CAPLIN

Caplin Refiner 5.0.0

Benchmarks

November 2011

CONFIDENTIAL

Contents

1 Preface

1.1 What this document contains

This document details the results of a set of performance benchmark tests carried out on Caplin Refiner 5.0. It is hoped that the information provided in this report will assist customers in production capacity planning when deploying Refiner.

About Caplin document formats

This document is supplied in Portable document format (.PDF file), which you can read on-line using a suitable PDF reader such as Adobe Reader®. The document is formatted as a printable manual; you can print it from the PDF reader.

1.2 Who should read this document

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

- Technical Managers
- System Architects
- System Administrators

1.3 Related documents

- **Caplin Xaqua: Overview**

  A business and technical overview of Caplin Xaqua.

- **Caplin Liberator 5.1 Administration Guide**

  Describes the Caplin Liberator server and its place within Caplin Xaqua.
  Explains how to install, configure, and manage the Liberator.
  Includes configuration reference information, and a list of Liberator's log and debug messages.

- **Caplin Xaqua: How To Use Containers**

  Describes what containers are, and how Caplin Xaqua applications can use them to group and manage objects such as financial instruments for manipulation in a user interface.

- **Caplin DataSource Overview**

  A technical overview of Caplin DataSource, its place within Caplin Xaqua, and how it can integrate with you own software and network infrastructure.

- **Caplin Xaqua: Permissioning Overview And Concepts**

  A technical overview of permissioning concepts and terms that describes subject mappings, and how they can be applied to provide price tiering to users of a Caplin Xaqua client application.

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.

Visit our feedback web page at https://support.caplin.com/documentfeedback/.

1.5 Acknowledgments

2 Overview

2.1 About Caplin Refiner

Caplin Refiner dynamically filters and sorts containers on the server side of Caplin Xaqua based on filter requests from client applications. This work is done on the server side to reduce the amount of data that has to be sent to the client, and also to reduce the load on client applications.

2.2 About the benchmark tests

The benchmark tests detailed in this document are designed to show how Caplin Refiner performance will change as certain operating parameters are varied. Each test shows the results of varying a single parameter. As each Caplin Xaqua system will have different sorting and filtering profiles, the benchmarks will help you to identify the parameters that are relevant in a particular situation, and therefore what kind of performance can be expected in that situation.

The tests cover:

- A product finder scenario, measuring the response time of multiple unique requests to filter a container.
- Another product finder scenario, measuring the latency of changes to the constituent objects in a container.
- A trade blotter scenario, measuring the latency of changes to the content of a trade blotter.

The tests were conducted on servers representing typical commercially available machines that can be used to host Web servers and server applications. A single Caplin Refiner instance was run on one machine, while test harnesses were run on other machines to provide data and client processes.

For detailed information on the test set up used at Caplin Systems for these benchmarks, see Section 7 on page 25.

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.

2.3 Headline results

Details of the test scenarios, environments, full results and graphs can be seen in later sections, but here are some headline results for these tests:

- Filtering and sorting on rapidly updating field values is not recommended with large underlying containers.
- When filtering rapidly updating fields, performance deteriorates rapidly when Caplin Refiner has to process more than five million updates per second. The update rate is given by multiplying the number of unique filter requests by the number of underlying container objects that update per second.
- When Caplin Refiner provides historic trade blotter data, two thousand users with one thousand blotter entries each produces two million unique objects, and requires about 16 GB of random access memory. Even although update latency remains low at these levels, available memory is the limiting factor.
- For unique filter request rates of 8.3 per second, average response times on containers approaching the upper limit of 65,555 constituent objects remained below 100 milliseconds.

3 Test scenarios

This section introduces the test scenarios used in these benchmarks. These scenarios are designed to simulate different types of activity typically seen in real time financial applications, and demonstrate the kind of performance that Caplin Refiner can achieve.

Parameter values can vary significantly between different business scenarios, *and therefore Caplin always advises customers to run benchmarks that reflect their actual requirements for client subscription and data update profiles*. Caplin's benchmark tools make this easier to do; once the test environment is set up using these tools, it is easy to configure and test different scenarios. Benchmarks can either be run against a test back end or against real data supplied by the customer.

Product Finder Scenario - Response Time

This scenario measures the response time of requests to filter the contents of a single container. Each filter is a unique request to filter the container by static field values of constituent objects (values that never update, such as the field containing an instrument description).

Parameters that vary in this scenario are the filter request rate, the underlying container size, the filtered container size, and the number of configured subject mappings.

Product Finder Scenario - Latency

This scenario measures the latency of updates to container objects that have already been filtered and returned to the client. The filter request in this scenario filters the constituent objects of a single container by dynamic field values (values that update, such as the field containing the price of an instrument).

Because the field being filtered is dynamic, the filter can add or remove objects from the filtered container as field values change. The latency measured is the difference in time between the DataSource updating an object and the container update being received at the client.

Parameters that vary in this scenario are the number of unique filter requests and the type of filter request (filter only or filter and sort).

Trade Blotter Scenario - Latency

This scenario measures the latency of updates to a container that represents the content of a user's trade blotter. Because each user has their own blotter showing the state of trades they submitted, Caplin Refiner must filter multiple underlying containers as trades change state and as users submit new trades.

The latency measured is the difference in time between the DataSource updating the state of a trade object and the update being received at the client.

The parameter that varies in this scenario is the number of unique filter requests, which corresponds to the number of logged in users, trade blotters, and therefore the number of underlying containers that Caplin Refiner must filter.

4 Product Finder Scenario – Response Time

This scenario simulates an instrument search using a Product Finder. A Product Finder might consist of a grid with a large number of items in it (possibly tens of thousands) that a user wants to filter down to a smaller subset, enabling them to more easily find the instruments they are looking for.

The filtered responses for all clients are derived by Caplin Refiner from a single underlying container that holds the entire product set.

In this benchmark, the time taken to respond to filter requests is the response time minus the request time.

4.1 Test parameters

The scenario consists of four tests that measure the response times to a number of unique filter requests. In each test, three of four parameters are held constant and one parameter is varied. The parameters are:

Requests per second: The number of unique filter requests per second.

The value of this parameter is determined by the number of users that log in to Liberator over a one minute period, where each user requests a filtered container as soon as they log in. Caplin's Benchrttp client (see Section 7 on page 25) was used to simulate user logins and filter requests.

Underlying container size: The size of the underlying container (the number of constituent objects in the container).

Filtered container size: The size of the filtered container that Caplin Refiner returns to Liberator.

The filter applied to the underlying container determines the filtered container size.

Number of tiers: The number of configured subject mappings.

Subject mappings can be used to apply different pricing tiers for different users. This means that different users could get different prices for the same instrument. When a subject mapping applies to a user, the subject requested from the DataSource is not the same as the subject requested by the client. In each test, the configured subject mappings are spread across all users, such that each user has one subject mapping and some users have the same subject mapping. Caplin Refiner must apply the mappings to the subjects of the constituent container objects before it requests these objects from the providing DataSource.

4.2 Test conditions

The following conditions apply to each of the four tests:

- The underlying container is cached by Caplin Refiner before each test is started, which removes the caching time from the test results. Caplin Refiner normally requests the underlying container from the providing DataSource when it receives the first filter request for that container, but not when it receives subsequent requests to filter the same container.

