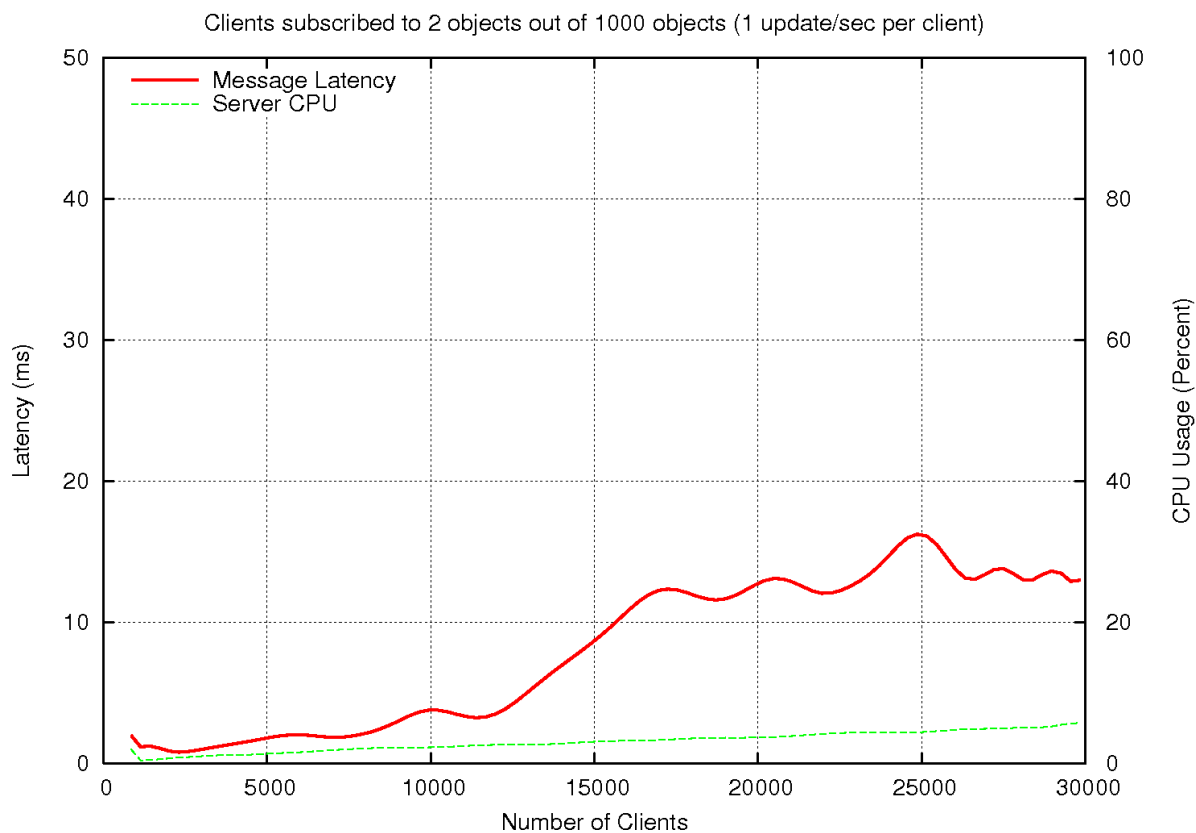
3 Test results for headline figures

The following graphs show the detailed results for the summary given in [Headline figures](#)^[4]. They are typical performance profiles for Caplin Liberator. They measure how message latency and server CPU usage changes as the number of subscribing simulated end users ("clients") increases, and hence as the message update rate to clients increases. Each graph shows a different usage profile; these profiles are defined in [Test details](#)^[10].

For the first three tests (low, moderate, and high update rates) the simulated end-users ("clients") subscribed to a random set of objects out of a pool of 1000 objects where each object was updated once every 2 seconds. The number of objects subscribed to differs between tests and therefore alters the update rate received by the clients.

3.1 Low update rates for large numbers of end users

In this test a single Liberator instance was run on [Alternative Test Machine 1](#)^[24] (HP ProLiant DL585 G2); for details of this hardware see [Test hardware](#)^[24].

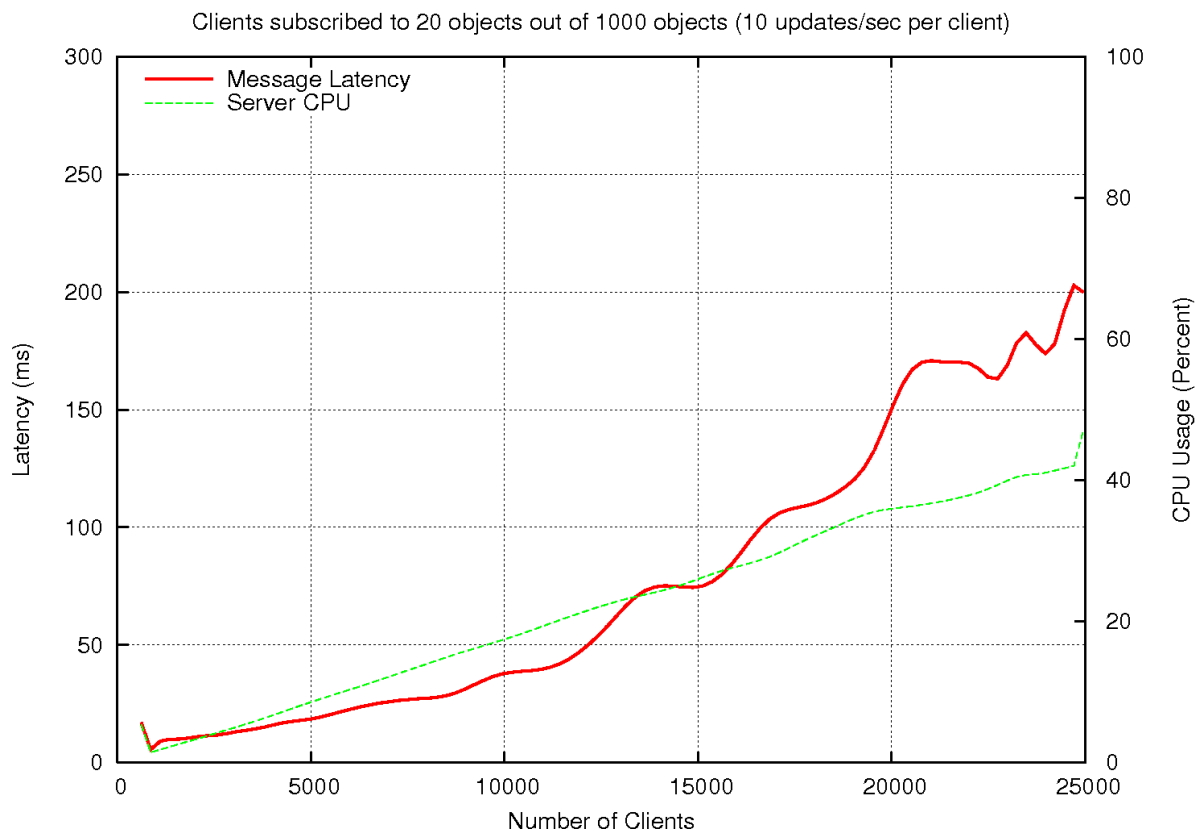


Low update rates

With this low update rate Liberator can reach its maximum of 30,000 clients with very low latency. For up to 10,000 users latency is under 5ms. Even with 30,000 users server CPU usage is under 5%.

3.2 Medium update rates for large numbers of end users

In this test a single Liberator instance was run on [Alternative Test Machine 1](#) ²⁴ (HP ProLiant DL585 G2); for details of this hardware see [Test hardware](#) ²⁴.

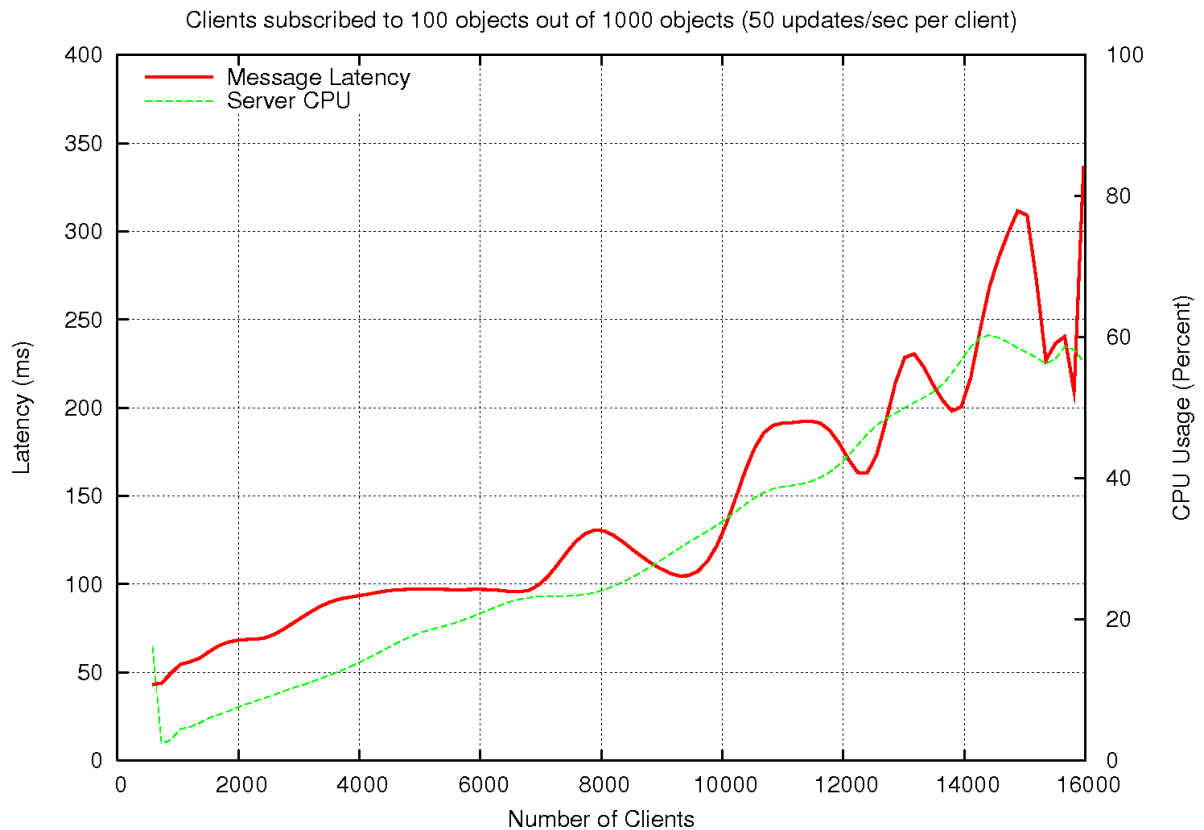


Medium update rates

With more objects subscribed to the clients receive a medium update rate. Latency is very good up to the 10,000 user mark and rises as it reaches 25,000 users. Server CPU usage is under 50% at this point.

3.3 High update rates for large numbers of end users

In this test a single Liberator instance was run on [Alternative Test Machine 1](#) ²⁴ (HP ProLiant DL585 G2); for details of this hardware see [Test hardware](#) ²⁴.



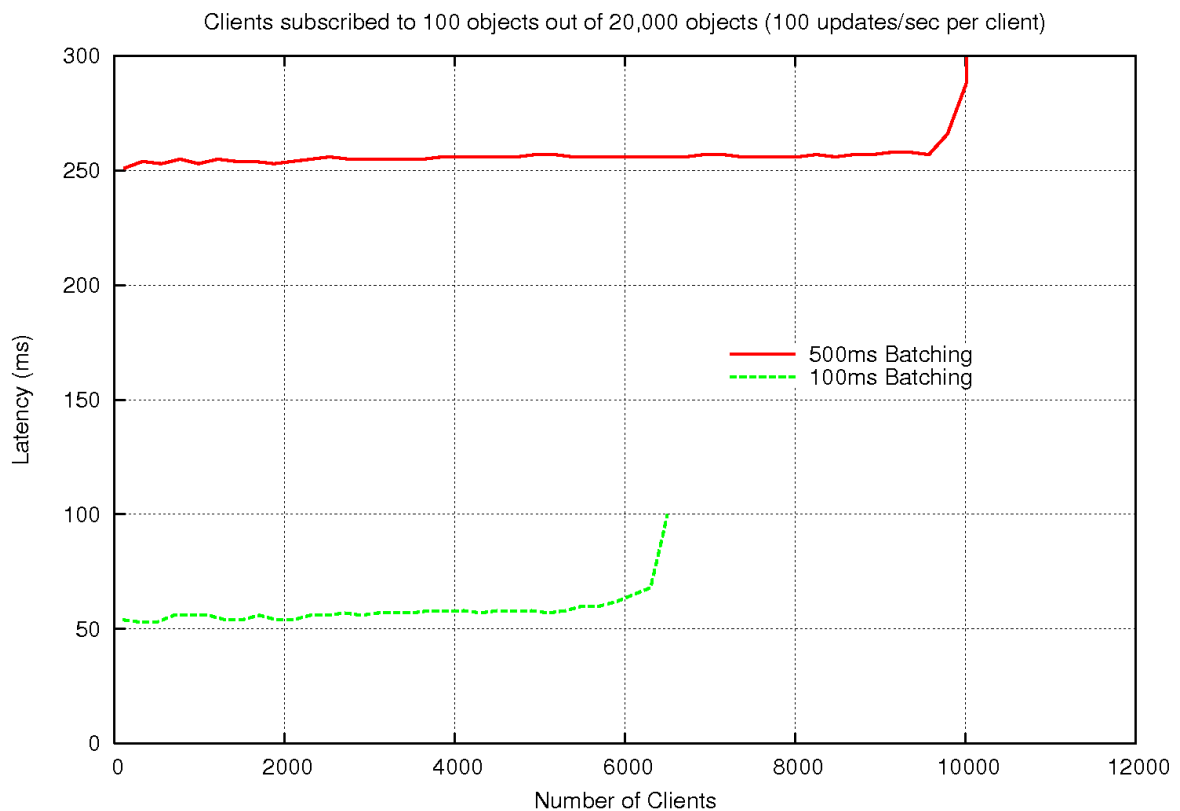
High update rates

In this test 100 objects were subscribed to by each client, giving a client update rate of 50 updates/sec. Again latency up to 10,000 users is stable, although understandably higher than previous tests. Server CPU usage goes past the 50% mark and latency increases as the test reaches 16,000 users.

