<Application name> Performance Tuning
Summary Report

Prepared for:
<Software Package Supplier>

Prepared by:
Equinox Limited

November 2003
Document Control

Author: Richard Leeke
Creation Date: 21 December 2003
Current Release: 1.0
File Name: Sample Performance Tuning Report.doc

Change Record:

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>19 Nov 2003</td>
<td>Document created</td>
</tr>
<tr>
<td>0.2</td>
<td>21 Nov 2003</td>
<td>Updated after internal Equinox review</td>
</tr>
<tr>
<td>1.0</td>
<td>24 Nov 2003</td>
<td>Issued following &lt;Software Supplier/Customer&gt; review</td>
</tr>
</tbody>
</table>

Reviewers:

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deane Sloan</td>
<td>Senior Consultant, Equinox</td>
</tr>
<tr>
<td>Pat Ryan</td>
<td>Senior Consultant, Equinox</td>
</tr>
<tr>
<td>&lt;Client&gt;</td>
<td>Consulting Manager, &lt;Software Supplier&gt;</td>
</tr>
<tr>
<td>&lt;Client&gt;</td>
<td>eBusiness Architect, &lt;Customer&gt;</td>
</tr>
<tr>
<td>&lt;Client&gt;</td>
<td>Architect, &lt;Software Supplier&gt;</td>
</tr>
</tbody>
</table>
Contents

DOCUMENT CONTROL ........................................................................................................... 2

1 EXECUTIVE SUMMARY .................................................................................................. 5
  1.1 Objectives .................................................................................................................. 5
  1.2 Summary of Issues Identified ..................................................................................... 5
  1.3 Expected Impact of Changes in Production .............................................................. 6
  1.4 Recommendations .................................................................................................... 6

2 BACKGROUND .............................................................................................................. 7
  2.1 About This Document ............................................................................................... 7
  2.2 The Problem ............................................................................................................. 7
  2.3 Objectives of Exercise .............................................................................................. 7
  2.4 Approach .................................................................................................................. 7
  2.5 Test Environment ...................................................................................................... 8
  2.6 Caveats ..................................................................................................................... 8

3 FINDINGS ....................................................................................................................... 9
  3.1 Issues Identified ........................................................................................................ 9
    3.1.1 Missing Index on Post Code Table ....................................................................... 9
    3.1.2 Impact of Logging ............................................................................................... 9
    3.1.3 MQ Polling Speed for Printing .......................................................................... 9
    3.1.4 Printing Architecture ......................................................................................... 10
    3.1.5 Repeated Loading of <Transaction> ................................................................ 10
    3.1.6 Explicit Garbage Collection ............................................................................. 10
    3.1.7 JVM Server Option ......................................................................................... 10
    3.1.8 Start<Application component> Serialisation .................................................... 11
    3.1.9 Impact of JVM Version .................................................................................... 11
  3.2 Improvements Achieved ........................................................................................... 12
    3.2.1 Single-User Performance ............................................................................... 12
    3.2.2 Multi-User Performance .................................................................................. 13
  3.3 JVM Memory Usage and Garbage Collection .......................................................... 15
  3.4 Indications of Scalability ......................................................................................... 16
  3.5 Evidence of Stability ............................................................................................... 17
  3.6 Production Database Performance .......................................................................... 17

4 POSSIBLE AREAS FOR FURTHER IMPROVEMENT ...................................................... 18
  4.1 JVM Version Upgrade ............................................................................................. 18
    4.1.1 Concurrent Garbage Collection ....................................................................... 18
    4.1.2 Other Memory Management Options ............................................................. 18
    4.1.3 Better Garbage Collection Diagnostics ............................................................ 19
  4.2 Possible Opportunities for Query Tuning ................................................................. 19
  4.3 <Package> Configuration Options ......................................................................... 19
  4.4 Rebase ..................................................................................................................... 19
  4.5 Multiple JVM Instances Per Server ....................................................................... 19
  4.6 Tuning of <Customer> or <Software Supplier> Code ............................................ 19

5 RECOMMENDED NEXT STEPS .................................................................................... 21
  5.1 Implementation of Changes in Production ............................................................... 21
  5.2 Production Performance Management ................................................................. 21
  5.3 Test Environment Improvements ........................................................................... 22
  5.4 More Comprehensive Diagnostic Analysis ............................................................ 22
  5.5 Capacity Planning ................................................................................................... 23
  5.6 Ongoing Testing Regime ......................................................................................... 23

6 APPENDICES .................................................................................................................. 24
  6.1 Workload Model ....................................................................................................... 24
  6.2 Detailed Performance Results for Final Multi-User Test ........................................ 25
6.3 Detailed Performance Results for Individual Changes ....................................................... 28
   6.3.1 Missing Index on Post Code Table ........................................................................ 29
   6.3.2 Impact of Logging ......................................................................................... 30
   6.3.3 MQ Polling Speed in Printing ......................................................................... 31
   6.3.4 Printing Architecture .................................................................................... 31
   6.3.5 Repeated Loading of Quote ........................................................................... 32
   6.3.6 Impact of Explicit Garbage Collection ............................................................ 32
   6.3.7 JVM Server Option ...................................................................................... 33
   6.3.8 Start <Application component> Serialisation ................................................ 34
   6.3.9 Impact of JVM Version .................................................................................. 35
6.4 Details of JVM Configuration Changes ......................................................................... 36
6.5 Possible Issues Requiring Further Investigation ........................................................ 37
   6.5.1 Zero Impact Queries ...................................................................................... 37
   6.5.2 XSLT Re-compilation .................................................................................. 39
   6.5.3 Scalability of <transaction> ......................................................................... 41
   6.5.4 Further Reduction in Logging ....................................................................... 41
   6.5.5 Thread Configuration and Missing Threads .................................................... 41
   6.5.6 Intermittent <transaction> Error .................................................................... 42
   6.5.7 Exceptions in logs......................................................................................... 42
6.6 Other Observations ................................................................................................. 45
   6.6.1 TE and <External user> System Issues ............................................................ 45
1 Executive Summary