- The window size of each filtered container is 50. This means that Liberator will never return a container with more than 50 constituent objects to the client no matter how many objects are in the filtered container that Caplin Refiner returns to Liberator. In this way the client is not sent a container with objects that it cannot display (such as a grid that can only display a maximum of 50 instruments).
- Each unique filter is on a static number field and a static text field (the DataSource never updates static field values). Other tests (not shown in this document) demonstrate that the kind and complexity of the applied filter does not have a significant effect on response times.
- Caplin Refiner is requested to sort each filtered container by the natural order of the underlying container, which in this scenario sorts on a text field. Other tests (not shown in this document) demonstrate that the field used to sort the container does not have a significant effect on the sort time.
- The constituent objects of the container are updated at the rate of one every second. Filtered fields are never updated, but Caplin Refiner must process all updates to determine this.
- The results of each test are plotted on two graphs. The first graph plots the average response time for all requests together with the 95 percentile for these requests (the value that 95 percent of requests fall within). The second graph plots the spread of response times for the other 5 percent, showing the peak response times and the 99 percentile values.

4.3 Requests per second

Response times were measured for each of the following rates of unique filter requests per second.

Requests per second: 0.2, 1.7, 8.3, 16.7, 33.3, 66.7. 83.3, 133.3, and 166.7

The request rate was increased by increasing the number of users that log in to Liberator over a one minute period, each requesting a unique filtered container as soon as they log in. Logins were equally spaced over this one minute period.

The following constant parameter values were used in each of the test runs.

Underlying container size	Filtered container size	Number of tiers
10,000	1,000 (1% of underlying container)	5

When interpreting the results of this test, the following should be taken into account. The request rate is a parameter that relates to the number of users requesting a unique filtter, but this is not a direct relationship. For example:

- If five users each request a unique filter on the same container, Caplin Refiner will have to process five unique filter requests.
- If five users each request the same filter on the same container, Caplin Refiner will have to process one unique filter request.
- If five users each make five unique filter requests on the same container, Caplin Refiner will have to process 25 unique filter requests.

Peak request rates are often caused by fail-over scenarios, such as when Liberator sends all filter requests it is serving to a secondary (backup) Caplin Refiner.

Test results (varying the request rate)

The following graph shows that the average response time increases linearly as the request rate increases from 0.2 to 166.7 unique reqests per second, but even at this higher request rate the response times are still in the region of 70 milliseconds.

The 95 percentile values closely follow the average response times, showing little spread until the request rate is 83.3 unique requests per second. At this point the spread increases, and the peak 95 percentile response time is 255 milliseconds at 133.3 unique requests per second. This spread is not considered to be significant and would not adversely affect user experience, as the user would probably not notice a delay of less than 200 milliseconds.

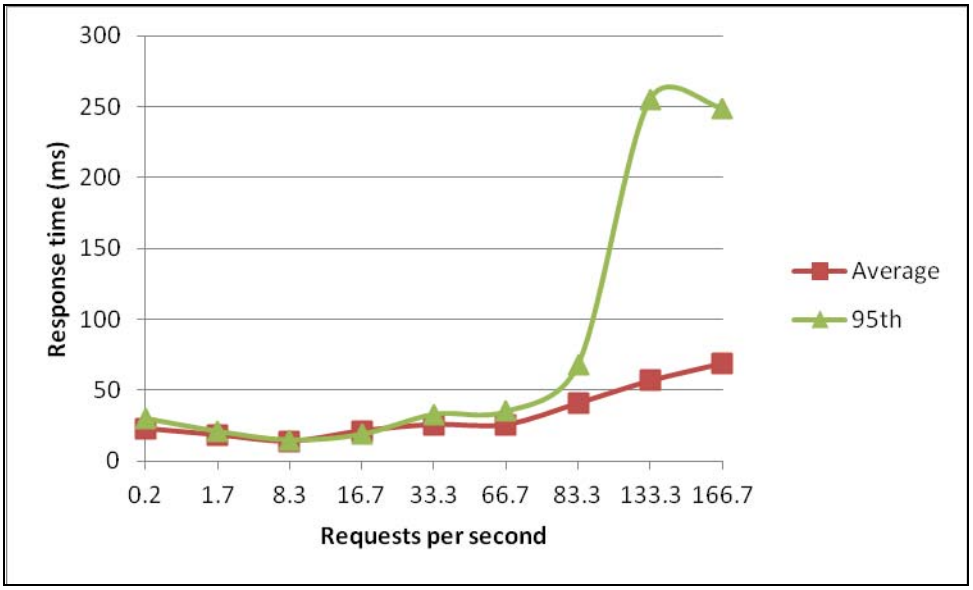


Figure 4.1 – Increasing the request rate

The next graph plots the worst case response times and 99 percentile values for the same test runs, and shows that only one percent of response times were greater than 500 milliseconds at requests rates above 66.7 requests per second. High response times like this can be caused by several factors, including Java garbage collection, and are not necessarily due to limitations in Caplin Refiner performance.

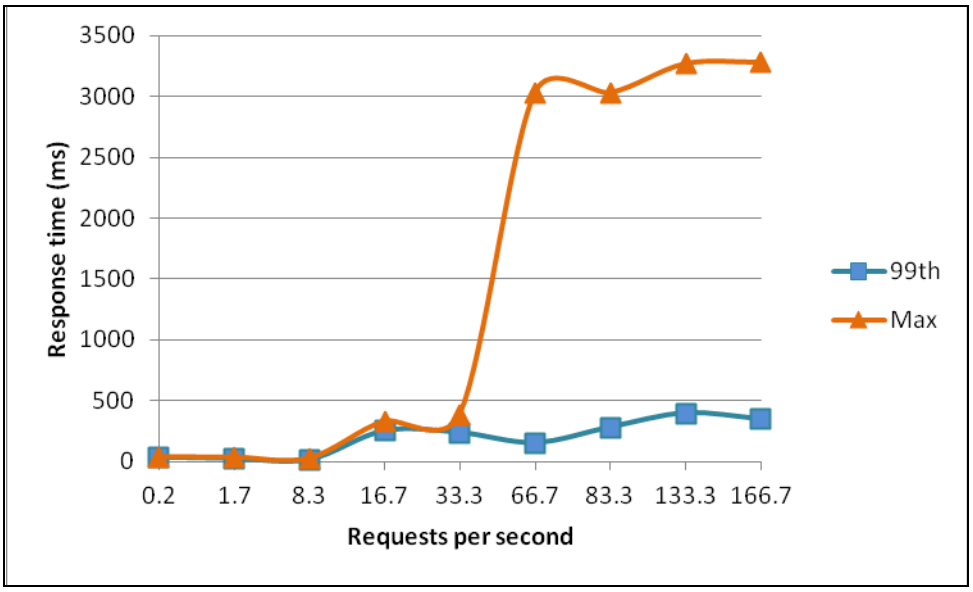


Figure 4.2 – Increasing the request rate (worst case)

4.4 Container size

Response times were measured for each of the following underlying container sizes.

Container size: 100, 1,000, 10,000, 30,000, 50,000, and 64,000

The following constant parameters values were used in each of the test runs.