3.4 Very high and extreme update rates

These two tests used a different test set up, where a single Liberator instance ran on a Dell PowerEdge server ([Test Machine 1](#)^[24] as detailed in [Test hardware](#))^[24]. In these tests a much larger pool of objects was available for clients to subscribe to. Therefore, much higher back end update rates are seen and higher client update rates are tested. These tests only plot the message latency and not the server CPU usage.

Very high update rates



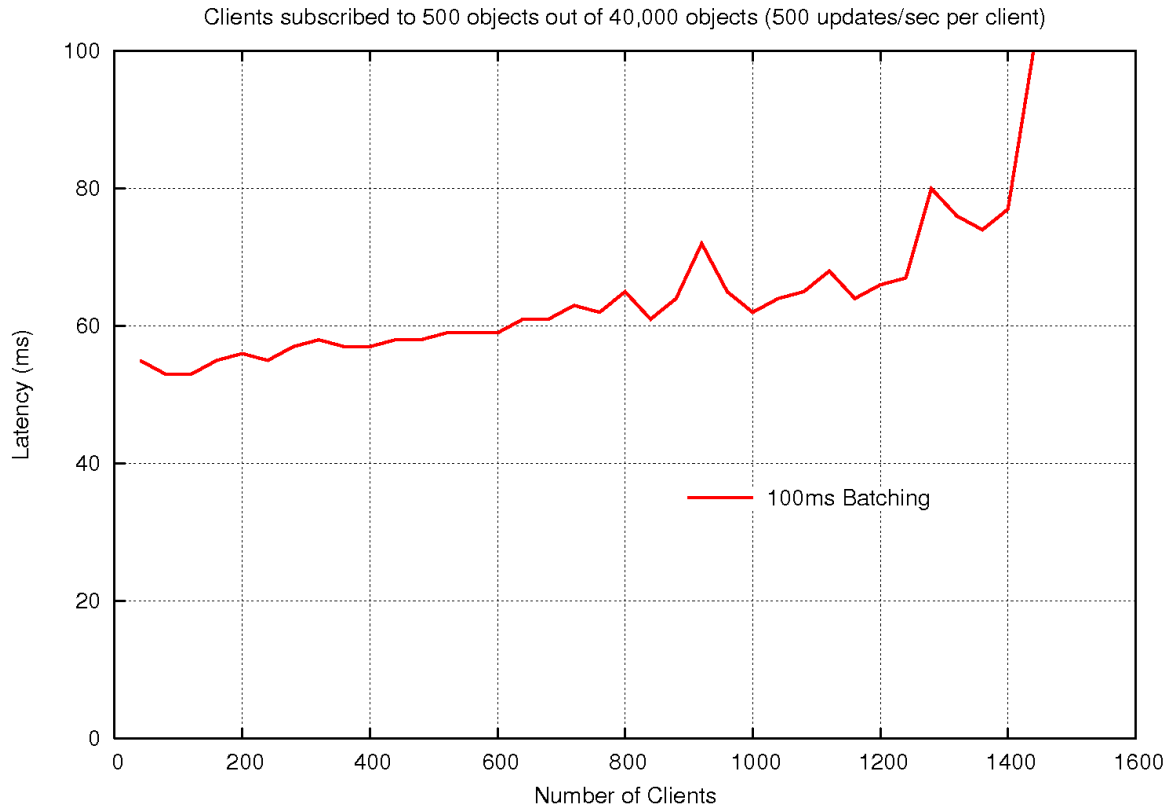
Very high update rates

In this test of very high update rates clients subscribe to 100 objects out of a pool of 20,000 objects, each updating once per second.

The red plot (solid line) shows Liberator running with a **burst-max** (batching) configuration parameter setting of 0.5 seconds. As the number of clients subscribing increases the message latency rises very slowly from about 250 milliseconds. The Liberator can handle over 10,000 clients (which is 1 million object updates per second delivered to the clients), with the message latency still below 300 milliseconds.

The green plot (dashed line) shows Liberator with the same subscription and update profile, but with the **burst-max** setting reduced to 0.1 seconds. By reducing **burst-max**, the average message latency has been substantially reduced to just over 50 milliseconds. This is at the expense of the number of subscribing clients (and hence the total overall message rate); the Liberator can now only handle about 6,300 clients (which is 630,000 object updates per second delivered to the clients).

Extreme update rates






Extreme update rates

In this test of extremely high update rates 500 objects are subscribed to by clients, choosing randomly from 40,000 objects each updating once per second. This means a back end update rate of 40,000 updates/sec and each client receiving 500 updates/sec. With 100ms batching configured latency is expected to be 50ms or more, the plot showing latency increases above this expected floor as the users increase.

3.5 Test details

The following table shows the details of the usage profiles for the "headline figures" tests.

	Low , Medium, High update rates	Very high update rates	Extreme update rates
Liberator version:	4.4	4.4	4.4
Hardware:	Alternative Test Machine 1 	Test Machine 1 	Test Machine 1 
Number of DataSources (= number of DataSource threads in Liberator):	1	2	3
Number of Liberator session threads configured:	8	3	1
Number of Clients:	Up to 30,000	Up to about 10,000	Up to about 850
Number of DataSource objects available:	1,000 objects	20,000 objects	100,000 objects
Each client subscribes (at random) to:	2, 20, 100 objects	100 objects	200 objects
Update rate on each object subscribed to:	0.5 updates/sec	1 update/sec	1 update/sec
Message Content for each update:	5 fields: One field of 13 characters (timestamp) Four fields of 5 characters each Total 33 bytes of update data Total message size: 58 bytes.	5 fields: One field of 13 characters (timestamp) Four fields of 3 characters each Total 25 bytes of update data Total message size: 50 bytes.	5 fields: One field of 13 characters (timestamp) Four fields of 3 characters each Total 25 bytes of update data Total message size: 50 bytes.
Bytes/second delivered to each client:	58, 580, 2,900 bytes/ sec	5,000 bytes/sec	10,000 bytes/sec
Liberator burst-max Setting:	0.1	0.1 and 0.5	0.1

4 How to benchmark using Caplin's tools

This section describes how to use Caplin Benchtools to measure the performance of Caplin Liberator—Caplin's highly scalable streaming distribution server—and of Caplin DataSource Adapters that integrate the Caplin Platform with third party market data platforms such as Reuters RMDS.

The Caplin Benchtools are designed to assist assessment of Caplin Platform's performance relative to other market data and trade message delivery solutions. They run with minimum hardware requirements, demonstrating the Platform's response to load until breaking point.

Caplin Benchtools impose load on the Caplin Platform by creating multiple concurrent client sessions that subscribe to real-time updating objects. They thus set up exactly the same load conditions as real end-users would when connected to Liberator over the Internet. The components are specifically designed to test persistent HTTP streaming connections, such as those established by the RTTP protocol. This provides an immediate advantage over traditional HTTP load testing tools that do not have similar scalability or flexibility to test streaming connections.

The performance of the Caplin Platform may be measured by various metrics such as average end-to-end latency between server and clients, server CPU and bandwidth requirements. These metrics provide a suitable base for comparison of Caplin against other technology vendors.

4.1 Package contents

The Benchtools package comprises two components:

- ◆ Benchsrc

Caplin's update publisher which generates updates of a configurable size and frequency.

This tool is described in more detail in the section [Test DataSource application \(Benchsrc\)](#) ¹².

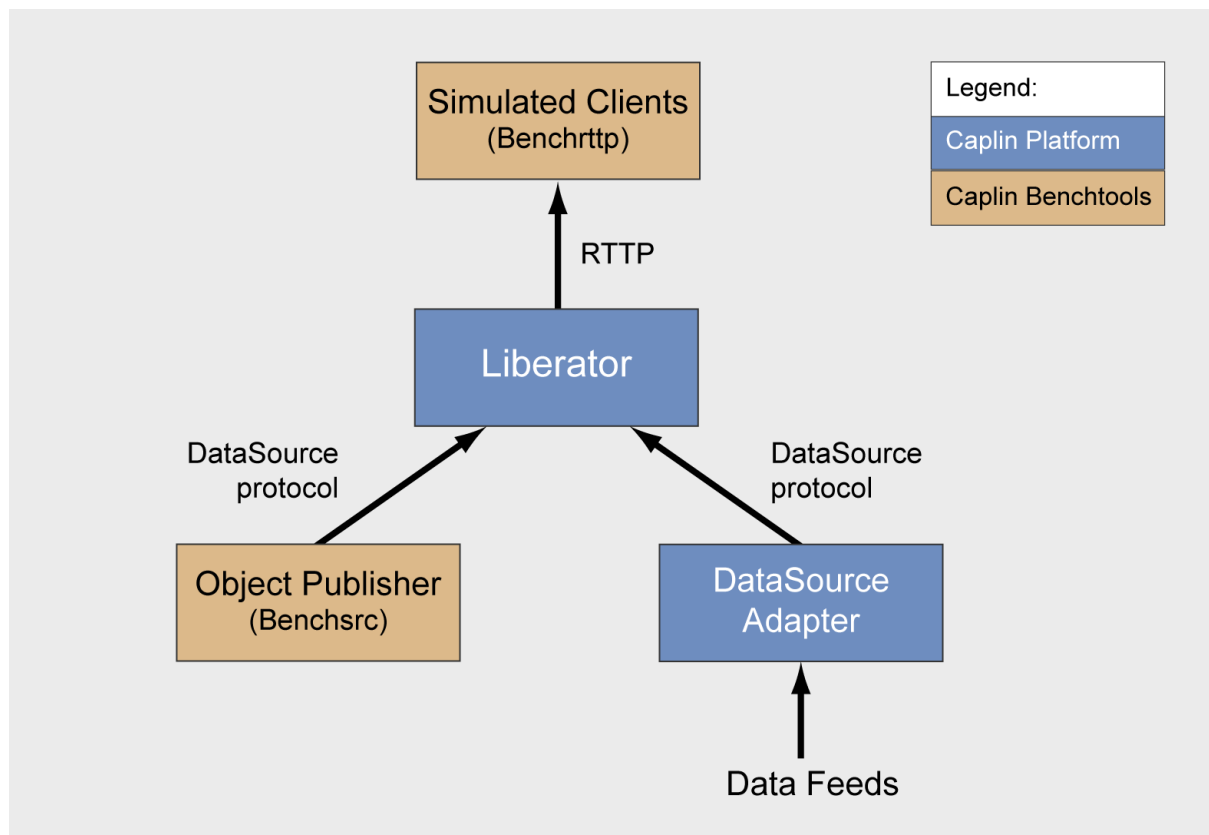
- ◆ Benchrttp

Caplin's scalable client simulator which simulates concurrent RTTP sessions.

This tool is described in more detail in the section [Test RTTP client application \(Benchrttp\)](#) ¹³.

4.2 Architecture

The diagram below indicates how the Benchtool components integrate with the Caplin Platform.



Benchtool components in the Caplin Platform

As with SL4B or SL4J clients, simulated clients use the RTTP protocol to establish a streaming connection to Liberator's HTTP port. Similar to other DataSources, Benchsrc sends updates to Liberator's DataSource port via the DataSource protocol.

The architecture above allows two test configurations:

1. Benchsrc generates updates.
2. Another publisher generates updates to a feed such as RMDS.

In both configurations Benchrttp clients consume the updates.

4.3 Test DataSource application (Benchsrc)

Caplin's Benchsrc tool is a standard DataSource application that can be configured to provide data in known formats and at known update rates. It is flexible, allowing sets of objects to be configured with different sets of fields and values, and at different update rates.

Benchsrc is used to supply data to Liberator that is subscribed to by clients. Typically the clients will be instances of Benchrttp, but since Benchsrc is a DataSource providing standard data, any StreamLink client can be used in

this setup if different kinds of tests are required. A StreamLink client could be used as a control client in a benchmark test, to perform more stringent testing of the data being supplied or as a visual client to monitor the test.

Benchsrc can act as a broadcast or active DataSource. In broadcast mode it will supply all configured data to Liberator all of the time, but in active mode only the data subscribed to by clients will be sent.