1.1 Objectives

Equinox was engaged by <Software Supplier> to assist with the diagnosis and resolution of issues impacting on the performance and scalability of the <Application name> application. The severity of the issues was such that the rollout of <Application name> to the <external users> had been placed on hold.

Note that the scope of this exercise did not include making an accurate assessment of the capacity of the production infrastructure, or estimating the number of servers required to support projected peak user numbers.

1.2 Summary of Issues Identified

A number of significant issues have been identified, impacting various components of the architecture. Fortunately, all of these issues are relatively easily resolved, whilst offering significant gains in system performance and scalability.

The most significant improvements identified were in the following areas.

- The Post Code table in the database, which is accessed frequently in the course of each transaction, did not have appropriate indexes to support the queries used. Adding this one index roughly halved the system time taken to process a new business transaction for a single user. This change has already been implemented in production.

- A component involved in launching the <Application name> browser window from the <External user> environment was “single-threaded” – i.e. only one user could be performing this type of operation at once. This issue severely constrained the scalability of the system, especially as there is only a single instance of this component in the solution (i.e. adding more servers would not have helped performance significantly while this issue was outstanding).

- The application explicitly requests that the Java Virtual Machine performs “garbage collection” (re-cycling freed memory) frequently. Garbage collection is a resource intensive process - performing garbage collection too frequently has a severe impact on overall system performance.

- The <Package> environment is supplied with a version of the Sun JVM that is optimised for a workstation rather than a server environment (this was possibly due to Sun’s licensing model, although this is no longer a restriction). Moving to the server version of the JVM offers significant benefits in this environment.

Note that at the outset of the exercise, it was believed that there was a significant memory leak occurring somewhere in the application, based on an incident in production when the JVM failed due to lack of memory. This testing exercise did not manage to reproduce this behaviour, and it is possible that the original issue may have been resolved. Evidence on this topic is not conclusive, however, due to the limited scope of the testing conducted.
1.3 Expected Impact of Changes in Production

It is expected that implementation of the changes discussed in this document will result in a substantial reduction in user-response times (of the order of a 60% or greater reduction), and more importantly much better scalability of the system (supporting several times as many concurrent users). It is not possible to be precise about the magnitude of the improvements expected in production, due to the differences between the test and production environments, but these indicative figures are expected to be of the correct order of magnitude.

The scalability improvements will mean that additional capacity can be added by increasing the number of servers as user numbers increase.

1.4 Recommendations

Most of the changes identified in this document can be implemented relatively quickly. These should be fast-tracked to production to allow the rollout to proceed.

A capacity and performance management process should be established for the production environment, to allow the performance of the system to be managed proactively.

Further testing of the application should be performed, both to diagnose and resolve outstanding issues, and to establish metrics for system capacity planning purposes.

A permanent performance-testing environment should be established to allow the <Application name> development and/or test team to conduct ongoing tuning and to verify the performance characteristics of new releases before they are released to production.
2 Background

2.1 About This Document

This document describes the outcome of a short (two week) exercise to diagnose and resolve performance issues impacting the rollout of <Customer>'s <Application name> <business area> application to <end-users>.

2.2 The Problem

The performance of the <Application name> application in production has degraded severely as the rollout to <end-users> has progressed. In addition, one component of the application, the <Application component> Enterprise Server, has failed on one occasion in production, due to running out of memory.

The impact has been sufficiently severe that the rollout of <Application name> has been put on hold, pending resolution of the issues.

2.3 Objectives of Exercise

The primary objectives of the exercise were as follows:

- Diagnose the cause of the performance issues.
- Identify potential system configuration, software or hardware changes required to address the issues.
- Estimate the number of concurrent users that can be supported by a given server configuration, for the <Application component> Enterprise component of the solution.

2.4 Approach

As the performance issues were only evident under multi-user load, the approach taken to this exercise was as follows. A set of test scripts was developed to allow a repeatable multi-user load to be applied to the system, using Rational TestStudio. The test scripts exercised a single “representative” transaction, consisting of a <end user> <transaction description>.

Repeated test runs were then conducted, measuring anything that moved (and some things that didn’t) across all components of the infrastructure. Measurements and results from both the Rational tools and other sources were then analysed to isolate problem areas. Specific, experimental test scenarios were developed to explore particular issues or test hypotheses.

Data captured and analysed included the following:

- Rational test result logs
- HPROF output
- Stack dumps
- Perfmon logs across all platforms
- Application logs
- JVM output
- MQ logs
• Zero Impact (database monitoring) log files showing SQL traffic

2.5 Test Environment

The test environment used had a number of dedicated platforms, to isolate it as far as possible from external influences. Given the compressed timeframe for the exercise, however, it had not been possible to build a completely isolated environment.