Requests per second	Result size	Number of tiers
8.3	1% of underlying container	5

The request rate of 8.3 requests per second was set by logging in 458 users to Liberator at equal intervals over a one minute period, each requesting a unique filtered container as soon as they log in.

Test results (varying the underlying container size)

The following graph shows that the average response time increases linearly from 13 milliseconds for a container size of 100, up to 48 milliseconds for a container size of 50,000. At this point the average response time increases more rapidly, but is still in the region of 70 milliseconds with an underlying container size of 64,000.

The 95 percentile values closely follow the average response times, showing little spread with a container size of 64,000 objects. Note that a container cannot contain more than 65,555 constituent objects.

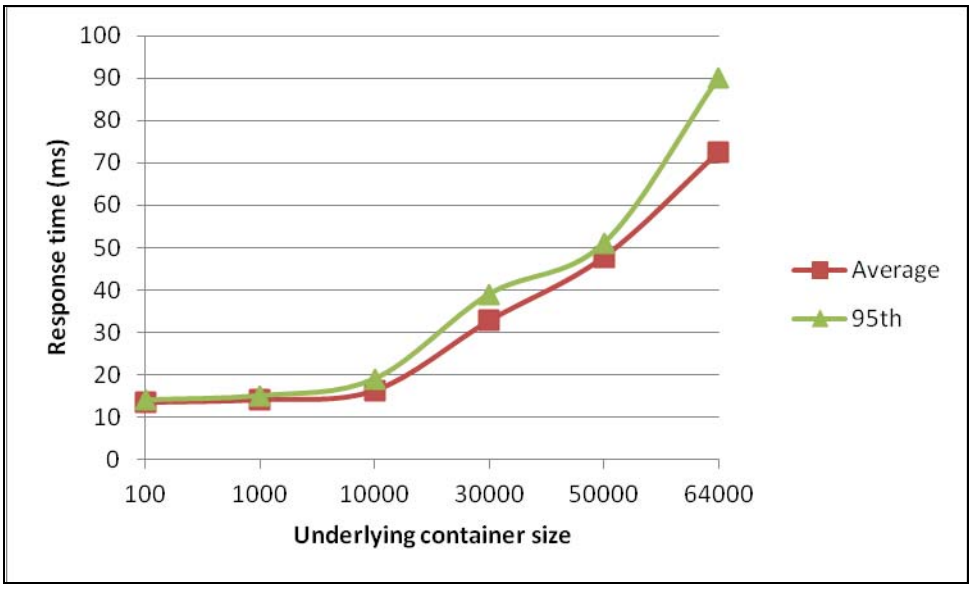


Figure 4.3 – Increasing the underlying container size

The next graph plots the worst case response times and 99 percentile values for the same test runs, and shows that only one percent of response times were above 500 milliseconds. High response times like this could be caused by several factors, including Java garbage collection, and are not necessarily due to limitations in Caplin Refiner performance.

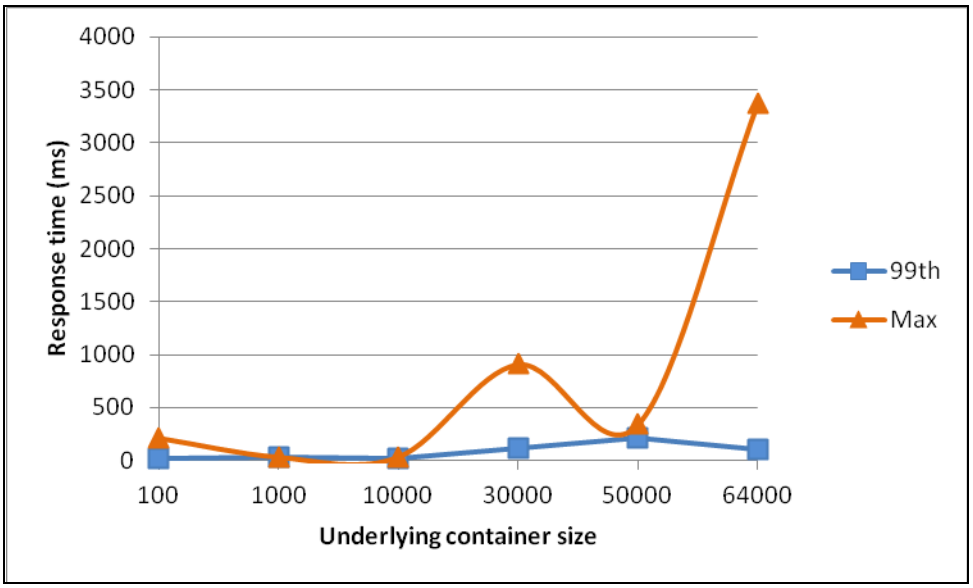


Figure 4.4 – Increasing the underlying container size (worst case)

4.5 Filtered container size

Response times were measured for each of the following filtered container sizes.

Filtered container size: 10, 100, 1,000, 5,000, 9,000 and 10,000

The filtered container size was varied by requesting different filters on the same underlying container.

The following constant parameters values were used in each of the test runs.

Underlying container size	Requests per second	Number of tiers
10,000	8.3	5

The request rate of 8.3 requests per second was set by logging in 458 users to Liberator at equal intervals over a one minute period, each requesting a unique filtered container as soon as they log in.

Test results (varying the filtered container size)

The following graph shows that the average response time increases linearly from 14 milliseconds for a filtered container size of 10, to 86 milliseconds for a filtered container size of 10,000.

The 95 percentile values closely follow the average response times, showing little spread with filtered container sizes of up to 10,000 objects.

Note that although the time it takes to sort a container generally grows exponentially with the number of items in the container, this had no noticable effect on the measured response times of the filtered containers used in this test.

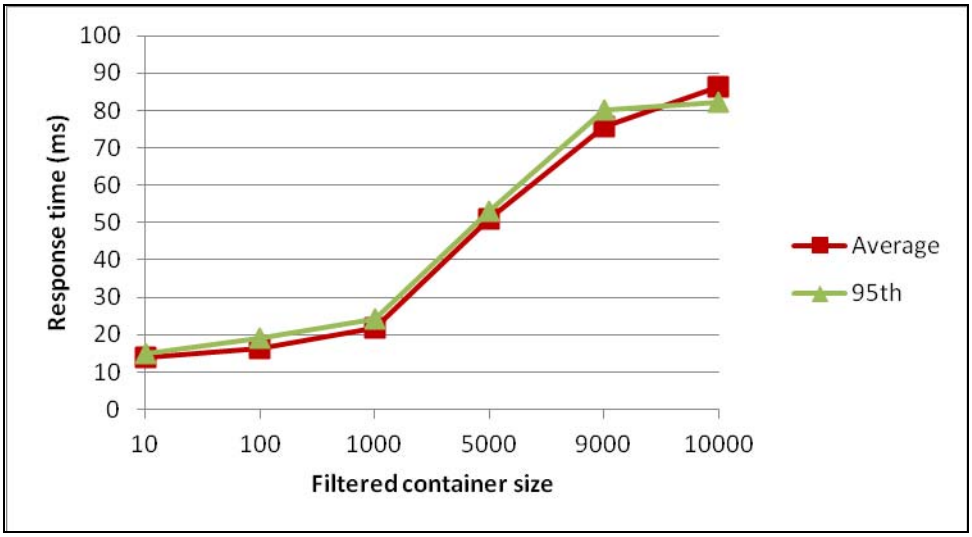


Figure 4.5 – Increasing the filtered container size

The next graph plots the worst case response rates and 99 percentile values for the same test runs, and shows that response times were always below 600 milliseconds.