Benchsrc is designed to supply data at the configured update rate in an efficient manner, allowing high update rates to be used without the need for too much hardware. It can be controlled remotely, via a UDP control protocol, to alter the rate of the messages being published. Test runs can be automated using scripts.

Latency of updates is a key factor in benchmark testing. By default, data updates sent by Benchsrc include a millisecond timestamp which is then processed by Benchrttp (or any appropriately configured StreamLink client) allowing message latencies to be calculated.

4.4 Test RTTP client application (Benchrttp)

Benchrttp is Caplin's client simulation tool. It is designed to simulate concurrent RTTP client sessions and requires minimal hardware resources for maximal scalability. Traditional HTTP solutions and load testing tools are designed to test non-persistent, non-streaming web applications which usually involve short-lived HTTP connections. Typically these tools neither provide the functionality nor scalability required to stress test persistent connections. Benchrttp aims to provide sufficient scalability to stress the Caplin Liberator server to breaking point, with minimal hardware requirements.

Benchrttp communicates with Caplin Liberator using the proprietary RTTP protocol. RTTP is a text-based HTTP-compliant protocol which optimally distributes RTTP objects, such as record images and updates, to web-based clients. Benchrttp simulates multiple RTTP clients, where each simulated client makes a *separate* connection to the Liberator server, just as though it were a real client. A simulated client exactly conforms to the RTTP protocol used by the other Caplin client-side APIs, such as StreamLink for Java™ or StreamLink for Browsers.

For validation purposes, Benchrttp can output raw RTTP data to verify that every update published at the source reaches the simulated clients.

To sustain as many concurrent RTTP clients as possible, Benchrttp uses a custom implementation of RTTP that is suited to supporting large numbers of concurrent client sessions, rather than offering the full RTTP functionality provided by the StreamLink client API libraries.

The custom implementation provides simulated clients the necessary functions to receive data and interrogate update values:

1. Logon to Caplin Liberator obtaining a unique session ID.
2. Establish a streaming back-channel.
3. Request a configurable number of objects at random from a configurable pool of objects.
4. Receive images for all requested objects.
5. Receive a stream of object updates. A millisecond timestamp within every object update is used to calculate the message latency between Liberator and client.

Every measurement period (5 seconds by default) Benchrttp displays the aggregate latency for all updates received by all simulated clients.

Benchrttp can be controlled remotely via a UDP control protocol. This allows the number of clients to be altered and also allows logging of the statistics to be turned on and off. Test runs can be automated using scripts.

4.5 Benchmark requirements

To run a realistic benchmark of Caplin Liberator, the following hardware, network, and operating system requirements should be met.

Hardware

- ◆ Caplin Liberator
1 x Linux/Solaris server machine.
- ◆ Benchtools
2 x Linux/Solaris server machine.

The machine reserved for Caplin Liberator should match the hardware specifications of an intended production service machine.

Minimum Liberator hardware requirements:

- ◆ 2 x Dual-Core AMD Opteron™ Processor 2.4 GHz
- ◆ 4 GB Ram
- ◆ 40 MB hard disk space for Liberator installation
- ◆ 200 MB hard disk space for Liberator logs

The Benchtools may run on any almost Linux or Solaris machine. If hardware specifications are inferior to those of Liberator's box, extra machines may be required.

Network

A gigabit (Gbit) network is required between Liberator and Benchtool boxes.

Operating System

Minimum requirements for Caplin Liberator are:

- ◆ Linux:
Red Hat Enterprise Linux 4 AS/ES
- ◆ Solaris:
Solaris 8

4.6 Installation and Setup

The following sections detail how to install and setup the various components required to perform benchmark testing.

Installing Benchtools

The Caplin Benchtools are provided within a gzipped tar package and should be installed on a different machine to that running Caplin Liberator. Assuming the current directory contains the package, uncompress using the appropriate command below:

Linux:

```
$ tar xzf Benchtools-4.x.x-x-i686-pc-linux-gnu.tar.gz
```

Solaris:

```
$ uncompress Benchtools-4.x.x-x-sparc-sun-solaris2.8.tar.Z
$ tar xf Benchtools-4.x.x-x-sparc-sun-solaris2.8.tar
```

The resultant *Benchtools-4.x.x-x* directory contains the uncompressed Benchtools.

Benchsrc setup

The object publisher, Benchsrc, must be configured with the host address and DataSource port of Liberator.

The Benchsrc configuration file is *Benchtools-4.x.x-x/etc/benchsrc.conf*

Within this file, change the `addr` parameter within the `add-peer` section to the hostname or IP address of the machine that will run Liberator.

If intending to use a Liberator DataSource port other than 25000, also change the `port` parameter.

An example is shown below:

```
# benchsrc.conf

...

add-peer
    addr          myLiberator.mydomain.com
    port          25000

# Uncomment this line to operate in ACTIVE mode
# local-type      active
end-peer

...
```

Benchrttpt setup

The client simulator, Benchrttpt, must now be configured with the host address and http-port of Liberator.

The Benchrttpt configuration file is *Benchtools-4.x.x-x/etc/benchrttpt.conf*

Within this file, change the `server` parameter to the hostname or IP address of the machine that will run Liberator.

If intending to use a Liberator HTTP port other than 8080, also change the `port` parameter.

An example is shown below:

```
# benchrttpt.conf

server                myLiberator.mydomain.com
port                  8080

...
```

Liberator installation

Liberator installation is covered fully within the Liberator Administration guide. The installation commands here are provided for reference.

The 'best practice' installation technique is shown below. It uses a symbolic link from the Liberator runtime directory to a *kits* directory. If a new Liberator version is released, the kit can be installed simply by unpacking to the *kits* directory and updating the symbolic link.

1. On the Liberator box, within the current directory designated for installation (e.g. */home/caplin*), create the *kits* and *liberator1* directory:

```
$ mkdir -p kits/liberator
$ mkdir -p liberator1
```

2. Change directory to the *kits/Liberator* directory:

```
$ cd kits/liberator
```

3. Uncompress the kit:

Linux:

```
$ tar xzf /tmp/Liberator-4.4.8-1-i686-pc-linux-gnu.tar.gz
```

Solaris:

```
$ uncompress /tmp/Liberator-4.4.8-1-sparc-sun-solaris2.8.tar.Z
$ tar xf Liberator-4.4.8-1-sparc-sun-solaris2.8.tar
```

4. Navigate to the *liberator1* directory

```
$ cd ../../liberator1
```

5. Create a *latest* symbolic link.

```
$ ln -s ../kits/liberator/liberator-4.4.8-1 latest
```

6. Create the new runtime instance of Liberator within the *liberator1* directory:

```
$ ln -s latest/bin
$ ln -s latest/doc
$ ln -s latest/htdocs
$ ln -s latest/include
$ ln -s latest/lib
$ cp -r latest/etc
$ mkdir users var
```

Upgrading Liberator

If a new version of the Liberator package is released:

1. Unpack the package within the *kits/liberator* directory:

```
$ tar xzf /tmp/Liberator-4.4.8-6-i686-pc-linux-gnu.tar.gz
```

2. Update the symbolic link inside *liberator1*:

```
$ rm latest
$ ln -s ../kits/liberator/liberator-4.4.8-6 latest
```

3. Remove the contents of the *liberator1/users* directory:

```
$ rm ./users/*
```

Liberator setup

Liberator should now be configured to accept a connection from Benchsrc on the `datasrc-port` and accept client connections on the `http-port`:

The primary Liberator configuration file is `liberator1/etc/rtttd.conf`

1. If a Liberator HTTP port other than 8080 is intended, change the `http-port` parameter.
2. If a Liberator DataSource-port other than 25000 is intended, change the `datasrc-port` parameter.
3. Benchsrc will connect to Liberator on ID 1 corresponding to the connection declared in the first `add-peer` section. Replace 'demosrc' with 'benchsrc' for both the `remote-name` and `label` parameter.
4. Within the `add-data-service` change the `service-name` to 'benchsrv' and rename the `label` from 'demosrc' to 'benchsrc'.
5. Conflation (or throttling of updates) should be turned off for benchmarking tests as the update frequency should be controlled by the Benchsrc DataSource rather than Liberator. Turn off conflation by adding the lines:

```
# default conflation off
object-throttle-off
```

6. The logging of every packet which travels through Liberator is not necessary for benchmarking or for a production service. Turn off packet logs by adding the line:

```
# disable packet logs
datasrc-pkt-log /dev/null
```

7. To allow clients to log on to Caplin Liberator the OpenAuth authentication module should be used. Comment out the line: 'auth-module xmlauth' and uncomment the 'auth-module openauth' line:

```
# openAuth module
#auth-module          xmlauth
#auth-module          cfgauth
auth-module           openauth
```

8. Each test case may require a specific batching period. The concept of batching (or "bursting") is explained in the [Frequently asked questions](#)^[49] section ([What burst configuration should we use?](#)^[49]). For now add the line:

```
# batching period
burst-max 0.5
```

9. Liberator is based upon a multi-threaded architecture that exploits multiple CPUs. The number of threads used to support client sessions is configured by the parameter `threads-num`. See the [Frequently asked questions](#) ^[49] section for more details on thread configuration ([How many threads should we configure?](#) ^[49]). Change the `threads-num` parameter to '4' for now:

```
# number of session threads
threads-num                4
```

5 How Caplin's benchmark tests were conducted

The following sections describe the [test method](#)^[20] used, give information about the [test configurations](#)^[21], and detail the [test software](#)^[23], [test hardware](#)^[24], and [the network](#)^[25] used.

5.1 Test method

Approach

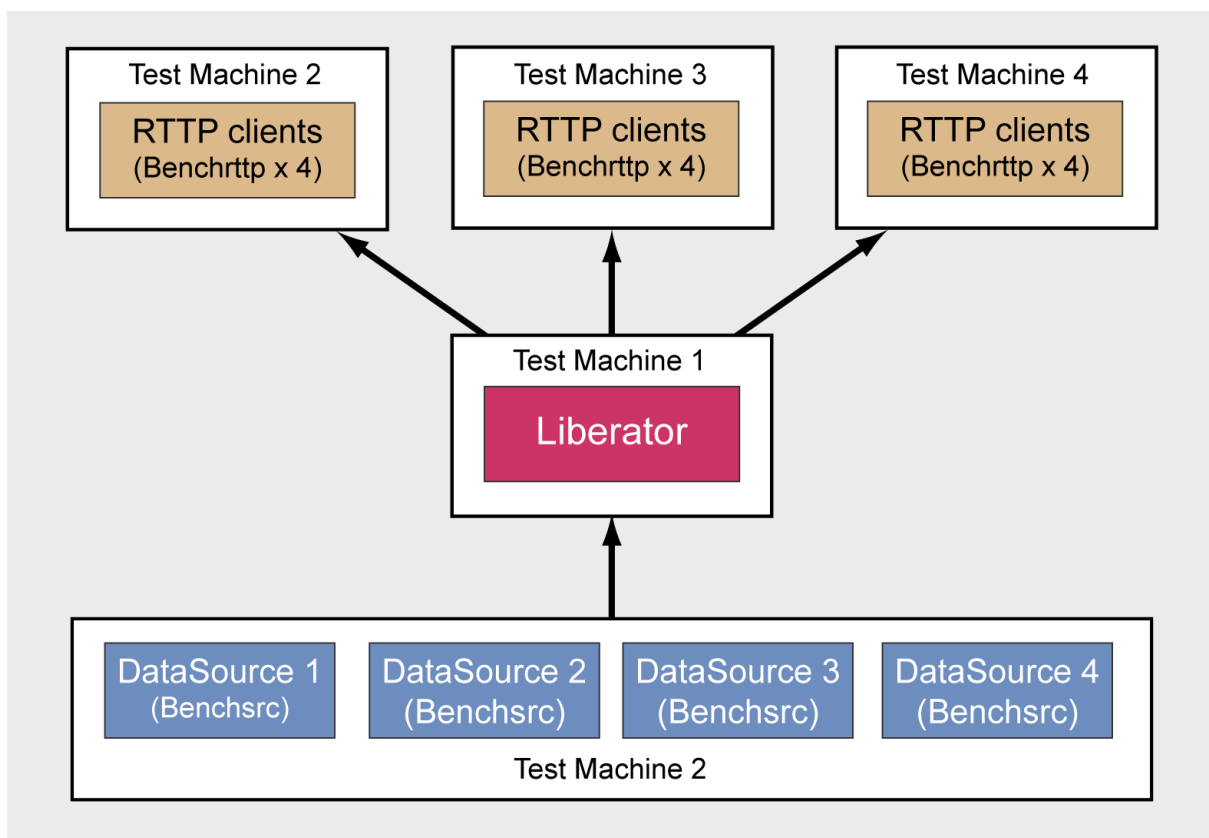
Although the benchmark consisted of several different tests, they all followed a similar method.

Each test consisted of one or more DataSources publishing messages into a Liberator which pushed the messages out to a set of subscribing clients through RTTP connections. Each subscribed object was updated at a regular rate by the supplying DataSource (usually once a second). Additional clients were logged on to the Liberator throughout the test run to determine the effect of increasing the load on the Liberator.