Separate desktop machines supported the following components:

• <Application component>Server (dedicated)
• IIS and JRUN (servlet container for Start<Application component> and <Application component>Dispatcher components)
• <Component> and <External user system> Transaction Executive
• Rational TestStudio platform to conduct the tests

The test environment also accessed the following shared components:

• Oracle database (a dedicated database within a shared Oracle instance)
• MQ server accessing shared mainframe and document production facilities

This configuration was sufficient to achieve useful progress, although the ability to isolate the testing from influences due to the shared database and MQ environments would have provided major benefits.

2.6 Caveats

This was a tuning and diagnostic exercise – the scope did not include capacity planning. Performance results quoted in this document should be taken as indicative, particularly in terms of the relative impact of many of the changes. Actual performance and capacity figures in the production environment are likely to be significantly different, for two reasons

• There has been no attempt to establish an accurate production workload model (i.e. how many users carry out how many transactions of what types per hour).
• The test platforms used were (high specification) single CPU workstations. The production server class machines will have significantly different performance characteristics. This will be particularly true for the <Application component>Server platform, since this is the limiting component of the infrastructure and the production platform is a dual CPU server.

The differences between the test and production environments also meant that there was no value in monitoring for other classes of resource constraints (such as I/O or communications bottlenecks) during the testing. There is no evidence to suggest that there are any issues in these areas in production, however.

It was noted that numerous errors were reported in the system logs during the testing. It appears that these issues also occur in production, but it is not clear what impact these are having on the performance of the system.
3 Findings

3.1 Issues Identified

This section presents a brief summary of each of the significant issues identified and resolved (or for which a solution was identified), during the course of the tuning exercise. Evidence of the impact of each of these changes is presented in Appendix 6.2.

3.1.1 Missing Index on Post Code Table

The “representative” transaction used for testing (<type of transaction>) performs 26 separate queries (5 distinct SQL statements) against the PCDV (Post Code) table. The Post Code table has around 45,000 rows, and had no indexes defined, which resulted in each of these queries taking around 2 seconds. A single index has been added to the column “pcode1”, supporting all of the queries efficiently. This resulted in a 44% reduction in the total single user response time to complete the test transaction.

This change will have reduced the load on the back-end Oracle database server, but will have made no difference to the total work performed by the middle-tier application servers in processing a transaction.

3.1.2 Impact of Logging

In production, the <Application name> application is run with comprehensive (debug level) logging enabled, in order to provide diagnostic information for support purposes. This level of logging produces hundreds of MB or GB of logs per day.

The logging level was reduced to “ERROR”, which resulted in a further 7% reduction is single-user response time.

This change will have reduced the workload of each of the servers for which logging was reduced – primarily the <Application component>Server.

3.1.3 MQ Polling Speed for Printing

Document production on completion of a transaction works by sending a request via MQ to a document production server, which generates a PDF file of the required document. The application then waits for a response message back from MQ before continuing. A new business transaction may involve up to three documents, requiring three round-trips via MQ.

The response from MQ is obtained by polling a queue. The polling delay was configured at 5 seconds. On average, this would be expected to introduce a 2.5 second delay per document.

An initial change was made in the test environment to reduce the polling delay to 1 second. This reduced the time for the test transaction (which requests one document) by around 2 seconds.
3.1.4 Printing Architecture

The printing architecture was designed to avoid the need for the polling mechanism described above, completely. The application knows the location of the generated PDF file, so does not need to wait for a response to the request. It is planned to change the application to take advantage of this feature.

Note that neither change described to the printing solution will have made much difference to the total workload on the application servers, just to the response time for the individual user transaction.

3.1.5 Repeated Loading of <Transaction>

The transaction acceptance process redundantly loaded the <transaction> twice, adding a couple of seconds to this step. The redundant step was eliminated, reducing the elapsed time and the server workload.

3.1.6 Explicit Garbage Collection

The <Application name> application explicitly calls the garbage collector to free JVM memory, rather than letting the JVM decide when this is necessary. It is understood that this has been introduced following an incident in production when the JVM failed with an out of memory exception.

Each time a full garbage collection operation occurs the JVM has to block all other processing for the duration of the operation. Forcing a full garbage collection more frequently than it is needed effectively causes the JVM to “get by” with less memory (since garbage collection frees memory before all memory available to the JVM has been used), however this has a couple of detrimental effects on the performance of the system:

- It means that more of the server’s CPU time is spent on the overhead task of memory management, and less on actual useful processing.
- It has the impact of reducing concurrency (since garbage collection is single threaded in version 1.3 of the JVM). This is particularly significant on the dual processor production servers, where it effectively means that one of the CPUs is idle whenever a garbage collection operation is underway.

Disabling explicit garbage collection in the JVM almost halved the average response time during a 20-user test. The impact is likely to be even more dramatic in the production environment, with a dual-CPU server and heavier user load.

3.1.7 JVM Server Option

The <Package> system is distributed with the “client” version of the JVM (which used to be referred to as “hotspot”). This version is really optimised for typical workstation usage, rather than for server usage. There is also a “server” version available optimised for typical server style processing.