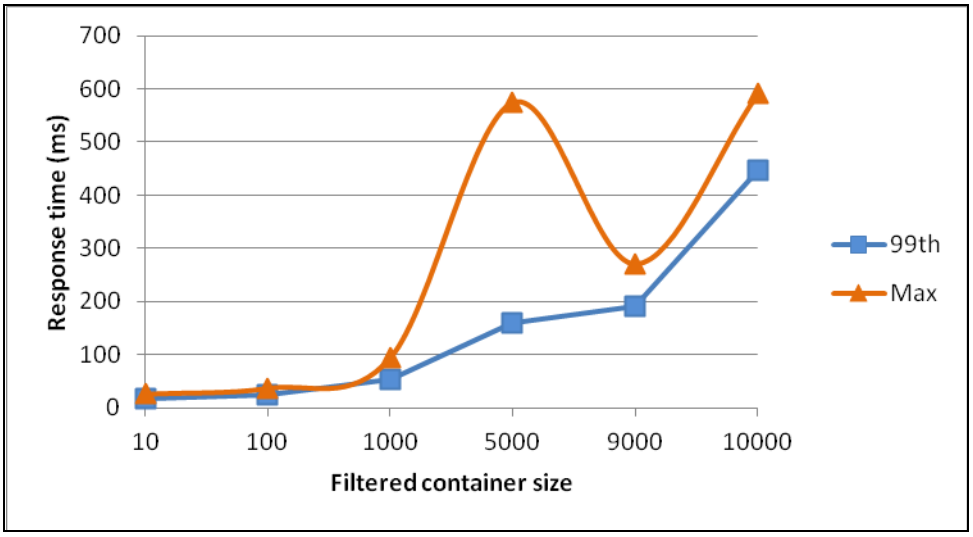


Figure 4.6 – Increasing the filtered container size (worst case)

Although not shown here, returning large filtered responses to client applications can impose a heavy load on Liberator.

4.6 Tier size

Response times were measured for each of the following tier sizes.

Tier size: 1, 5, and 10

The tier size was varied by applying different subject mappings to the logged in users.

The following constant parameters values were used in each of the test runs.

Underlying container size	Result size	Requests per second
10,000	1,000 (1% of underlying container)	8.3

The request rate of 8.3 requests per second was set by logging in 458 users to Liberator at equal intervals over a one minute period, each requesting a unique filtered container as soon as they log in.

Test results (varying the tier size)

The following graph shows that increasing the number of tiers had little effect on the average response times.

The 95 percentile values closely follow the average response times, showing little spread at each tier size.

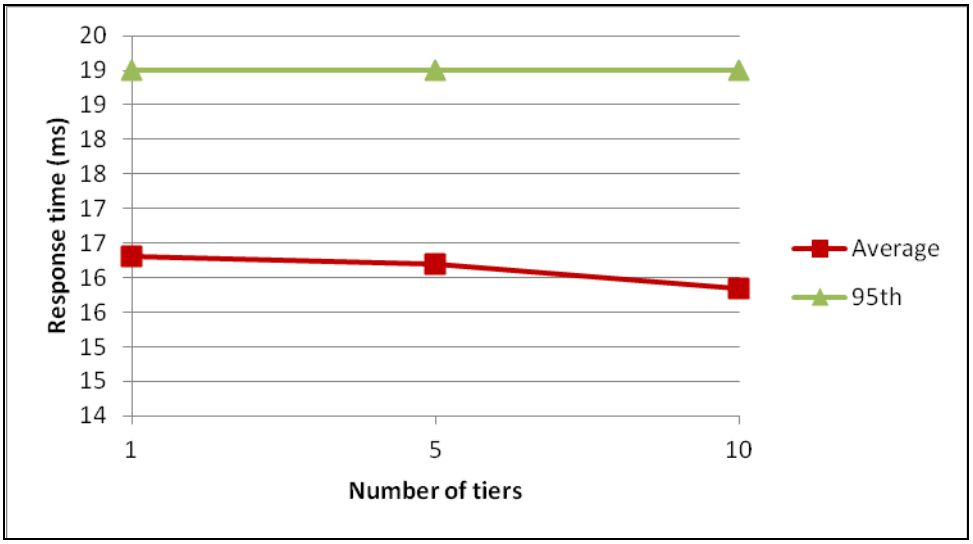


Figure 4.7 – Increasing the number of tiers

The next graph plots the worst case response rates and 99 percentile values for the same test runs, and shows that response times were always below 70 milliseconds.

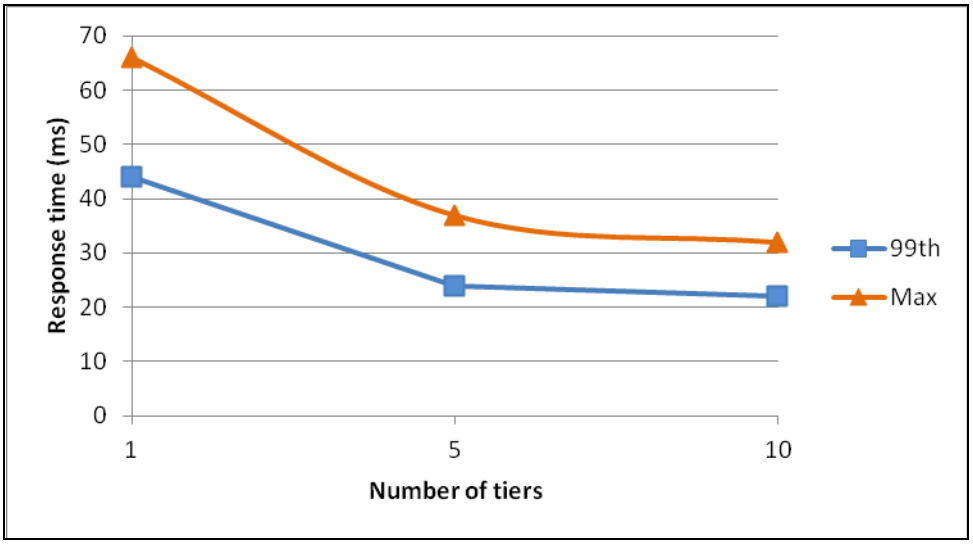


Figure 4.8 – Increasing the number of tiers (worst case)

5 Product Finder Scenario - Latency

This scenario simulates the behaviour of a Product Finder, where the user is able to filter a container on dynamically changing field values of the constituent container objects.

This benchmark measures the latency of updates from the time each update is sent from the providing DataSource, until the time the update is received at the client.

Benchmarks like this show the limits of live sorting and filtering on large containers of frequently updating fields. Caplin do not recommend using Caplin Refiner in such a scenario, as it is extremely processing intensive and could result in poor overall system performance. It also provides a poor user experience with grid rows jumping around quickly and randomly on the screen.