Test setup

The multiple RTTP client connections were simulated using a specially written application called Benchrttp. The DataSource application supplying the Liberator (Benchsrc) was also specially written. Both Benchrttp and Benchsrc are controllable using a UDP command protocol, thus allowing message rates and number of clients to be remotely managed using scripts.

The following diagram shows the hardware configuration used for the tests, and how the test software was distributed across the hardware.



Liberator 4.4 benchmark - Hardware and software configuration

The Liberator server was hosted on a machine of its own (Test Machine 1). A separate machine (Test Machine 2) was used to host up to four DataSources (Benchsrc) feeding the Liberator. Test Machine 2 and two further machines, Test Machines 3 and 4, hosted up to four instances each of Benchrttp.

The number of Benchrttp instances required for each test was determined by the maximum number of simulated clients needed to run the test; enough Benchrttp resource was required to ensure that Liberator limits could be reached before any limits imposed by Benchrttp.

For more detailed information about the hardware and software used to run the tests see the following sections ([Test software](#)^[23], [Test hardware](#)^[24], and [The network](#)^[25]).

5.2 Test configurations

Most of the tests determined the average latency of update messages delivered to clients, against the number of subscribing (logged on) clients. This gave a measure of how the Liberator performed as the client update rate increased, for the following reason.

Each client subscribed to a number of objects (the number varied according to the test). Because each subscription required Liberator to update those objects on the client in line with the updates from the DataSources, increasing the number of logged on clients had the effect of proportionally increasing the Liberator's client update load.

For example, if each client subscribed to 100 objects and there were 2,000 clients logged on to the Liberator, then the Liberator would have to supply $100 \times 2,000 = 200,000$ object updates per second to all the clients.

The latency of the client update messages was measured at the client end. In all cases measurements were taken from the application log files and results plotted.

Where the test required no more than 4 instances of Benchrttp, these were hosted on the same machine (Test Machine 2) as the DataSources (Benchsrc). This allowed accurate latency measurements to be made between the DataSources and the simulated clients run on that machine. Where a test required more than 4 instances of Benchrttp, the additional instances were run on Test Machine 3 and Test Machine 4, but latency was still measured on Test Machine 2.

Each set of test results in also contains details of the data used for the test and the relevant configuration parameters of the Liberator.

The most important Liberator parameters affecting performance are:

- ◆ The number of DataSource threads
- ◆ The number of session threads (**threads-num** configuration parameter)
- ◆ The value of the **burst-max** configuration parameter

Many of the tests were performed with varying numbers of DataSource threads and/or session threads in order to determine the impact of the thread settings on Liberator performance.

DataSource threads

Liberator 4.4 uses a separate thread for each DataSource connection. Since each instance of the Benchsrc DataSource only supports one connection to Liberator, the number of Liberator DataSource threads in a test was determined by the number of Benchsrc instances used.

5.3 Test software

Caplin Liberator server

The tests were run against a Caplin Liberator 4.4 server. The server configuration only had a few changes from the default settings, as follows:

Log Cycling	This was modified to prevent the very large amounts of data being processed using up too much disk space. Packet logging was turned off completely.
System Max Files	This was increased to 32768 to allow high numbers of clients to connect.
Object Throttling	The default object throttling of 1 second was turned off for benchmarking. This allowed all clients to receive all the messages they are subscribed to.
Threads	The numbers of DataSource threads and session threads were adjusted according to the needs of the individual tests. Many of the tests were repeatedly run with different numbers of threads in order to determine the impact of the thread settings on Liberator performance. See also DataSource threads ^[22] in Test configuration ^[21] .
Burst Settings	The default setting for burst-max is 0.5. This was altered to 0.1 for some of the tests. The Burst Performance ^[37] test used several values for this parameter.

Operating system

The operating system used on all the server hardware was Linux – Red Hat® Enterprise Linux® version 4. This was a standard configuration with only one significant change; the number of open file descriptors was increased to allow Liberator to support high numbers of client connections. This is detailed in the Caplin Liberator Administration Guide.

Test DataSource application (Benchsrc)

The [Benchsrc](#) ^[12] test tool was configured to produce updates to sets of objects at a known message rate. Benchsrc is a single threaded application, so for those tests where the Liberator was configured to use multiple DataSource threads there was an instance of Benchsrc for each DataSource thread.

Test RTTP client application (Benchrttp)

The [Benchrttp](#) ^[13] test tool was configured to request sets of objects that were published by the DataSource. It measured the number of messages being received and also the latency of the messages. The simulated clients all used RTTP Type 2 connections – this type of connection is HTTP tunneled.

5.4 Test hardware

The benchmark tests used up to 4 machines as shown in the diagram in [Test method](#)^[20]. Caplin Liberator ran on a dedicated machine. The test RTTP client processes (Benchrttp) ran on up to 3 separate machines to spread the load. The test DataSources (Benchsrc) ran on one of the Benchrttp machines (Test Machine 2).

Liberator server

Test Machine 1

Vendor:	Dell™
Model:	PowerEdge™ SC1435
Processors:	2 x Dual core AMD Opteron™, 2.4GHz per core, total 4 CPUs
Memory:	4 GBytes
Operating system:	Redhat Linux Enterprise Linux 4 – 64 bit (Kernel 2.6.9)

Alternative Test Machine 1

Vendor:	HP ProLiant DL585 G2
Model:	DL585 G2
Processors:	4 x Dual core AMD Opteron™, 2.8GHz per core, total 8 CPUs
Memory:	8 Gbytes
Operating system:	Redhat Linux Enterprise Linux 4 – 64 bit (Kernel 2.6.9)

DataSource servers

Test Machine 2

Vendor:	Dell
Model:	PowerEdge SC1425
Processors:	2 x Hyper-Threading Intel® Xeon®, 3.0GHz per processor, total 2 CPUs
Memory:	4 Gbytes
Operating system:	Redhat Linux Enterprise Linux 4 – 32 bit (Kernel 2.6.9)

RTTP clients

Test Machine 2

(This was the same machine as that hosting the DataSource servers.)

Vendor: Dell
Model: PowerEdge SC1425
Processors: 2 x Hyper-Threading Intel® Xeon®,
3.0GHz per processor, total 2 CPUs
Memory: 4 Gbytes
Operating system: Redhat Linux Enterprise Linux 4 – 32 bit
(Kernel 2.6.9)

Test Machine 3

Vendor: Dell
Model: PowerEdge SC1425
Processors: 2 x Hyper-Threading Intel® Xeon®,
3.0GHz per processor, total 2 CPUs
Memory: 4 Gbytes
Operating system: Redhat Linux Enterprise Linux 4 – 32 bit
(Kernel 2.6.9)

Test Machine 4

Vendor: Dell
Model: PowerEdge SC1435
Processors: 2 x Dual core AMD Opteron, 2.4GHz per core,
total 4 CPUs
Memory: 4 GBytes
Operating system: Redhat Linux Enterprise Linux 4 – 64 bit
(Kernel 2.6.9)

5.5 The network

The test machines were set up on a Gigabit network which was used solely for transmitting the test data. Any control messages or terminal access used a separate network.

6 Results of additional tests

The following sections show the results of some additional detailed performance tests carried out on Liberator:

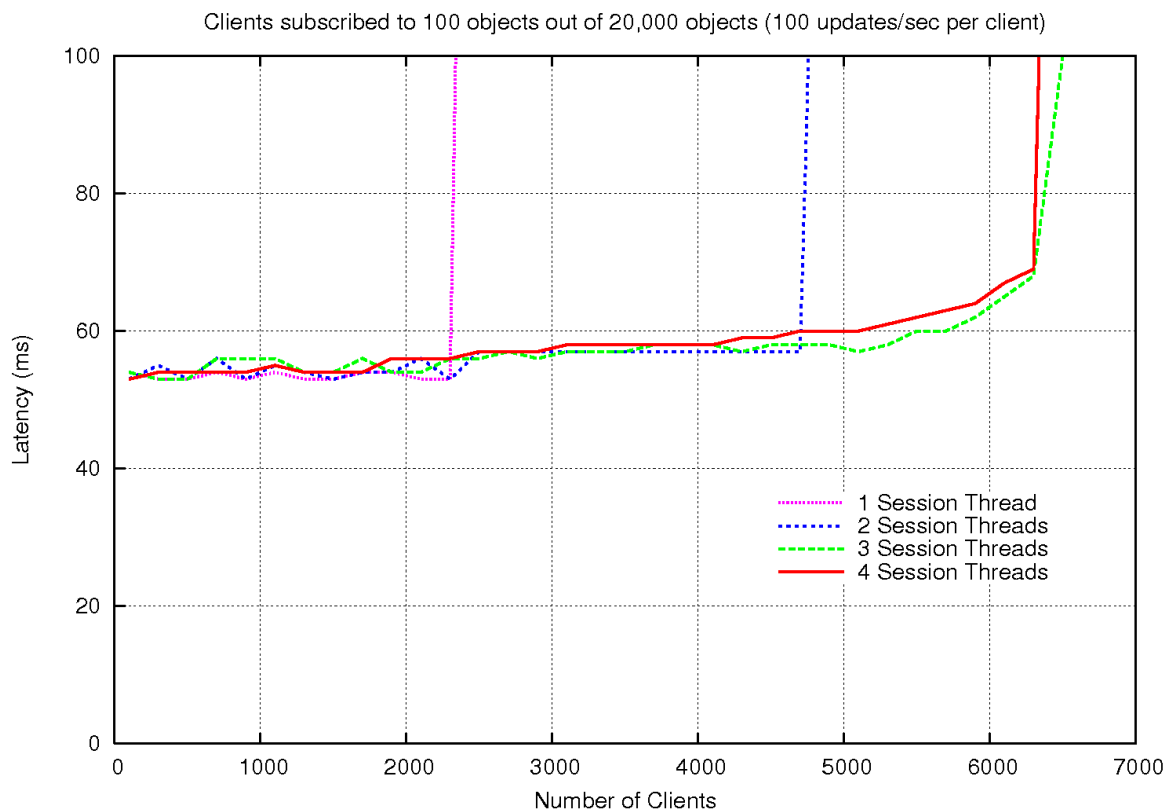
- ◆ The effect of improvements to how threads are implemented in Liberator 4.4, most noticeably the introduction of multiple threads to handle input from DataSources.
- ◆ The effect of burst configuration (batching)
- ◆ The effect of message size
- ◆ The effect of failover
- ◆ Performance improvement relative to Liberator 4.2

6.1 Effect of session threads on performance

Liberator is a multi-threaded application, it uses a number of threads to handle incoming data from DataSources and a number of threads to handle client sessions, this section shows how the number of session threads has an effect on the performance of Liberator. This and the subsequent section on DataSource threads show that for a given usage profile there is an optimum thread configuration.

Low backend data set

This test demonstrates how Liberator performs when there are a large number of updates per second to be delivered to clients compared with the rate at which DataSource updates arrive at the Liberator. This can happen when the number of objects available for clients to subscribe to is relatively low. As more clients use the system the number of updates coming into the system will reach a peak and not increase further. It measures how message latency changes as the number of subscribing clients is increased. The graph shows the results of test runs for a Liberator configured with from one to four session threads.



Session threads
Low backend data set

The graph shows that, provided there are enough session threads allocated (and enough available CPUs to run the threads in), increasing the number of subscribing clients has a relatively small impact on message latency.

When there is just one session thread, the message latency is around 53 milliseconds, until, when there are about 2,300 subscribing clients, the CPU running the thread becomes fully loaded and as a consequence the latency increases rapidly when just a few more clients are added.

When there are two session threads configured the number of clients the Liberator can support more than doubles to around 4,700, with a small linear increase in message latency. Once again, beyond this point message latency increases rapidly as the CPUs running the two threads become fully loaded.

Adding a third session thread extends the number of supportable clients to around 6,300. Since the machine running the Liberator only has four CPUs, adding a fourth session thread gives very little improvement, because the new thread must share a CPU with the DataSource threads and the main Liberator thread.