Installation and use of the server version of the JVM gave a further 10% to 15% reduction in response times. Note that it is possible that the impact of this JVM may be even greater in the production environment under heavier user loads and with a dual CPU server.
Note that it appears that <Package Vendor> may have been prevented by the Sun licensing agreement from distributing the server version of the JVM in the past, but as of 1.3.1_08 the server version of the JVM may be freely distributed with the JRE.

3.1.8 Start<Application component> Serialisation

Two steps in the new business transaction tested launch the <Application name> application in a separate browser window, from the <External user system> Executive. These steps (<transaction steps>) both call the Start<Application component> servlet, which is running in the servlet container in iPlanet in production and was running under JRUN in the test environment. Start<Application component> in turn calls the <Application component>Dispatcher servlet to invoke various <Application component>Server functions.

It turned out that an implementation detail of the calls from Start<Application component> to <Application component>Dispatcher caused this step to be serialised (i.e. only one user at once could be making a call to <Application component>Dispatcher from Start<Application component>). (The Java API component used for the communication was declared as a static object.) As some of these requests were lengthy operations (such as document printing requests), this had the effect of severely reducing the concurrency and hence throughput of the system.

Resolving this issue reduced the multi-user response time for the steps involving the servlets, and increased the overall throughput of the system significantly.

3.1.9 Impact of JVM Version

Version 1.4.2 of the JVM offers several new features designed to improve performance in a server environment, as discussed more fully in section 4.1.

None of the additional optional features were tested explicitly, but a comparison between version 1.3.1_09 (current production version) and 1.4.2 of the JVM was performed. The 1.4.2 results indicated a small improvement in performance, even with none of the new options selected, however the results were a little inconclusive as one of the test runs appear to have been influenced by an external issue (affecting some of the <transaction> steps).

Based on past experience, it is expected that some of the additional features in 1.4.2 could offer a significant improvement.
3.2 Improvements Achieved

This section presents summary statistics showing the extent of the performance improvements achieved through the changes described in section 3.1. Further statistics showing the impact of the individual changes are included in appendix 6.3.

3.2.1 Single-User Performance

The graphs in this section show a comparison of the single user performance between the initial state of the test system and the final state after the implementation of all changes (except the use of the 1.4.2 JVM).

The comparisons are shown as “relative” and “absolute” charts of the average response times recorded for all timing points for the test transaction. Each of the coloured bars represents an individual timing point. The timing points are listed in the order shown appendix 6.1. The average ratio of the response times for the relative report and the total “system” time taken by all timing points for a single transaction, for the absolute reports, are also shown.

Relative Performance Comparison

```
<table>
<thead>
<tr>
<th>Ratio</th>
<th>Average Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.46</td>
<td></td>
</tr>
</tbody>
</table>
```

Absolute Performance Comparison

```
<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>85.74</td>
<td>30.98</td>
</tr>
</tbody>
</table>
```

The total single user response time to complete all steps in a transaction was thus reduced to 36% of the original time.
3.2.2 Multi-User Performance

The following graph shows the performance of a selection of timing points from the test transaction, as the number of concurrent users was increased to 50. No directly comparable figures were obtained prior to the tuning, since response times at even 20 concurrent users were so long that transactions failed due to timeouts.

From the graph it can be seen that the “knee” in the performance curve occurred at around 30 users, although these tests should not be taken as indicating absolute performance expected in the production environment.

It is noticeable that the <Txn 2>, <Txn 3> and <Txn 4> transaction steps degrade particularly sharply with increasing user numbers (<transaction> for the other <equivalent> types also showed this behaviour). This may be an indication of some further serialisation issues, however there was not time to investigate this further.

The two scatter charts overleaf show the performance of the Get Acceptance transaction step during multi-user tests before and after the tuning changes. In the first test 20 users caused the system to become overloaded and transactions to time out. After tuning, the system worked successfully up to 50 users, although response times were extended and unlikely to be acceptable at this load.

The graphs show the response time (ms, y-axis) over the course of the test (ms, x-axis). User numbers were varied over the two tests as explained in detail below.
Before:
This test shows the performance of a multi-user test in the baseline application and JVM configuration, with the exception that the missing index on the Post Code table had been created.

During this test the number of concurrent users was increased progressively from 5 to 10 to 20 to 30. At 30 users the Get Acceptance transaction response time became excessive and the transaction timed out after the default TestStudio timeout time of 240 seconds (the first horizontal line on the chart). The number of active users was then reduced to 5 for a while to allow the system to catch up, before being increased to 20 once more. Even at 20 users the system became backlogged and transaction timeouts began to happen once again (second horizontal line).

After:
This test shows the performance after all changes described in section 3.1 (except the JVM version) had been applied (i.e. the test used the 1.3.1_09 JVM with the server option and explicit garbage collection disabled).

This run progressively increased the number of users by 5 every 20 minutes, ranging from 5 to 50 users over the three hour twenty minute run.
The timescale of the graph is in ms, so 6000000 is 100 minutes, which corresponds to 25 users and 9000000 is 2.5 hours, which corresponds to 35 to 40 users.

Note the major spike at around 1050000, or 45 users, indicating an extended blockage of some sort. There was not time to investigate whether this represented a real issue under stress or whether some external environmental issue caused this.

### 3.3 JVM Memory Usage and Garbage Collection

The following scatter charts show the full garbage collection events that occurred during the 3 hour, 5 to 50 user test run described above. The JVM logs statistics for the memory used and freed and the elapsed time of the garbage collection operation. Unfortunately, the time when the event occurred is not recorded, so it is not possible to determine the frequency of the garbage collection events, although this will certainly have become more frequent as user numbers increased.