5.1 Test Parameters

The scenario consists of two tests that measure the latency of object updates, the first on a filtered container that is sorted by the natural order of the underlying container, and the second on a filtered container that is sorted on a dynamically changing field. The variable parameters in this test are:

Unique requests: The number of unique filter requests that Caplin Refiner has to process.

Request type: Whether each request is to filter the underlying container, or to filter and sort the underlying container.

In each test the request type is held constant as the number of unique filter requests is varied.

5.2 Test conditions

The following conditions apply to each of the two tests:

- The underlying container is cached by Caplin Refiner before each test is started, which removes the caching time from the test results. Caplin Refiner normally requests the underlying container from the providing DataSource when it receives the first filter request for that container, but not when it receives subsequent requests to filter the same container.
- The underlying container has 30,000 constituent objects.
- Filters are applied in such a way that each filtered container always has 300 constituent objects (one percent of the underlying container).
- The objects in the underlying container are updated every five seconds. Updates are applied to filtered fields in such a way that every five seconds, one record is added to, and one record removed from, each filtered container.
- A custom StreamLink for Java (SL4J) client measures the latency of updates to the filtered container it requests, while a varying number of Benchrttp clients load the system with similar filter requests.

- The window size of each Benchrttp client is 50. This means that Liberator will never return a container with more than 50 constituent objects to the client no matter how many objects are in the filtered container that Caplin Refiner returns to Liberator. In this way the client is not sent a container with objects that it cannot display (such as a grid that can only display a maximum of 50 instruments).
- The window size of the SL4J client is 300.
- The results of each test are plotted on two graphs. The first graph plots the average latency for all object updates, together with the 95 percentile for these updates (the value that 95 percent of results fall within). The second graph plots the spread of response times for the other 5 percent, showing the peak latency and 99 percentile values.
- The vertical scale of the graphs used to present the latency results are logarithmic and do not start at zero. This is to allow the wide range of latency results to be displayed on the graphs.

5.3 Filter only requests

Latency was measured over a ten minute period for each of the following unique filter requests.

Unique requests: 10, 50, 100, 300, 500, 700, 800, 850, 900, 950, 1,000

The number of unique requests was increased by increasing the number of users that log in to Liberator, each requesting a unique filtered container as soon as they log in. Because each filter is applied to a dynamic field, the number of filtered container updates that Caplin Refiner has to process is directly proportional to the number of unique filter requests.

Because a sort is not specified with these filter requests, Caplin Refiner sorts each filtered container by the natural order of the underlying container.

Test results (filter only requests)

The following graph shows an average latency of 48 milliseconds with 10 unique requests, increasing steadily to an average of about 506 milliseconds with 900 unique requests. At 950 unique requests the latency increases dramatically to 10 seconds, and then to 66 seconds at 1,000 unique requests.

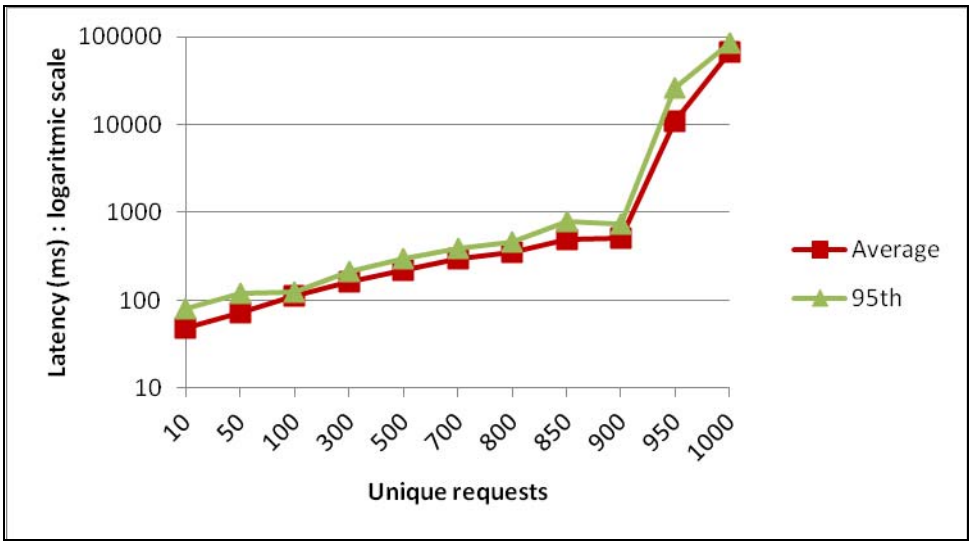


Figure 5.1 – Increasing the number of unique filter requests

This rapid increase in latency indicates the limit of filtering dynamically changing fields. In this test the underling container has an update rate of 6,000 objects per second (30,000 objects updating once every five seconds). With 900 unique requests, Caplin Refiner is processing 5.4 million updates per second (900 x 6,000). For this reason it is recommended that Caplin Refiner does not process more than 5 million updates per second, where the update rate is given by:

unique requests × number of underlying container objects that are updated per second

If a field is updating frequently but is not needed to filter or sort the container, the update should not be sent to Caplin Refiner. If you want to know how to do this, please contact Caplin support.

Although not shown here, the latency added by Caplin Refiner when handling a large amount of updates has little effect on the response time to new filter requests. This is due to the threading model used by Caplin Refiner.

The 95 percentile values closely follow the average latency times, showing little spread from the average.

The next graph plots the worst case latency and 99 percentile values for the same test runs, and shows that one percent of latency times were over 990 milliseconds at all request rates.

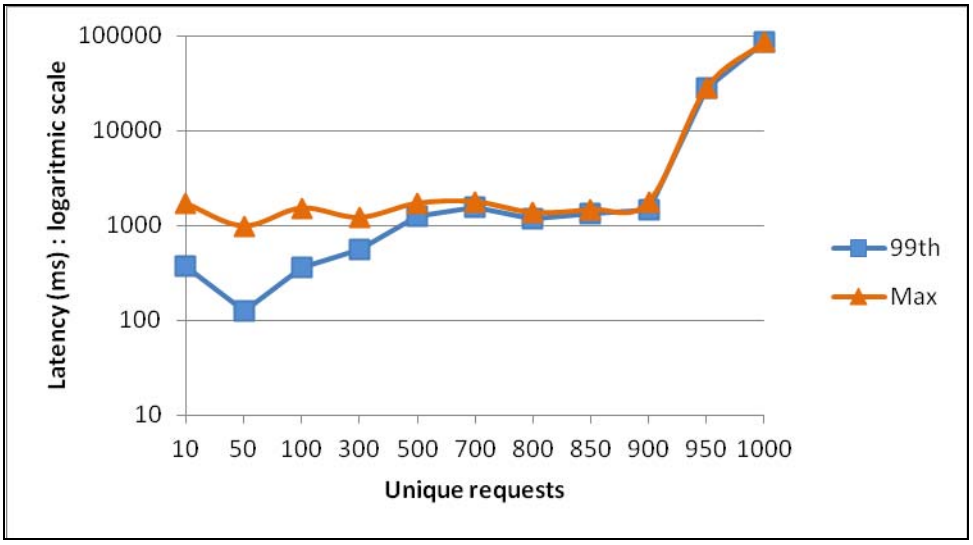


Figure 5.2 – Increasing the number of unique filter requests (worst case)

5.4 Filter and sort requests

Latency was measured over a ten minute period for each of the following unique filter and sort requests.