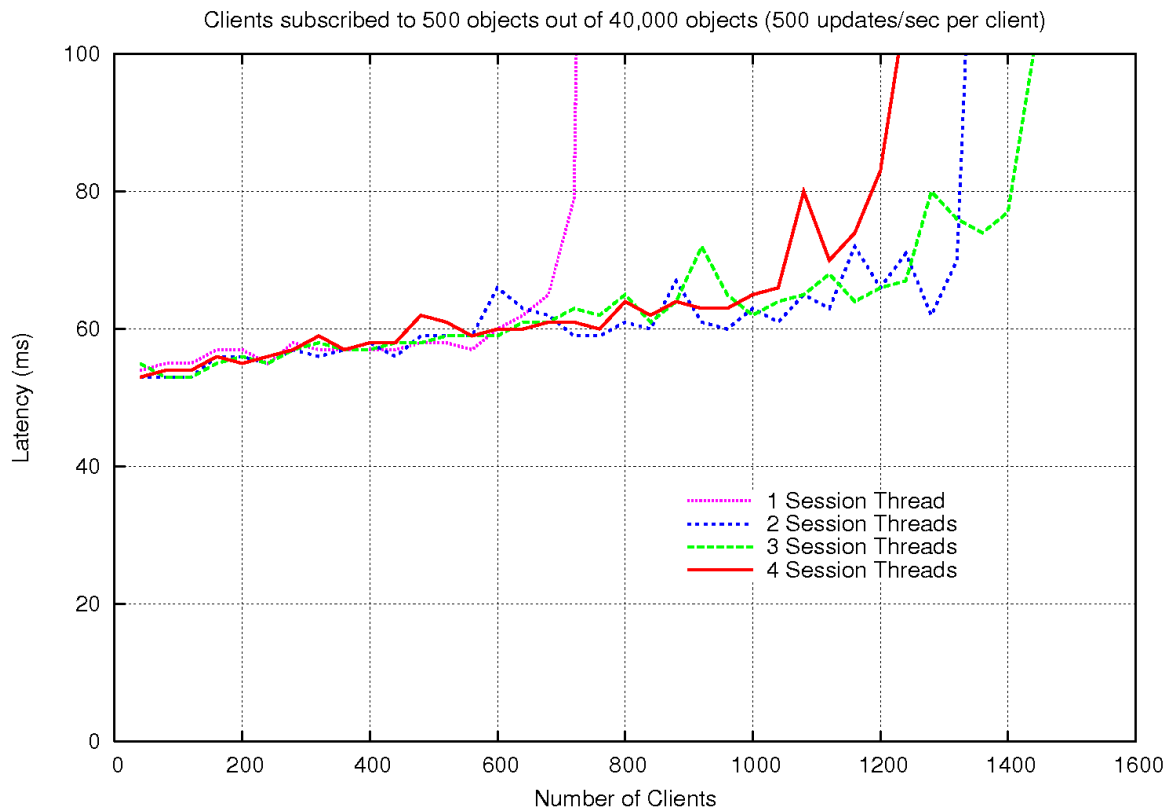
Test details

The DataSources only supplied updates at a relatively low rate (20,000 updates per second, which is 20,000 objects subscribed to with each object updated once per second). The Liberator was configured accordingly with two DataSource threads (and hence two instances of Benchsrc to supply the updates).

Liberator version:	4.4
Number of DataSources (= number of DataSource threads in Liberator):	2
Number of Liberator session threads configured:	1 – 4
Number of Clients:	From 100 to over 6,000
Number of DataSource objects available:	20,000 objects (across minimum 200 clients)
Each client subscribes (at random) to:	100 objects
Update rate on each object subscribed to:	1 update/sec
Message Content for each update:	5 fields: One field of 13 characters (timestamp) Four fields of 3 characters each Total 25 bytes of update data Total message size: 50 bytes.
Bytes/second delivered to each client:	5,000 bytes/sec
Liberator burst-max Setting:	0.1

Medium backend dataset

This test shows the effect on Liberator performance of changing the number of session threads.



Session threads Medium backend data set

The test results show that, subject to not reaching the overall CPU limit on the Liberator server, adding more session threads improves performance.

The Liberator was configured to use three DataSource threads, so it could easily cope with all the updates originating from the DataSources. Increasing the number of session threads from one to two almost doubles the effective maximum number of clients (from around 700 to around 1300). With three session threads there is a further (but smaller) increase in performance.

However, with four session threads the performance drops, because the four CPU limit of the Liberator machine has been reached. The four CPUs can accommodate three session threads plus three DataSource threads, but the addition of another session thread yields no benefit.

Test details

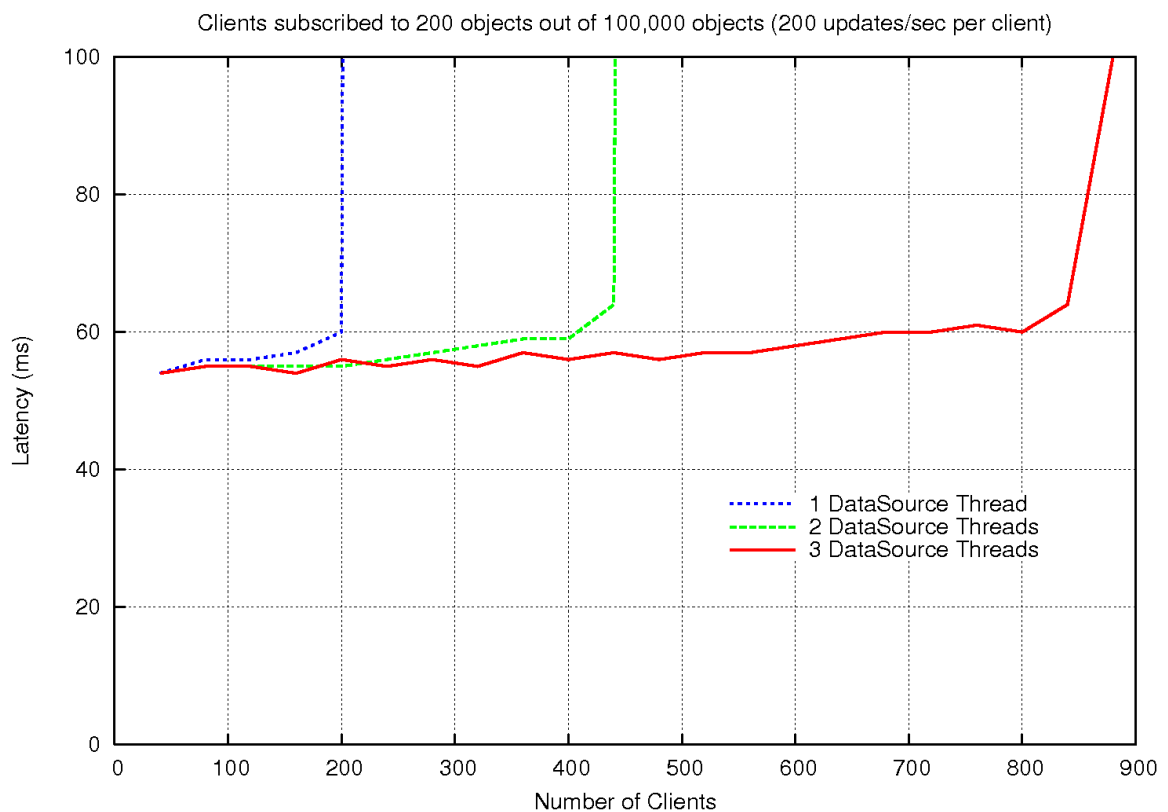
Liberator version:	4.4
Number of DataSources (= number of DataSource threads in Liberator):	3
Number of Liberator session threads configured:	1 – 3
Number of Clients:	From 40 to over 1,480
Number of DataSource objects available:	40,000 objects (across minimum 200 clients)
Each client subscribes (at random) to:	500 objects
Update rate on each object subscribed to:	1 update/sec
Message Content for each update:	5 fields: One field of 13 characters (timestamp) Four fields of 3 characters each Total 25 bytes of update data Total message size: 50 bytes.
Bytes/second delivered to each client:	25,000 bytes/sec
Liberator burst-max Setting:	0.1

6.2 Effect of DataSource threads on performance

Liberator uses a thread for each DataSource. It can also be configured for multiple connections to the same DataSource, which means each connection is treated as a separate DataSource and therefore has its own thread. This section shows how the number of threads used for DataSources has an effect on the performance of Liberator.

High backend dataset

This test demonstrates how Liberator performs when handling a high rate of updates from the DataSources.



DataSource threads
High backend data set

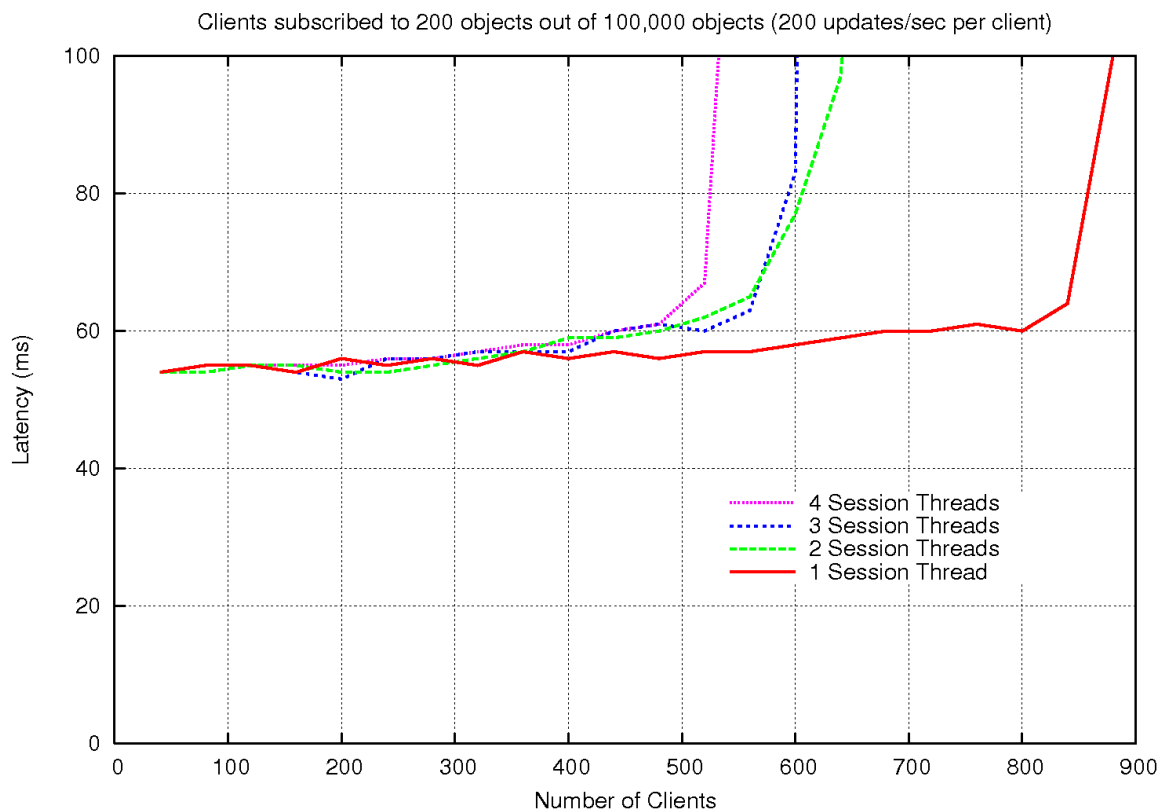
The graph shows that when there is a very high rate of updates from the DataSource(s), the more DataSource threads running in the Liberator the better.

Test details

Liberator version:	4.4
Number of DataSources (= number of DataSource threads in Liberator):	1 – 3
Number of Liberator session threads configured:	1
Number of Clients:	From 40 to about 850
Number of DataSource objects available:	100,000 objects
Each client subscribes (at random) to:	200 objects
Update rate on each object subscribed to:	1 update/sec
Message Content for each update:	5 fields: One field of 13 characters (timestamp) Four fields of 3 characters each Total 25 bytes of update data Total message size: 50 bytes.
Bytes/second delivered to each client:	10,000 bytes/sec
Liberator burst-max Setting:	0.1

High backend dataset - adding session threads

This test also demonstrates how Liberator performs when handling a high number updates from the DataSources. In this test the number of DataSources was kept constant at 3, and each plot is for a different number of session threads.



DataSource threads High backend data set with more session threads

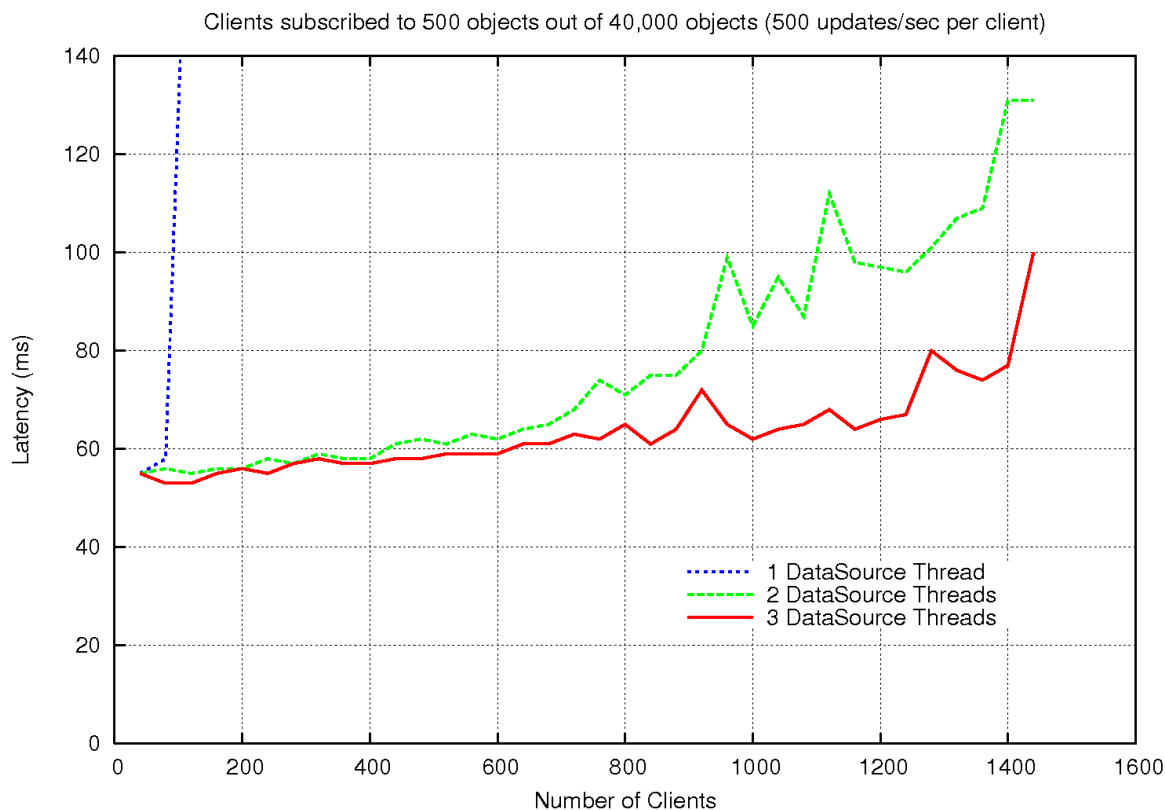
The graph shows that, on a four CPU machine with a very high rate of updates from the DataSource, adding extra session threads *decreases* performance, because the CPUs are already almost fully loaded. This result is not only because all the CPUs are working hard on DataSource threads, but as session threads increase there is a small extra load on the DataSource threads to feed them. So, with a potentially very high backend update rate, the extra load added by handling more session threads limits the capacity.