The first chart shows JVM memory usage before (pink) and after (blue) each garbage collection event. This clearly shows that even at 50 users, under half of the total JVM memory was used. However, some headroom must be allowed between the amount used after garbage collection and the total available to the JVM, so it is difficult to predict how many users could be supported with this memory configuration, without further testing to explore this issue.

![Memory Usage Before and After Full GC - 50 User Test Run](image)

The second chart shows that the elapsed time to complete full garbage collection increased as user numbers (and hence total memory usage) grew, as the garbage collector had to inspect the state of more objects on each pass. The longest garbage collection operation took over 4 seconds, and on a few occasions two full garbage collection operations appear to have happened consecutively. This means that the server would be completely frozen, not processing any user requests for several seconds.

From the memory headroom shown above, it would appear possible to increase user numbers to of the order of 100 concurrent users in a single JVM. It is likely that with the corresponding increase in memory usage, the elapsed time to complete full garbage collection operations could also double. This would probably also be
accompanied by an increase in the frequency of full GCs (since there would be less headroom between used and available memory). The impact of frequent, prolonged garbage collection operations on system performance is likely to be substantial.

3.4 Indications of Scalability

Prior to making the various changes discussed in this document, none of the platforms in the test infrastructure was heavily loaded, even during a high volume test. This was an indication of bottlenecks causing serialisation of processing, particularly the issue with the Start<Application component> servlet.

Following the changes, the <Application component>Server platform was CPU constrained during a high volume test. This is a good thing, since it means that the system is able to make full use of the resources available to it.

As discussed, it is virtually impossible to predict the capacity of the production <Application component>Server platforms, based on the testing, since there are too many unknowns. However, a few general observations are possible.

It appears that the major obstacles to the scalability of the system have been identified and eliminated.

On the basis of the test transaction, it appears that there may be sufficient memory headroom within the 1.6 GB allocated to the JVM to support of the order of 100 concurrent users.

The performance of the test server degraded sharply beyond about 30 concurrent users. Assuming that the test workload profile was realistic (and this is a major assumption), this would suggest that the production servers, which are dual CPU, could be CPU constrained at around 60 concurrent users. Note that this estimate is very approximate, a more accurate estimate requires detailed workload modelling and testing on a production specification platform.
As user numbers and memory usage increases, the impact of garbage collection is also likely to increase, but it is impossible to quantify this without testing.

### 3.5 Evidence of Stability

As discussed earlier, testing was limited to one specific transaction type (albeit exercising a substantial proportion of the system). It is therefore not possible to make any definitive statements about the stability of the system under load, however some observations based on the testing that was completed are justified.

Several prolonged test runs were attempted. Some of these were successful (such as the 5 to 50 user run); others failed due to database or mainframe availability issues. However even the failed runs provided useful insight into the behaviour of the system. Various metrics were collected during each of these runs, including server and per process operating system level statistics, such as memory and "handle" usage.

Inspection of these logs after each run showed no evidence of memory leaks or other unexplained growth in resource usage for the transaction tested. However, more complete test coverage and longer duration runs would be needed to achieve a high degree of confidence that there are no issues of this sort anywhere in the application.

### 3.6 Production Database Performance

The missing index on the Post Code table was identified by monitoring SQL query performance in the test environment with the database performance-monitoring tool Zero Impact, from SQL PowerTools. As the testing was only exercising one specific transaction type, it was therefore decided to monitor the production database traffic for a while, to identify any other database issues.

Based on an hour’s monitoring, database access appears to be generally well behaved. No other high-impact queries were identified, although there might be scope for some small improvements in one or two areas, as discussed in appendix 6.5.1.
4 Possible Areas for Further Improvement

4.1 JVM Version Upgrade

Version 1.4.2 of the JVM allows various additional features, which potentially offer significant performance advantages, however <Package Vendor> does not yet support 1.4.2. It is worth confirming with <Package Vendor> their plans for 1.4.2 support.

It may also be worthwhile to measure the benefits of these features in the test environment, to help determine how much effort is warranted in seeking <Package Vendor> support, or even to evaluate the risk/benefit trade-off of moving to an unsupported release.

A good summary of the Garbage Collection features in the 1.4.2 JVM can be found at the following site: http://java.sun.com/docs/hotspot/gc1.4.2/faq.html.

4.1.1 Concurrent Garbage Collection

Under 1.3, the JVM freezes all threads for the duration of each full garbage collection operation, and garbage collection is single-threaded. As discussed in section 3.3 above, in the 50 user tests performed in the test environment, full garbage collection took up to 4 seconds, at a time when under half of the JVM memory was committed (i.e. ineligible for garbage collection). It is therefore possible that full GCs in production may take longer.

With 1.4.2 of the JVM, full garbage collection can be multi-threaded, and does not freeze all other activity for the duration of the operation.

As the production servers are dual processors, the advantage of concurrent and multi-threaded operation are likely to be even greater than on the single CPU test server.