Unique requests: 10, 50, 100, 300, 500, 700, 800, 850, 900, 950, 1,000

The number of unique requests was increased by increasing the number of users that log in to Liberator, each requesting a unique filtered container as soon as they log in. Because each filter is applied to a dynamic field, the number of filtered container updates that Caplin Refiner has to process is directly proportional to the number of unique filter requests.

The requested sort field updates randomly once every five seconds, although it is unrealistic to sort on a field like this in a production environment.

All filter and sort requests are to filter on one dynamic field and to sort on another dynamic field.

Test results (filter and sort request)

The following graph shows an average latency of 95 milliseconds with 10 unique requests, increasing steadily to an average of about 1038 milliseconds with 700 unique requests. At 900 unique requests the latency increases dramatically to 50 seconds, and then to 102 seconds at 1,000 unique requests.

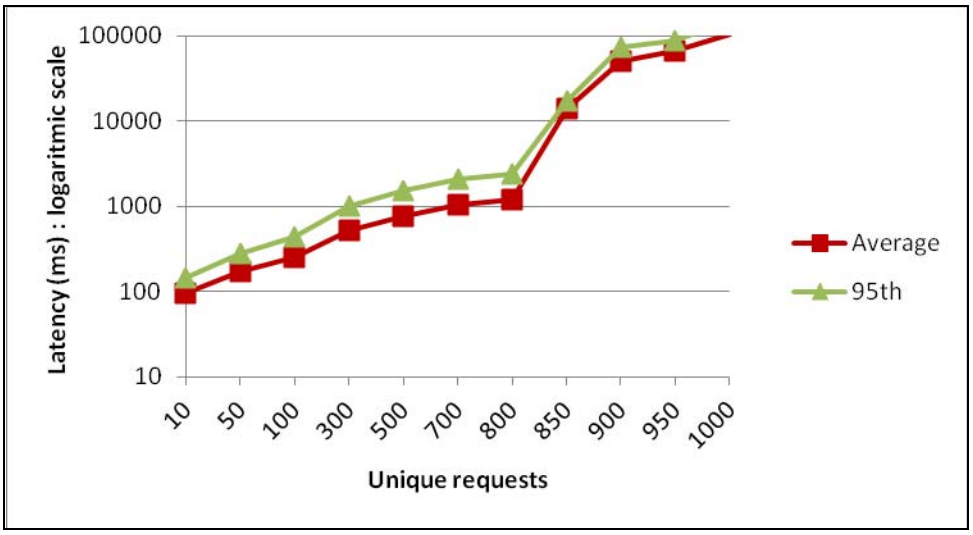


Figure 5.3 – Increasing the number of unique filter and sort requests

With 850 unique requests, every five seconds Caplin Refiner is sorting 30,000 randomly updating fields for 850 containers, each containing 300 objects (approximately 50,000 sorts per second). Randomly updating fields are the worst case scenario, and this test shows that sorting the filtered container more than doubles the latency (compared to filter only requests). Other tests (not shown in this document) indicate that sorting is much faster when updates to objects do not significantly change the order of the sorted container.

Sorting on dynamically changing fields is very CPU intensive and is affected by many factors, and Caplin recommends that you benchmark your own use cases if you want to use this kind of sorting. Factors that affect sorting times include the container size (proportional to *n* log *n,* where n is the number of items to sort), the update rate of the sort field, the number of unique filter and sort requests, and how stable the sort order is.

The next graph plots the worst case latency and 99 percentile values for the same test runs, and shows that two percent of latency times were over 560 milliseconds at all request rates, rising to a worst case of 156 seconds at 1,000 unique requests.

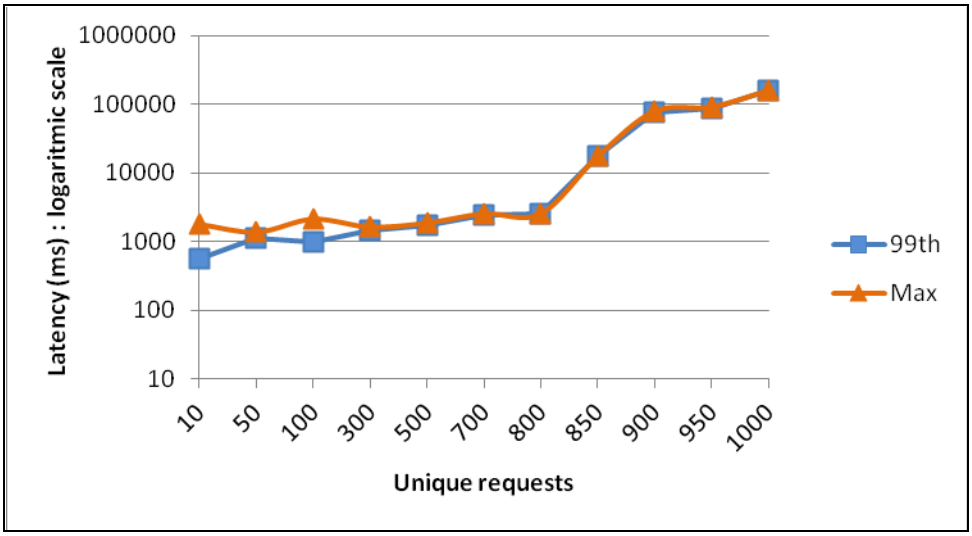


Figure 5.4 – Increasing the number of unique filter and sort requests (worst case)

6 Trade Blotter Scenario – Latency

This scenario is intended to reproduce the behaviour of a Trade Blotter consisting of a grid showing the details of submitted trades. New trades would be added to the top of this grid, and filtering could be requested on the fields of blotter entries to show or hide certain trades.

In this scenario, instead of all responses being unique filters on a single underlying container (as in the other benchmarks shown in this document), each user has their own underlying container holding only the trades they are permitted to see.

The benchmark measures the latency of updates from the time each update is sent by the providing DataSource, until the time it is received at the client.

6.1 Test Parameter

There is one variable parameter in this scenario:

Unique requests: The number of unique filter requests that Caplin Refiner has to process.

6.2 Test conditions

The following conditions apply to this test:

- Each underlying container is cached by Caplin Refiner before the test is started, which removes the caching time from the test results. Caplin Refiner normally requests the underlying container from the providing DataSource when it receives the first filter request for that container, but not when it receives subsequent requests to filter the same container.
- Each underlying container starts with 100 constituent objects that have a Status field set to 10.
- One object is added every 10 seconds (to simulate new trades), until each container has 1,000 constituent objects. The Status field of each of these objects is set to one.
- Every three seconds the DataSource updates the numeric Status field of all objects in the underlying containers. The Status field increments from one up to 10, simulating the trade moving through various trading states, with 10 meaning that the trade is complete. The filter removes all objects with a Status less than 10, simulating a filter on all trades currently in progress. Because only two or three trades are ever in progress, the filtered result size varies between two and three objects.
- A custom StreamLink for Java (SL4J) client measures the latency of updates to the filtered container it requests, while a varying number of Benchrttp clients load the system with similar filter requests.
- The window size of each client is 20. This means that Liberator will never return a container with more than 20 constituent objects to the client no matter how many objects are in the filtered container that Caplin Refiner returns to Liberator. In this way the client is not sent a container with objects that it cannot display (such as a grid that can only display a maximum of 20 instruments).