Test details

Liberator version:	4.4
Number of DataSources (= number of DataSource threads in Liberator):	3
Number of Liberator session threads configured:	1 - 4
Number of Clients:	From 40 to about 850
Number of DataSource objects available:	100,000 objects
Each client subscribes (at random) to:	200 objects
Update rate on each object subscribed to:	1 update/sec
Message Content for each update:	5 fields: One field of 13 characters (timestamp) Four fields of 3 characters each Total 25 bytes of update data Total message size: 50 bytes.
Bytes/second delivered to each client:	10,000 bytes/sec
Liberator burst-max Setting:	0.1

Medium backend dataset

This test shows the effect on Liberator performance of changing the number of DataSource threads.



DataSource threads Medium backend data set

The high rate of updates from the DataSources (30,000 updates per second once 80 clients have subscribed) means that Liberator cannot cope when it has only one DataSource thread configured. As more users subscribe the total update rate from the DataSources increases until the thread processing limit is reached at around 80 clients. After this the message latency rapidly increases as more clients subscribe.

When two DataSource threads are configured, Liberator can easily process the 40,000 updates per second. The limiting factor on performance is then the number of session threads (in this test the number of session threads remains constant at three). The graph shows that adding a third DataSource thread gives little performance improvement, since two DataSource threads can easily handle the updates arriving from the DataSources.

Test details

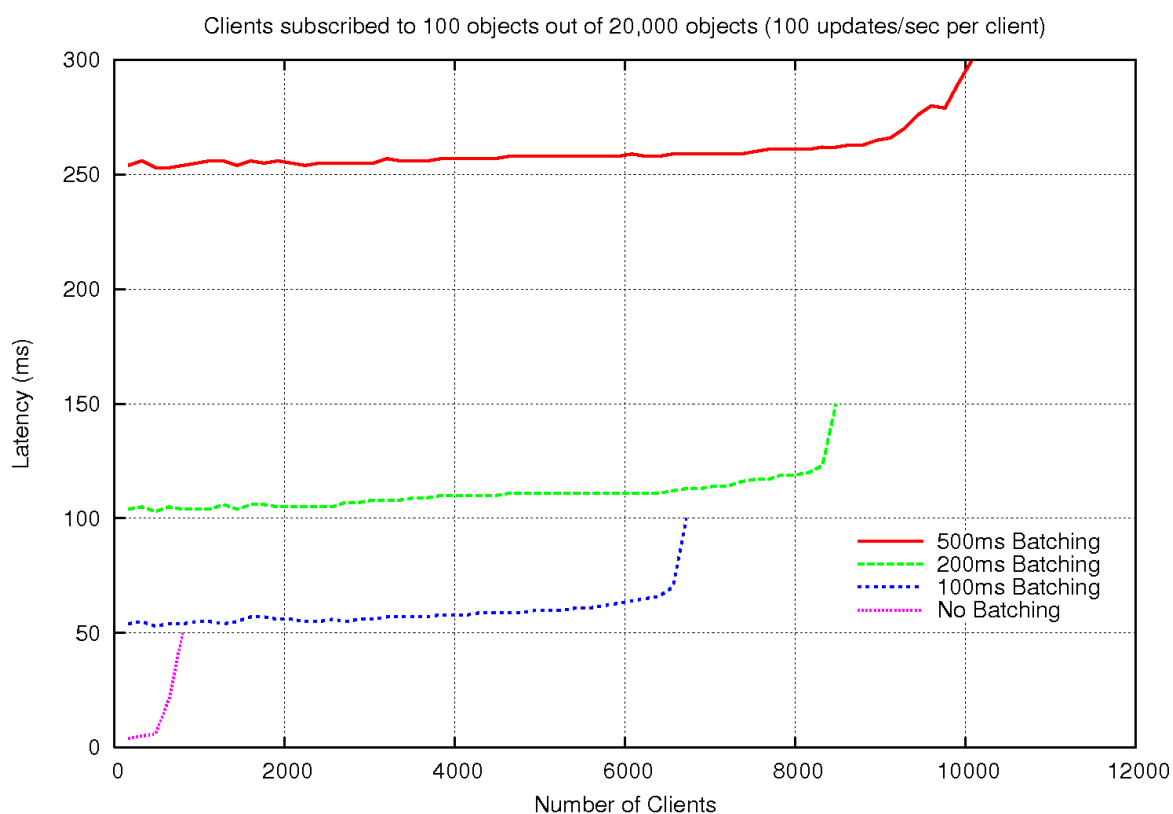
Liberator version:	4.4
Number of DataSources (= number of DataSource threads in Liberator):	1 – 3
Number of Liberator session threads configured:	3
Number of Clients:	From 40 to over 1,480
Number of DataSource objects available:	40,000 objects (across minimum 200 clients)
Each client subscribes (at random) to:	500 objects
Update rate on each object subscribed to:	1 update/sec
Message Content for each update:	5 fields: One field of 13 characters (timestamp) Four fields of 3 characters each Total 25 bytes of update data Total message size: 50 bytes.
Bytes/second delivered to each client:	25,000 bytes/sec
Liberator burst-max Setting:	0.1

6.3 Effect of burst configuration (batching) on performance

Liberator can be configured to batch messages together before writing them to the network. This feature is called bursting as it is designed to smooth a peak in data rates, but it is also relevant to constant update rates as used in these tests. Bursting can increase the scalability of Liberator as it makes better use of the network and also results in less CPU work. The trade off is that pausing to batch updates together adds latency to the messages. As can be seen from the results in these tests, average latency is generally half of the batch time configured.

Low backend dataset

This test shows the effect on performance of Liberator's **burst-max** configuration parameter.



Burst configuration
Low backend data set

The test results show that as **burst-max** is increased Liberator can handle higher numbers of users and achieve higher rates of client updates, at the expense of higher message latency. The graph also shows that even a small **burst-max** setting (0.1 sec) can achieve far higher message rates than when bursting is not enabled at all (**burst-max** = 0 sec).

With a higher **burst-max** setting more client updates are batched together to send in a single packet. This batching causes a delay in the packet being sent, but has the effect of making better use of the network, so the overall message throughput is increased, with a corresponding trade off of increased latency.

Note: For higher DataSource update rates (for example 40,000 object updates/sec, as opposed to the 20,000 object updates/sec in this test), it has been found that the extra performance gain from increasing **burst-max** above 0.1 sec is relatively small.

Test details

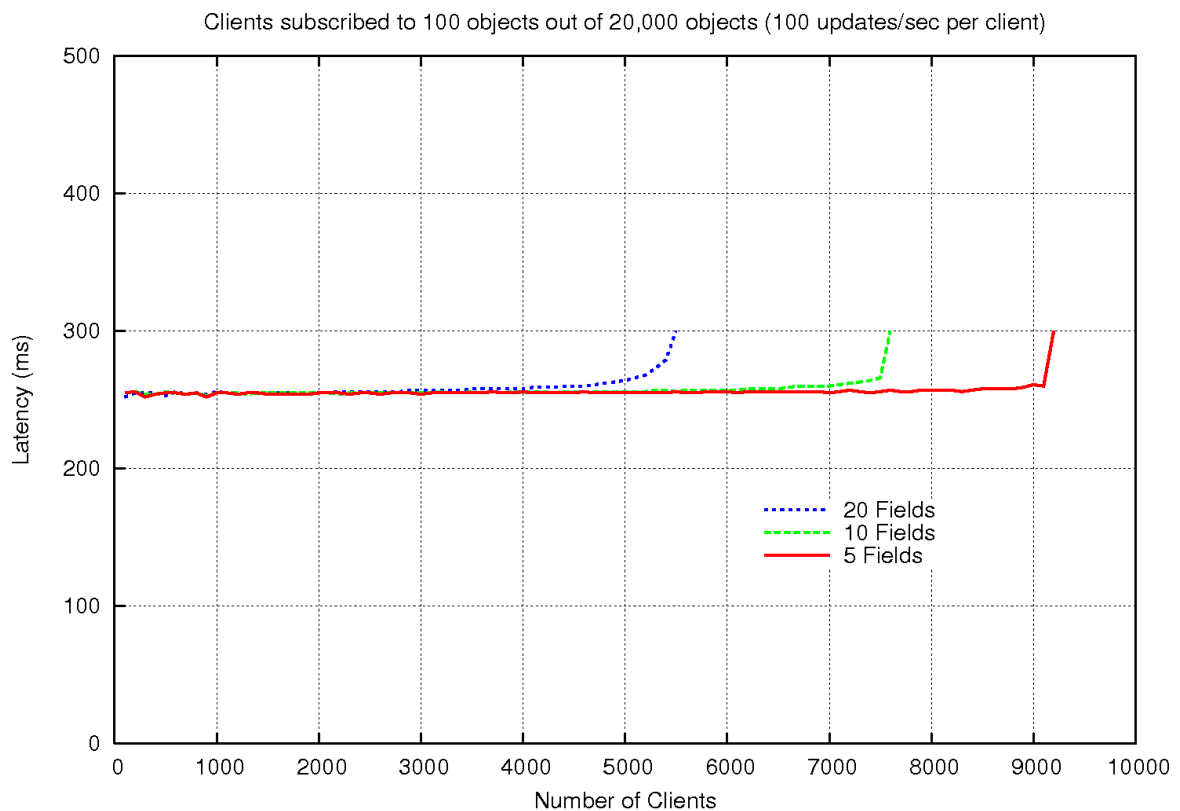
Liberator version:	4.4
Number of DataSources (= number of DataSource threads in Liberator):	1
Number of Liberator session threads configured:	4
Number of Clients:	From 40 to over 2,400 (depending on the burst-max setting)
Number of DataSource objects available:	20,000 objects
Each client subscribes (at random) to:	100 objects
Update rate on each object subscribed to:	1 update/sec
Message Content for each update:	5 fields: One field of 13 characters (timestamp) Four fields of 3 characters each Total 25 bytes of update data Total message size: 50 bytes.
Bytes/second delivered to each client:	5,000 bytes/sec
Liberator burst-max Setting:	0.0, 0.1, 0.2, 0.5 sec

6.4 Effect of message size on performance

This section investigates the effect that the message has on performance. The tests vary the size of the message and the number of fields within the message.

Overall size of a message

This test shows the effect on Liberator performance of increasing the size of the update messages. In each successive test run the size of each message was increased by adding more fields of the same size.



Overall size of a message

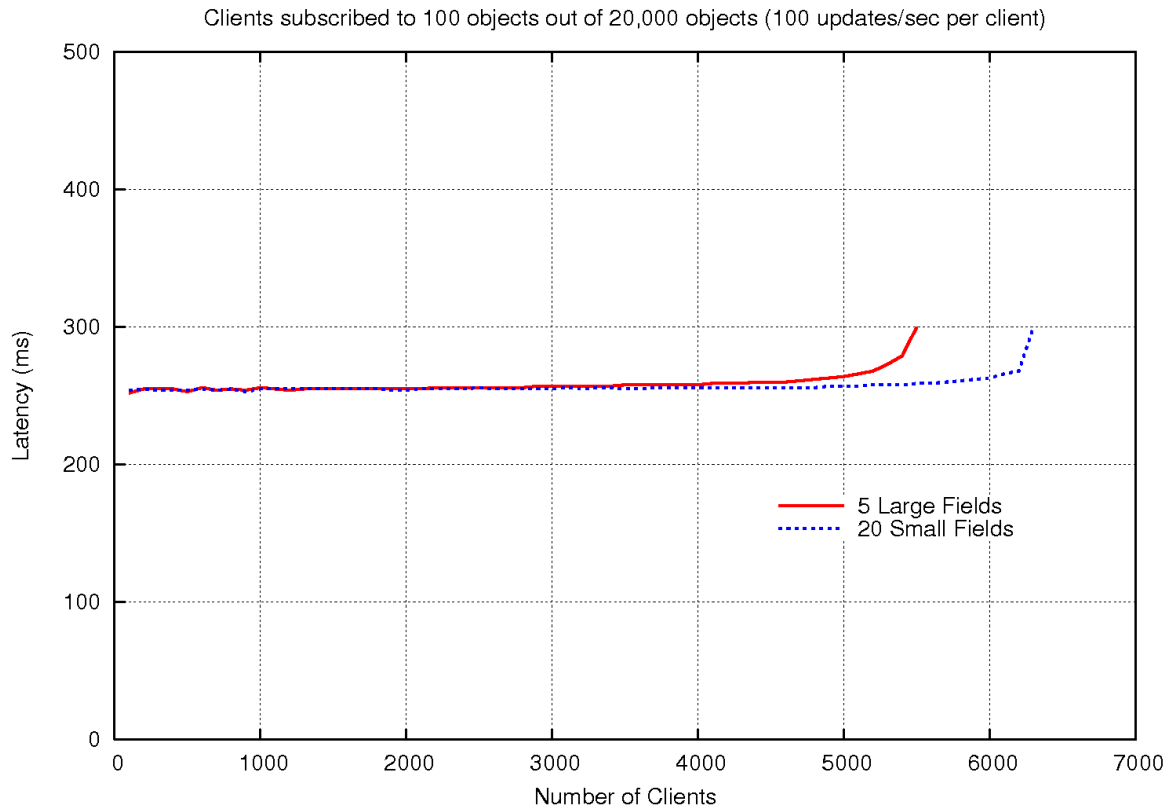
Increasing the size of each update message by sending more fields decreases Liberator's maximum effective message rate. For example, the test results show that increasing the message size by four times, more than halves the maximum effective message rate (as shown by the reduction in the maximum number of clients that can subscribe, from about 9,100 to about 4,500).