4.1.2 Other Memory Management Options

The 1.4.2 JVM has various other memory management options. Based on previous experience and studying the documentation, it appears that there may be merit in exploring at least the following settings:

-XX:+AggressiveHeap,
-XX:NewSize and
-XX:MaxNewSize

Brief initial investigation of these options in the test environment was not successful. The JVM would not even start with these settings selected, although identical settings used in a test environment at Equinox did not have this problem. There was insufficient time to investigate the reasons, although it is possible that this was because the <Customer> test server was running Windows Server 2003.
4.1.3 Better Garbage Collection Diagnostics

Even if 1.4.2 is not a candidate for production usage until <Package Vendor> support is available, the improved garbage collection diagnostics may be helpful for tuning purposes in the test environment.

4.2 Possible Opportunities for Query Tuning

There is perhaps some minor scope for improvement in a few database queries, as discussed in section 6.5.1.

4.3 <Package> Configuration Options

The <Package> environment has a number of configuration parameters affecting factors such as the number of threads available to the <Application component>Server engine and how these threads are initialised. Once the issues identified in this document have been resolved, it may be worth exploring the impact of these options.

4.4 Rebase

The address space allocated to the JVM needs to be contiguous. The maximum contiguous memory that can be allocated is constrained by the addresses at which certain DLLs are located. It is possible to overcome this restriction by a process known as "rebasing", which effectively modifies the jvm.exe file to cause it to load certain DLLs at different locations. If the JVM proves to be memory constrained this option may be worth considering.

4.5 Multiple JVM Instances Per Server

If testing of a production configuration <Application component>Server platform, or observation of the behaviour of the production platforms shows that the <Application component>Server JVM is memory constrained, it may be viable to run two instances of <Application component>Server per platform. This would require considerable re-configuration and careful testing.

4.6 Tuning of <Customer> or <Software Supplier> Code

It appears that the capacity of the system will be constrained by CPU utilisation on the <Application component>Server platforms. Whilst the architecture includes multiple <Application component>Server platforms, and capacity can be increased by adding extra servers, it is clearly worth trying to minimise the number of servers required. As well as the purchase cost of the servers and licences, increasing the number of servers increases the administration complexity and hence operating cost of the system.

This conclusion suggests that the best return from ongoing tuning efforts is likely to come from work that reduces the CPU load on the <Application component>Server platforms. In practice, the most likely scope for such changes will be in the <Customer> or <Software Supplier> developed code, simply because that can be modified locally.

As a general rule in this type of tuning, the biggest wins tend to come from identifying processing that can be eliminated completely, or where the frequency of execution can be reduced. Eliminating the duplicate retrieval of the <transaction> is a good
example of this. Identifying scope for caching and re-using results is another common example.

There was not time during the onsite tuning activity to focus on this area in any detail, however brief analysis of HPROF and Java debugging interface output obtained at the end of the tuning exercise highlighted the following three routines as the most frequently seen <Customer> specific code.

- `<customer>.util.mq.MQDispatcher.requestReply`
- `<customer>.<package>Interaction.setParameterValue`
- `<customer>.<package>.XSLHelper.performXSLTransformation`

The first of these appears to be simply the polling mechanism for retrieving replies from MSMQ. No investigation has been done of the second. The third possibly indicates scope for optimisation in the area of XML transformation during acceptance processing. This is discussed further in appendix 6.5.2.
5 Recommended Next Steps

5.1 Implementation of Changes in Production

Most of the changes identified in section 3.1 should be implemented in production as soon as practicable. This is likely to be governed by the development, test and release cycle.

Most of the identified improvements can be achieved through either configuration changes or relatively minor application changes. All except the proposed change in the printing architecture were tried successfully in the test environment during the course of the tuning exercise, and it is understood that the printing change is straightforward. All changes will, of course, need to be tested thoroughly through the acceptance testing process, however.

It is not practicable to implement two of the changes immediately:

- Reducing the logging level is probably not practical at this stage, since the detailed log files are needed to help with production support issues. If possible, the use of the logging mechanism should be reviewed with a view to differentiating between different classes of logging messages, to avoid the need for the comprehensive logging currently used.

- Moving to version 1.4.2 of the JVM will require far more substantial regression testing than the other changes proposed, and is also likely to be constrained by <Package Vendor> support issues.

In the longer term, the goal should be to reduce logging substantially and migrate to a later JVM (1.4.2 or higher).

Note that the effect of explicit garbage collection was eliminated by the use of an option causing the JVM to ignore explicit garbage collection requests from the application code. The application is still requesting garbage collection, however. It is possible that there may be some residual overhead associated with these requests (though no attempt has been made to quantify either the impact or the frequency of these requests). It is suggested that the calls to the garbage collector should be removed from the application code, although this will probably have little noticeable impact on performance.

5.2 Production Performance Management

It is strongly recommended that <Customer> should introduce comprehensive, targeted production performance monitoring, to allow trends in workload and performance to be detected and facilitate performance and capacity management. This requires the capture and ongoing analysis of appropriate metrics from each of the components of the infrastructure. This is not just a question of gathering simple operating system and network statistics; it is generally necessary to “look inside the black box”.

This is likely to require specialised tools (such as the Compuware tools that <Customer> is currently considering).
Some specific monitoring, targeting internal JVM memory usage could be achieved by extracting and analysing full garbage collection statistics from the JVM output. Collection of process level statistics using Perfmon for each JVM and the servlet containers would also be worthwhile. For maximum value, it is worth ensuring that these statistics are easily available (for example via an intranet site) and that extended history is maintained to allow trends to be monitored.

5.3 Test Environment Improvements

In order to facilitate ongoing performance testing and tuning activities it, would be desirable to enhance the performance-testing environment established for the current exercise in a number of ways.