- The results of this test are plotted on two graphs. The first graph plots the average latency for all object updates, together with the 95 percentile for these updates (the value that 95 percent of results fall within). The second graph plots the spread of response times for the other 5 percent, showing the peak latency and 99 percentile values.

6.3 Blotter latency

Latency was measured as the content of the underlying container increased from 100 to 1,000 constituent objects (to simulate new trades being added to the blotter). The test was run for each of the following unique filter requests.

Unique requests: 100, 500, 1,000, and 2,000

The number of unique requests was increased by increasing the number of users that log in to Liberator, each requesting a unique filtered container as soon as they log in.

Test results (blotter latency)

The following graph shows an average latency of nine milliseconds with 100 unique requests, increasing steadily to an average of 44 milliseconds with 2,000 unique requests. The 95 percentile values have a maximum spread of about 80 milliseconds with 2,000 unique requests.

With 2,000 unique requests at the end of this test there will be two million records in the underlying containers. This requires 16 GB of memory and imposes a memory constraint on Caplin Refiner.

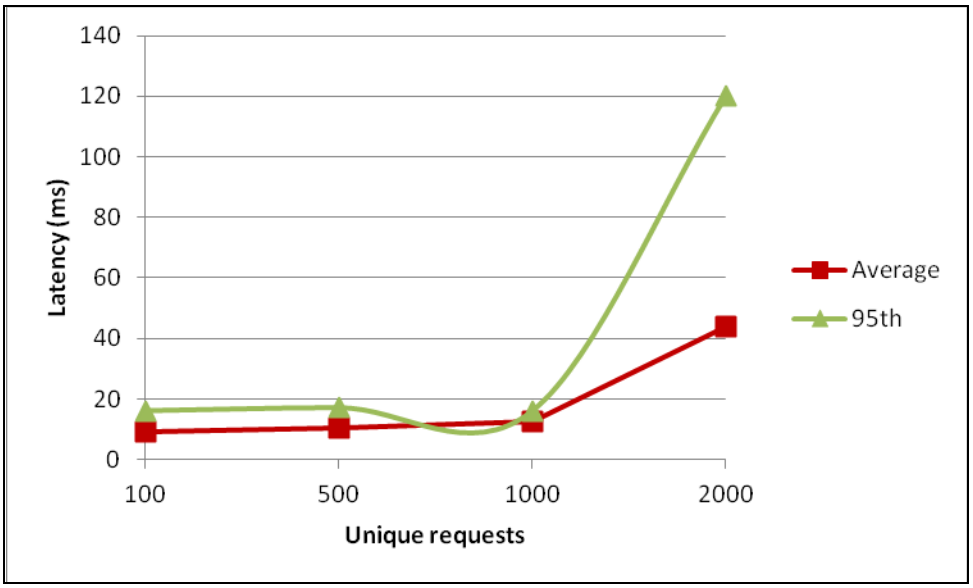


Figure 6.1 – Increasing the number of unique filter requests

The next graph plots the worst case latency and 99 percentile values for the same test runs, and shows that one percent of latency times were 400 milliseconds at 500 unique requests, rising to 1033 milliseconds at 2,000 unique requests.

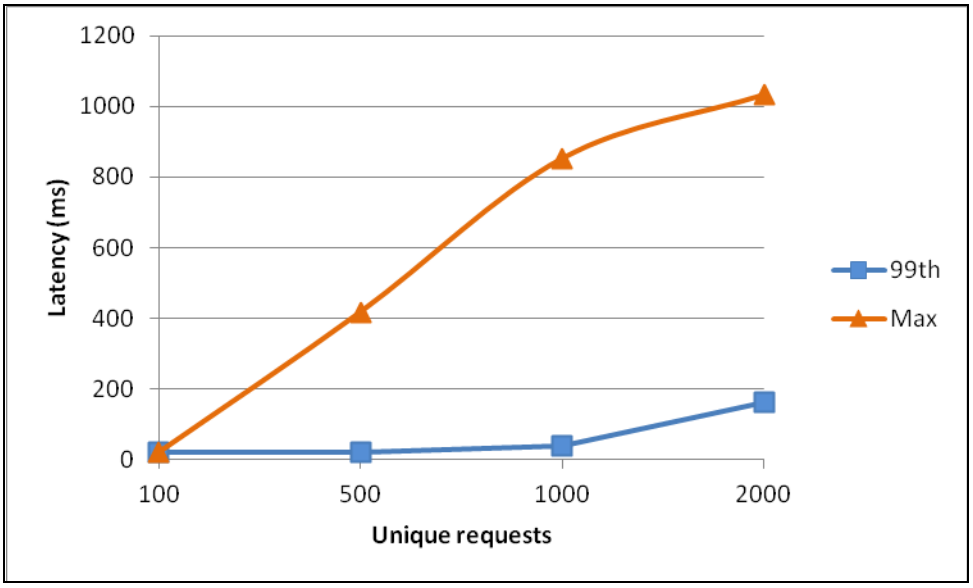


Figure 6.2 – Increasing the number of unique filter requests (worst case)

7 How Caplin's benchmark tests were conducted

The following sections describe the test method used, give information about the test configurations, and detail the test software, test hardware, and the network used.

7.1 Test method

Approach

Each benchmark test followed a similar approach, which was to simulate users logging in to Liberator and requesting filtered and sorted containers. Custom DataSources provided the data to Caplin Refiner, which processed the data before passing the filtered and sorted containers back to the requesting users.

The DataSource applications supplying the data to Caplin Refiner were custom Java DataSources.

Multiple RTTP client connections were simulated using a Caplin benchmarking tool called Benchrttp. Benchrttp is Caplin's scalable client simulator that uses the RTTP protocol to establish concurrent streaming connections to Liberator's HTTP port.

For the benchmark tests described in Section 4, the response times for each Benchrttp user were calculated by subtracting the time the user received the filtered container from the time the user made the filter request.

For the benchmark tests described in Sections 5 and 6, latency in container updates was calculated by measuring the delay between a DataSource updating container objects and the updates being received by a custom StreamLink for Java (SL4J) client.