Test details

Liberator version:	4.4
Number of DataSources (= number of DataSource threads in Liberator):	1
Number of Liberator session threads configured:	4
Number of Clients:	From 100 to over 9,000 (depending on message size)
Number of DataSource objects available:	20,000 objects
Each client subscribes (at random) to:	100 objects
Update rate on each object subscribed to:	1 update/sec
Message Content for each update:	a) 5 fields each containing "AAA" Actual message size: 50 bytes b) 10 fields each containing "AAA" Actual message size: 80 bytes c) 20 fields each containing "AAA" Actual message size: 110 bytes
Bytes/second delivered to each client:	a) 5,000 bytes/sec b) 8,000 bytes/sec c) 11,000 bytes/sec
Liberator burst-max Setting:	0.5 sec

Number of fields in a message

This test shows the effect on Liberator performance of increasing the sizes of the fields in the update messages. In each test run the message size was the same, but the number of fields differed.



Number of fields in a message

The test results show that Liberator performs slightly better when handling update messages consisting of a small number of large fields rather than messages containing a larger number of smaller fields. However, the actual message size has a greater impact on performance, as shown in the test [Overall size of a message](#)³⁹.

Test details

Liberator version:	4.4
Number of DataSources (= number of DataSource threads in Liberator):	3
Number of Liberator session threads configured:	3
Number of Clients:	From 100 to over 6,000 (depending on message field size)
Number of DataSource objects available:	20,000 objects
Each client subscribes (at random) to:	100 objects
Update rate on each object subscribed to:	1 update/sec
Message Content for each update:	a) 5 large fields Actual message size 110 bytes b) 20 small fields Actual message size 110 bytes
Bytes/second delivered to each client:	a) & b) 11,000 bytes/sec
Liberator burst-max Setting:	0.5 sec

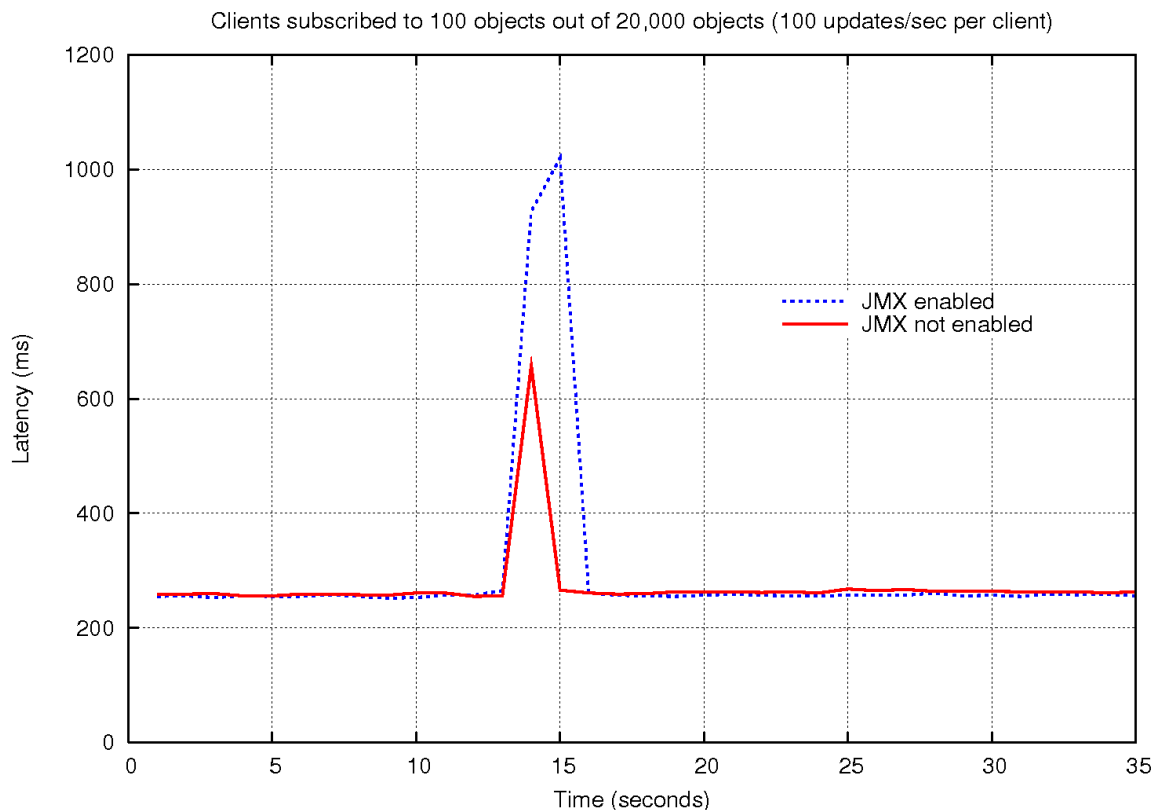
6.5 Effect of failover on performance

This section shows how a failover scenario could impact the performance of a system.

Latency during failover scenario

This test simulates the impact on performance of Liberator failover. The Liberator was initially loaded with 1,000 clients and after about 12 seconds another 1,000 clients were logged in at once. This simulates the situation where the Liberator would be required to take over the client logins and their subscriptions from another Liberator that has suddenly failed. Two scenarios were tested, one where JMX™ monitoring was enabled, and one where it was not.

Note: This test illustrates an extreme case involving Liberator failover. In normal circumstances login processing has almost no impact on Liberator performance.



Latency during failover scenario

In both scenarios, at the point of the failover the update message latency rapidly increases as the Liberator handles the sudden rush of client login requests and sets up the clients' subscriptions. When JMX is not enabled, after about two seconds the Liberator completes the login processing and message latency drops back to the previous steady state level. When JMX is enabled, the failover imposes a greater processing load, so the latency increase lasts a second longer and the peak latency increase is rather higher.

Test details

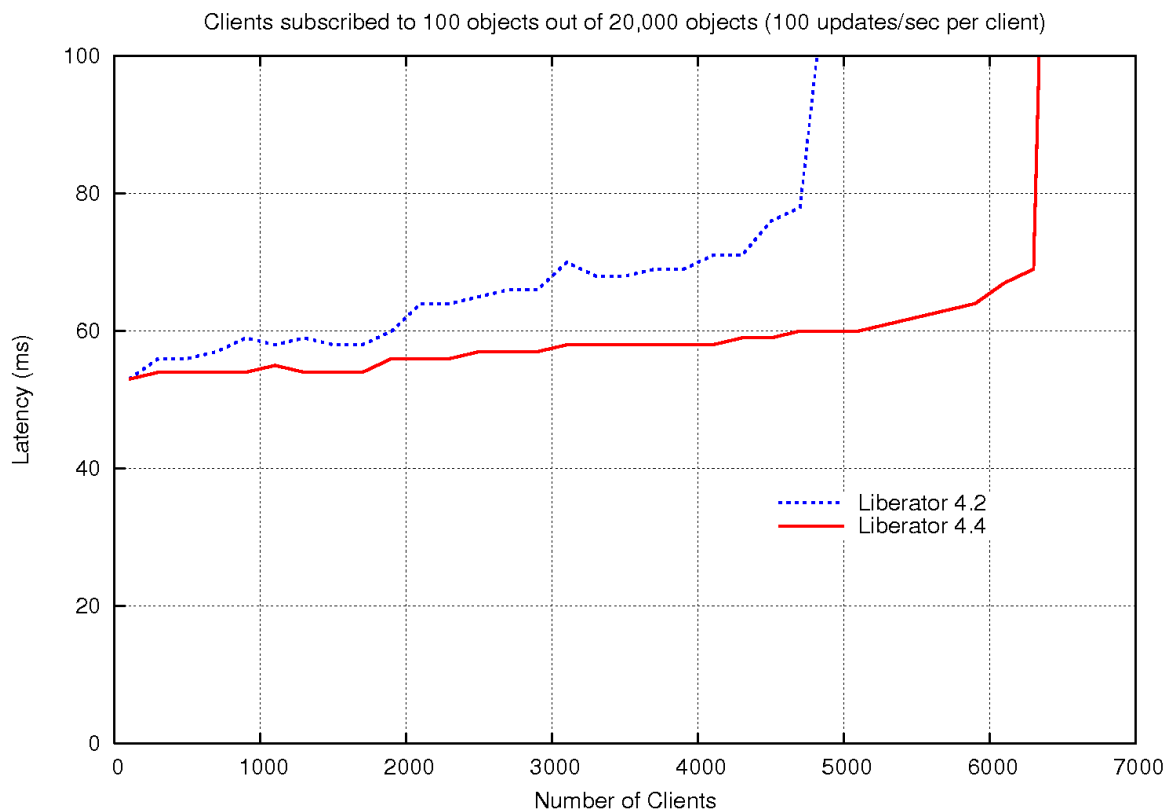
Liberator version:	4.4
Number of DataSources (= number of DataSource threads in Liberator):	3
Number of Liberator session threads configured:	3
Number of Clients:	1,000 then 2,000
Number of DataSource objects available:	20,000 objects
Each client subscribes (at random) to:	100 objects
Update rate on each object subscribed to:	1 update/sec
Message Content for each update:	5 fields: One field of 13 characters (timestamp) Four fields of 3 characters each Total 25 bytes of update data
Bytes/second delivered to each client:	2,500 bytes/sec
Liberator burst-max Setting:	0.5 sec

6.6 Liberator 4.4 versus 4.2

This section shows the results of tests that illustrate the performance improvement of Liberator 4.4 relative to Liberator 4.2 .

Low backend data set

This test compares an optimally configured version 4.4 Liberator with an optimally configured version 4.2 Liberator. It demonstrates how message latency changes as the number of subscribing clients is increased.



Liberator 4.4 versus 4.2 Low backend data set

Liberator 4.4 can make better use of session threads, due to improvements that have been made in its internal messaging compared to release 4.2. The graph shows that Liberator 4.4. has superior performance. Its message latency is lower than that of Liberator 4.2 and does not rise as rapidly as the number of subscribing clients increases. It can also support more clients (about 35% more in this case).

Test details

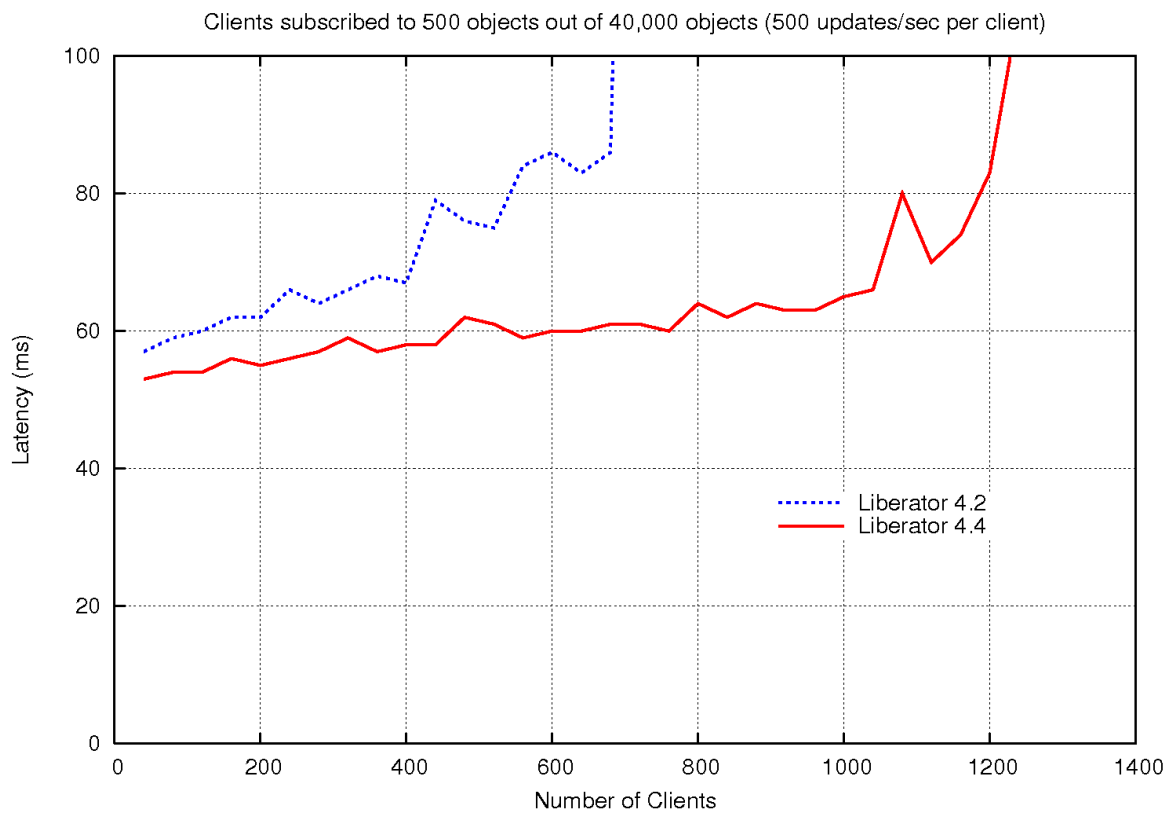
The DataSources only supplied updates at a relatively low rate (20,000 updates per second – 20,000 objects subscribed to, each object updated once per second). This meant that the Liberator 4.4 test only needed two instances of Benchsrc to drive it and the Liberator was configured accordingly with two DataSource threads.