For performance testing to be most effective, it is essential to have a dedicated test environment, isolated as much as practicable from external influences. It is also desirable for the test environment to reproduce the production environment as closely as possible, especially for the limiting components of the infrastructure.

It is particularly important to be able to conduct long duration (soak) tests, to identify resource creep, such as memory leaks. It is also desirable to be able to conduct extended test runs overnight, leaving the daytime free for analysis of test results and the execution of short (experimental) test runs.

During the current exercise, there were three notable aspects of the test environment that materially impacted the effectiveness of the testing.

1. The test Oracle database was running within a shared Oracle instance, which was taken down overnight for backups. This prevented prolonged overnight test runs.

2. <Transaction> and printing occurred via a shared MQ environment and <transaction> used a shared mainframe environment. The mainframe was unavailable for extended periods overnight, and the shared use of the MQ environment prevented the use of “stubs” to eliminate the dependency on the mainframe.

3. The <Application component>Server platform (which is the most heavily used of the application infrastructure components) was a (high specification) single CPU desktop machine, as opposed to the dual CPU Compaq DL380 used in production. It is not possible to draw conclusions about production capacity in absolute terms with this configuration.

To maximise the benefits from ongoing performance testing work it would be desirable to establish dedicated Oracle (or DB2) and MQ environments. If any accurate capacity planning figures were required, it would also be necessary to secure a production specification <Application component>Server platform, at least for the duration of such a capacity planning exercise.

5.4 More Comprehensive Diagnostic Analysis

The testing during the current exercise has exercised only a single <external user> new business transaction. The issues affecting multi-user scalability also prevented issues of memory utilisation and garbage collection from being explored fully within the timeframe of the exercise.
After the changes identified in this document have been implemented, it is recommended that additional diagnostic testing should be carried out, with a particular focus on the following areas:

- Additional <transaction> types.
- <Internal user> transactions.
- Longer runs to confirm the stability of memory usage.
- Further investigation of garbage collection.

### 5.5 Capacity Planning

The current tuning exercise has provided comparative performance figures for the various changes explored, but has not allowed production server capacity to be estimated. Effective capacity management of the production environment requires a reliable assessment of the capacity of the various production infrastructure components – particularly the <Application component>Server platforms, since these are almost certainly the limiting components.

Testing to estimate production capacity would require a production specification <Application component>Server platform (at least). A workload model based on production utilisation statistics would need also to be developed.

An alternative approach to establish some indication of the capacity of the production <Application component>Server platforms might also be possible. As there are multiple, load-balanced, servers, it may be possible to make some useful measurements of utilisation and response times while deliberately limiting the number of servers available. However, this approach has various limitations, which would make it hard to draw reliable conclusions.

### 5.6 Ongoing Testing Regime

The complexity of the <Application name> environment is such that new performance issues will inevitably be introduced as modifications and enhancements are implemented.

It is recommended that a performance-testing environment should be retained as part of the development and test infrastructure, to allow ongoing both ongoing reproduction and diagnosis of issues and the establishment of an ongoing performance-testing regime as part of routine application regression testing.
6 Appendices

6.1 Workload Model

For the purposes of the tuning exercise, a “typical” New Business transaction was defined, involving the creation and acceptance of a new transaction type. The table below shows the average user think time allowed per transaction step for this model transaction.

Note that this model was based on discussions with members of the project team and assumptions based on experience. As the objective of the exercise was to compare relative performance with various tuning changes, rather than to determine any estimates of absolute capacity metrics, the model was not based on research of actual production usage, in order to minimise the time required to develop the model.

<table>
<thead>
<tr>
<th>Transaction Step</th>
<th>Think Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login Page</td>
<td>5</td>
</tr>
<tr>
<td>Submit Login</td>
<td>5</td>
</tr>
<tr>
<td>Select New Business</td>
<td>20</td>
</tr>
<tr>
<td>Launch &lt;Application name&gt;</td>
<td>5</td>
</tr>
<tr>
<td>Select Account</td>
<td>5</td>
</tr>
<tr>
<td>Select Suburb</td>
<td>5</td>
</tr>
<tr>
<td>&lt;Txn&gt; Questions</td>
<td>10</td>
</tr>
<tr>
<td>&lt;Txn&gt; OK</td>
<td>5</td>
</tr>
<tr>
<td>General &lt;Txn&gt;</td>
<td>5</td>
</tr>
<tr>
<td>Business Search</td>
<td>10</td>
</tr>
<tr>
<td>Submit Business Search</td>
<td>20</td>
</tr>
<tr>
<td>Select Business</td>
<td>15</td>
</tr>
<tr>
<td>&lt;Txn &gt; Tab</td>
<td>10</td>
</tr>
<tr>
<td>&lt;Step 1 1&gt;</td>
<td>5</td>
</tr>
<tr>
<td>&lt;Step 1 2&gt;</td>
<td>5</td>
</tr>
<tr>
<td>&lt;Step 1 3&gt;</td>
<td>15</td>
</tr>
<tr>
<td>&lt;Step 1 4&gt;</td>
<td>20</td>
</tr>
<tr>
<td>&lt;Step 1 5&gt;</td>
<td>15</td>
</tr>
<tr>
<td>&lt;Step 1 6</td>
<td>10</td>
</tr>
<tr>
<td>&lt;Step 2 1&gt;</td>
<td>10</td>
</tr>
<tr>
<td>&lt;Step 2 2&gt;</td>
<td>5</td>
</tr>
<tr>
<td>&lt;Step 2 3&gt;</td>
<td>10</td>
</tr>
<tr>
<td>&lt;Step 3 1&gt;</td>
<td>10</td>
</tr>
<tr>
<td>&lt;Step 3 2&gt;</td>
<td>10</td>
</tr>
<tr>
<td>&lt;Step 3 3&gt;</td>
<td>10</td>
</tr>
<tr>
<td>&lt;Step 3 4&gt;</td>
<td>10</td>
</tr>
<tr>
<td>&lt;Step 3 5&gt;</td>
<td>15</td>
</tr>
<tr>
<td>&lt;Txn&gt; Summary</td>
<td>30</td>
</tr>
<tr>
<td>&lt;Txn&gt; Period</td>
<td>30</td>
</tr>
<tr>
<td>Complete Tab</td>
<td>10</td>
</tr>
<tr>
<td>Close Now</td>
<td>10</td>
</tr>
<tr>
<td>End / Save</td>
<td>15</td>
</tr>
<tr>
<td>(Review Transaction)</td>
<td>60</td>
</tr>
<tr>
<td>Get Acceptance</td>
<td>5</td>
</tr>
<tr>
<td>Logoff</td>
<td>0</td>
</tr>
</tbody>
</table>
6.2 Detailed Performance Results for Final Multi-User Test