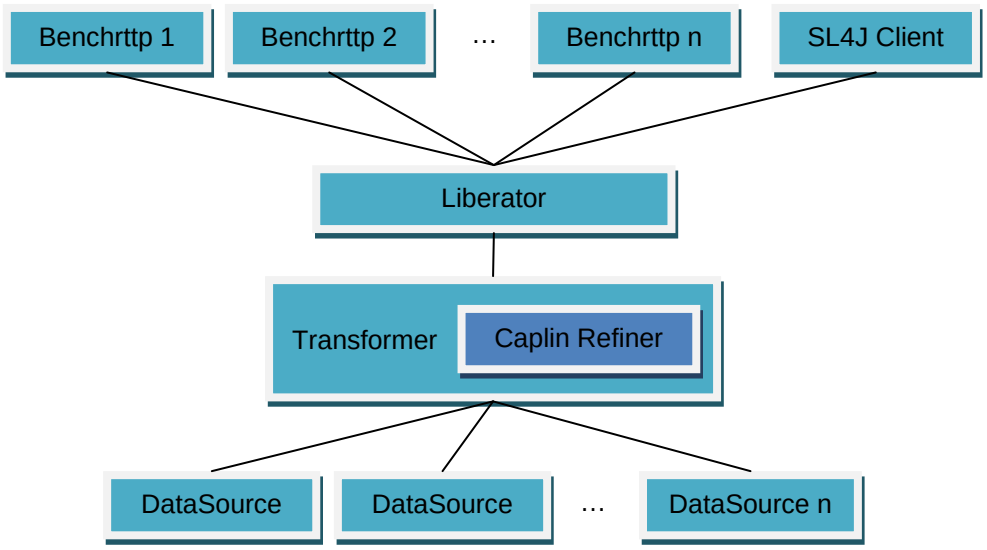
In all benchmark tests, Caplin Refiner loaded the unfiltered container into its cache before each test started. This removed the time to cache the unfiltered container from the test results. In practice, this time is independent of the number of filter requests because Caplin Refiner caches the container when it receives the initial filter request.

Test setup

The components used in each benchmark scenario included:

- A number of Java DataSources supplying data for the container and constituent record objects.
- One Transformer running the Caplin Refiner module.
- One Liberator distributing the filtered responses.
- A number of Benchrttp clients simulating multiple user requests for filtered containers.
- For the benchmarks tests in Sections 5 and 6, a custom StreamLink for Java client measuring the latency of updates to container objects, while Benchrttp clients simply load the system with filter requests.

The following diagram shows the hardware configuration used for the tests.



Each single Benchrttp instance can simulate a maximum of 500 client connections, and the group of instances was spread evenly across 3 host machines. 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 Caplin Refiner limits could be reached before any limits imposed by Benchrttp.

The Liberator and Transformer were each hosted on their own dedicated machine.

Another machine was used to host the SL4J Client and the Java DataSources feeding the Caplin Refiner.

For more detailed information about the hardware and software used to run the tests, see the following sections.

7.2 Test software

Versions

The following table shows the software versions used in these benchmarks.

Name	Version
Caplin Refiner	5.0.1
Transformer	5.1.1
Liberator	5.1.0

Name	Version
Benchrttp	5.1.0
SL4J Client	Custom
Java DataSource	Custom

The operating system used on all the server hardware was Linux – CentOS 5.5. 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**.

Configuration

The following table shows the configuration options that were changed from their default values.

Name	File	Configuration
Caplin Refiner	refiner.properties	Default
Transformer	transformer.conf	Default
	java.conf	JDK 1.7.0 64-bit Server VM -Xms14g -Xmx14g -XX:NewSize=4g -XX:MaxNewSize=4g -XX:+UseConcMarkSweepGC
Liberator	rttpd.conf	object-throttle-off burst-max 0.1 burst-min 0.05 threads-num 4
	java.conf	JDK 1.7.0 64-bit Server VM
Benchrttp	benchrttp.conf	The following options were all varied during testing: • clients (number of clients) • connect-time (time to wait before connecting) • object sets (control what is being requested)

Note: JDK 1.7.0 was used to benchmark Caplin Refiner. If your Caplin Refiner uses JDK 1.6.0, you could get worse latency and response times than those shown in this document.

Java DataSource Application

For the latency benchmarks in Sections 5 and 6, the custom Java DataSource produced updates to container objects at known rates. This allowed the StreamLink for Java client to calculate the latency of updates to the filtered container.

7.3 Test hardware

testlinux1	
Components	SL4J Client, Java DataSources
Vendor	Dell
Model	PowerEdge R415
Processors	Dual P4 4 cores
Memory	2GB
Operating System	CentOS 5.5 (Kernel 2.6.18-194.el5 64bit)

benchlinux2	
Components	Transformer, Caplin Refiner
Vendor	Dell
Model	PowerEdge R415
Processors	2 x Six-Core AMD Opteron™ Processor 4180 2.6GHz
Memory	16GB
Operating System	CentOS 5.5 (Kernel 2.6.18-194.el5 64bit)

benchlinux3	
Components	Liberator
Vendor	Dell
Model	PowerEdge R415
Processors	8 cores (Dual Intel Xeon X3460 2.8GHz Processors (8 cores total))
Memory	4GB
Operating System	CentOS 5.5 (Kernel 2.6.18-194.el5 64bit)

benchlinux4	
Components	Benchrttp
Vendor	Dell
Model	PowerEdge R415
Processors	8 cores
Memory	4GB
Operating System	CentOS 5.5 (Kernel 2.6.18-194.el5 64bit)

benchlinux5	
Components	Benchrttp
Vendor	Dell
Model	PowerEdge R415
Processors	8 cores
Memory	4GB
Operating System	CentOS 5.5 (Kernel 2.6.18-194.el5 64bit)

benchlinux6	
Components	Benchrttp
Vendor	Dell
Model	PowerEdge R415
Processors	8 cores
Memory	4GB
Operating System	CentOS 5.5 (Kernel 2.6.18-194.el5 64bit)

8 Glossary of terms and acronyms

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

Term	Definition
Benchrttp	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 used in conjunction with control scripts from the Caplin Benchmarking kit.
Caplin Liberator	Caplin Liberator is a real-time financial internet hub that delivers trade messages and market data to and from subscribers over any network.
Caplin Refiner	A high performance server-side filtering and sorting module for financial records.
Caplin Transformer	An event-driven real-time business rules engine that hosts **Caplin Refiner**.
Caplin Xaqua	A framework for building single-dealer platforms that enables banks to deliver multi-product trading direct to client desktops. Caplin Xaqua was formerly called "the Caplin Platform".
DataSource	DataSource is the internal communications infrastructure used by Caplin Xaqua's server components such as **Caplin Liberator**, Caplin Transformer.
RTTP	Real Time Text Protocol. Caplin's protocol for streaming real-time financial data from **Caplin Liberator** servers to client applications, and for transmitting trade messages between clients and Liberator in both directions.
StreamLink	The StreamLink libraries connect client applications to **Caplin Liberator** via the **RTTP** protocol. They provide an object oriented API that gives access to RTTP functionality.
Subject mapping	Subject mapping allows **Caplin Liberator** and **Caplin Refiner** to modify the subject of the **RTTP** message it receives when an end user attempts to view or trade a product, and can be used to provide tiered prices to selected users.
Unique request	A request that has a subject that is not currently being processed.
Underling container	A container that has not been filtered or sorted by **Caplin Refiner**. Underling containers are shared between unique requests.
Requested container	A container that has been filtered or sorted by **Caplin Refiner**. Each unique request creates a requested container.

Single-dealer platforms for the capital markets

CAPLIN

Contact Us

Caplin Systems Ltd

Cutlers Court

115 Houndsditch

London EC3A 7BR

Telephone: +44 20 7826 9600

www.caplin.com