Liberator 4.2 only has one DataSource connection per thread, regardless of the number of DataSources connected to it.

Liberator version:	4.4 & 4.2
Number of DataSources:	2
Number of DataSource threads in Liberator 4.4:	2
Number of DataSource threads in Liberator 4.2:	1
Number of Liberator 4.4 session threads configured:	4
Number of Liberator 4.2 session threads configured:	1
Number of Clients:	From 100 to over 6,300
Number of DataSource objects available:	20,000 objects (across minimum 200 clients)
Each client subscribes (at random) to:	100 objects
Update rate on each object subscribed to:	1 update/sec
Message Content for each update:	5 fields: One field of 13 characters (timestamp) Four fields of 3 characters each Total 25 bytes of update data
Bytes/second delivered to each client:	2,500 bytes/sec
Liberator Burst Max Setting:	0.1

Medium backend data set

This test compares an optimally configured version 4.4 Liberator with an optimally configured version 4.2 Liberator. Compared with the previous test the number of subscribed DataSource objects was doubled (to 40,000) and each client subscribed to five times more objects (500).



Liberator 4.4 versus 4.2 Medium backend data set

In this test Liberator 4.4 shows an even more marked performance improvement compared to Liberator 4.2. It can support double the number of clients before message latency rises to an unacceptable level. Liberator 4.2 can only use one DataSource thread, which limits how much DataSource input it can handle.

Test details

Liberator version:	4.4 & 4.2
Number of DataSources:	3
Number of DataSource threads in Liberator 4.4:	3
Number of DataSource threads in Liberator 4.2:	1*
Number of Liberator 4.4 session threads configured:	3
Number of Liberator 4.2 session threads configured:	1
Number of Clients:	From 40 to over 1,480
Number of DataSource objects available:	40,000 objects (across minimum 200 clients)
Each client subscribes (at random) to:	500 objects
Update rate on each object subscribed to:	1 update/sec
Message Content for each update:	5 fields: One field of 13 characters (timestamp) Four fields of 3 characters each Total 25 bytes of update data
Bytes/second delivered to each client:	12,500 bytes/sec
Liberator Burst Max Setting:	0.1

7 Frequently asked questions

The following sections discuss various issues concerning how to configure and tune Liberator and its environment to achieve the required performance.

7.1 What burst configuration should we use?

Liberator can be configured to batch messages together to improve overall performance. Liberator's default configuration uses a 0.5 second batch time. The configuration option for this is called **burst-max**. With this configuration Liberator may batch together outgoing messages to a client, delaying them by up to 0.5 seconds. If the message rate is slower than the burst setting the messages will be sent immediately.

Batching messages in this way means that with a fairly constant message rate an average latency of half the burst-max setting will be introduced. However, the benefit of this is that the Liberator and network can cope with higher message rates. There is clearly a trade off between latency and message rate, which is why the tests carried out in these Benchmarks show results for different settings of burst-max (see [Effect of burst configuration \(batching\) on performance](#)^[37]).

It is clear that a low **burst-max** setting (for example 0.1 sec) can improve overall average latency and achieve much higher messages rates than with no batching. In some cases, increasing burst-max to the default of 0.5 sec will allow Liberator to achieve even higher message rates, but this is at the expense of increased latency.

7.2 How many threads should we configure?

There isn't a simple answer to this question.

Liberator's configuration option **threads-num** sets the number of session threads used to service client connections. Liberator 4.4 also implements multiple DataSource threads – there is one thread per DataSource connection.

How many threads you require depends on the maximum anticipated update rate from Liberator's DataSource peers, the maximum expected number of subscribing clients, the subscription profile of the clients (see "DataSource threads" below), and the hardware characteristics of the machine running the Liberator (CPU speed and number of CPUs).

DataSource threads

DataSource threads enable Liberator to better handle high update rates from its DataSources. Even if there is only one DataSource instance feeding Liberator, you can still configure more than one connection to the DataSource so that the Liberator's performance can benefit from using multiple DataSource threads.

There is also a relationship between the number of *clients* using the Liberator and the number of DataSource threads required, but this depends on the subscription profile of the clients, as follows.

At one extreme, if each client subscribes to a different set of objects, then the update demand on the DataSources will increase roughly in proportion to the number of subscribing clients. So in this case, as the maximum anticipated number of clients increases, more DataSource threads will be required in order to achieve the same message latency.

At the other extreme, if each client subscribes to the same set of objects, then the update demand on the DataSources will remain constant as the number of subscribing clients increases. So in this case the number of DataSource threads required will *not* depend on the number of subscribing clients.

You should configure enough DataSource connections (and hence DataSource threads) to allow Liberator to handle the maximum anticipated update rate from the DataSource(s) with acceptable message latency for the maximum number of clients expected to use the Liberator.

Don't use more DataSource threads than you need, because the additional threads will not produce a significant additional improvement in performance. There will be a limit above which adding more DataSource connections has little extra benefit, because the limiting factor becomes the number of session threads (see test results for [Effect of DataSource threads on performance](#)^[31]).

Session threads

Provided there are enough DataSource threads to handle updates from the DataSources, increasing the number of session threads will allow more clients to subscribe with no unacceptable increase in message latency (see test results for [Effect of session threads on performance](#)^[26]). However there will be an upper limit on the number of session threads, beyond which performance will decrease – see "CPU resource considerations" below.

CPU resource considerations

Once CPU resource limits have been reached on the machine running the Liberator, adding more threads will not necessarily improve performance. For example, in the test results [Effect of session threads on performance](#)^[26] the graphs shows that, increasing the number of session threads from three to four actually caused a decrease in performance, because the CPU limits of the machine running the Liberator had been reached. In these particular tests the optimum thread configuration for the machine was two or three DataSource threads and three session threads, that is, five or six threads, plus the main Liberator thread, running in four CPUs.

7.3 How do message sizes affect performance?

When high numbers of clients and messages are used, the size of the message plays a significant part in the overall performance. Larger update messages decrease Liberator's maximum effective message rate. Liberator also performs slightly better when handling update messages consisting of a small number of large fields rather than messages containing a larger number of smaller fields.

See the test results in [Overall size of a message](#)^[39] and [Number of fields in a message](#)^[41].

7.4 How much bandwidth will our Liberator use?

In the context of Liberator performance, bandwidth is the update rate delivered to all subscribed clients in bytes/sec.

Each test gives the details of the bandwidth used per client (look in the Test Details at the entry labeled "Bytes/second delivered to each client:"). For each test you can use this figure, together with the maximum number of clients that the Liberator was able to handle in the test (shown on the X-axis of the graph where the "elbow" occurs in the plot), to calculate the maximum bandwidth used by the Liberator for that test: $\text{bandwidth} = \text{update rate/client} \times \text{maximum number of clients}$.

7.5 How many subscriptions can Liberator handle?

The number of subscriptions a client has does not significantly affect performance directly, rather the number of messages is far more significant. This can be controlled using throttling.

7.6 How much disk space will our Liberator need?

Liberator uses approximately 30 megabytes of disk space when it is installed. Running Liberator requires extra disk space for log files.

The amount of disk space needed for the log files depends entirely on messages rates and client activity. Liberator supports configuration options to control the cycling of log files, so it is possible to limit how much disk space is used and how much information is saved in log files. Log files can grow to high single-digit gigabytes per day in some setups.

7.7 Why are there no CPU usage measurements for some tests?

The results for low, moderate and high update rates in [Test results for headline figures](#)⁵ show server CPU usage, whereas the rest of the results in this report do not.

The reason for this lies in the rate at which updates are fed to the Liberator from the DataSource. In the first set of tests the update rate from the DataSource is fairly low. The single CPU core running the single Liberator thread that manages the DataSource input is easily able to cope, and the overall processing load is spread evenly across the CPU cores. This means that a plot of average CPU usage is meaningful.

In the rest of the tests the DataSource update is higher and more DataSource connection threads are configured in the Liberator to handle this load. As a result, CPU usage is no longer evenly distributed across the cores, so a plot of average CPU usage would be misleading. Only separate plots of the individual CPUs would be meaningful, but this would complicate the results graphs.

In practice end-to-end message latency is more significant as a measure of Liberator performance than CPU usage, because message latency has a direct impact on users and may increase long before CPU usage reaches its maximum.

Note that in the results graphs the point where the graph rises very steeply is where at least one of the processors on the machine hosting the Liberator has reached 100% CPU usage.

8 Glossary of terms and acronyms

This section contains a glossary of terms and acronyms relating to the Liberator benchmark.

Term	Definition
Benchsrc	<p>A Caplin benchmark test tool that provides DataSource messages as input to a Liberator server. It is primarily intended to be used in conjunction with Benchrttp and the control scripts from the Caplin Benchmarking kit.</p> <p>See Test DataSource application (Benchsrc)¹².</p>
Benchrttp	<p>A Caplin benchmark test tool that connects to a Liberator server and simulates a configurable number of clients and contributors of streamed RTTP data. It is primarily intended to be used in conjunction with Benchsrc and the control scripts from the Caplin Benchmarking kit.</p> <p>See Test RTTP client application (Benchrttp)¹³.</p>
burst-max	<p>The efficiency of the Liberator can be increased by writing user output in defined "bursts", particularly in a system with a large number of clients where bursting batches together small messages before outputting them to a client.</p> <p>burst-max is a Liberator configuration parameter that controls bursting. It is the maximum time in seconds of client update buffering before Liberator will send updates to a client.</p> <p>For more information see the Caplin Liberator 4.4 Administration Guide</p>
DataSource peer	An application that can communicate with another application using the Caplin DataSource protocol.
JMX	<p><u>J</u>ava <u>M</u>anagement <u>E</u>xtensions</p> <p>A Java™ technology for application and network management.</p>
RTTP	<p><u>R</u>eal <u>T</u>ime <u>T</u>ext <u>P</u>rotocol</p> <p>Caplin's object-oriented, real-time, protocol for the distribution of financial data and trade messages over internet-protocol networks between client applications and Caplin Liberator.</p>
SL4B	<p>StreamLink for Browsers</p> <p>Caplin StreamLink is a family of SDKs that allows developers to add RTTP streaming capability to client applications. StreamLink for Browsers is the StreamLink SDK for use in Ajax/JavaScript™/HTML environments.</p>
SL4J	<p><u>S</u>tr<u>e</u>am<u>L</u>ink for <u>J</u>ava</p> <p>Caplin StreamLink is a family of SDKs that allows developers to add RTTP streaming capability to client applications. StreamLink for Java is the StreamLink SDK for use in Java environments</p>
Throttling	A technique used by Caplin Liberator to improve performance by restricting the rate at which object updates are sent to a client.



The information contained in this publication is subject to UK, US and international copyright laws and treaties and all rights are reserved. No part of this publication may be reproduced or transmitted in any form or by any means without the written authorization of an Officer of Caplin Systems Limited.

Various Caplin technologies described in this document are the subject of patent applications. All trademarks, company names, logos and service marks/names ("Marks") displayed in this publication are the property of Caplin or other third parties and may be registered trademarks. You are not permitted to use any Mark without the prior written consent of Caplin or the owner of that Mark.

This publication is provided "as is" without warranty of any kind, either express or implied, including, but not limited to, warranties of merchantability, fitness for a particular purpose, or non-infringement.

This publication could include technical inaccuracies or typographical errors and is subject to change without notice. Changes are periodically added to the information herein; these changes will be incorporated in new editions of this publication. Caplin Systems Limited may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

Contact Us

Caplin Systems Ltd.
Triton Court
14 Finsbury Square
London EC2A 1BR
UK

Telephone: +44 20 7826 9600
Fax: +44 20 7826 9610

www.caplin.com
info@caplin.com

Document version 2