This section shows more detailed timing figures for the results presented in section 3.2.2. The graphs and figures show relative and absolute performance for all transaction types, for various user loads, from the final test run. Figures presented show the 95th percentile response time for each transaction step at each user load. This effectively indicates that 95% of user transactions will complete each step in less than or equal to the response shown, at the given user load.

Note, however, that these figures are for the test environment and the test workload model. These numbers may be taken as indicative of the behaviour as workload increases, but do not represent an accurate model of expected production performance.

A few anomalous results can be seen if these results are studied in detail. For example, the 95th percentile, 10-user response time for the “Business Search” step is considerably higher than the 20 or 30-user value. This reflects the fact that two of the results for this transaction during the 10-user phase of the test were considerably slower than the surrounding values. One other transaction type showed a similar effect at the same time. This probably reflects a brief, transient issue – possibly due to the test environment not being fully isolated. Similarly, all <transaction> operations for all three <transaction> types slowed down significantly for a brief period early in the test run. This was almost certainly due to an issue in the shared MQ or mainframe environment.
### Relative Performance Comparison

![Graph showing relative performance comparison](image)

<table>
<thead>
<tr>
<th>Step</th>
<th>1 User</th>
<th>10 Users</th>
<th>20 Users</th>
<th>30 Users</th>
<th>40 Users</th>
<th>50 Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Search</td>
<td>1.00</td>
<td>8.63</td>
<td>3.25</td>
<td>5.38</td>
<td>22.13</td>
<td>66.88</td>
</tr>
<tr>
<td>Close Now</td>
<td>1.00</td>
<td>2.64</td>
<td>3.04</td>
<td>8.52</td>
<td>12.23</td>
<td>17.23</td>
</tr>
<tr>
<td>Complete Tab</td>
<td>1.00</td>
<td>1.83</td>
<td>3.33</td>
<td>6.05</td>
<td>9.17</td>
<td>49.55</td>
</tr>
<tr>
<td>&lt;Step 3 1&gt;</td>
<td>1.00</td>
<td>2.50</td>
<td>3.60</td>
<td>10.42</td>
<td>12.54</td>
<td>24.33</td>
</tr>
<tr>
<td>&lt;Step 3 2&gt;</td>
<td>1.00</td>
<td>2.27</td>
<td>3.02</td>
<td>5.89</td>
<td>10.39</td>
<td>42.28</td>
</tr>
<tr>
<td>&lt;Step 3 3&gt;</td>
<td>1.00</td>
<td>1.65</td>
<td>3.15</td>
<td>8.38</td>
<td>12.12</td>
<td>60.38</td>
</tr>
<tr>
<td>&lt;Step 3 4&gt;</td>
<td>1.00</td>
<td>2.86</td>
<td>6.90</td>
<td>11.71</td>
<td>19.05</td>
<td>28.62</td>
</tr>
<tr>
<td>&lt;Step 3 5&gt;</td>
<td>1.00</td>
<td>2.71</td>
<td>3.48</td>
<td>9.94</td>
<td>19.43</td>
<td>27.90</td>
</tr>
<tr>
<td>End / Save</td>
<td>1.00</td>
<td>1.27</td>
<td>1.43</td>
<td>2.33</td>
<td>4.51</td>
<td>8.22</td>
</tr>
<tr>
<td>&lt;Step 1 1&gt;</td>
<td>1.00</td>
<td>4.99</td>
<td>5.98</td>
<td>8.55</td>
<td>17.35</td>
<td>46.81</td>
</tr>
<tr>
<td>&lt;Step 1 2&gt;</td>
<td>1.00</td>
<td>2.21</td>
<td>4.44</td>
<td>7.81</td>
<td>16.32</td>
<td>35.60</td>
</tr>
<tr>
<td>&lt;Step 1 3&gt;</td>
<td>1.00</td>
<td>2.43</td>
<td>9.38</td>
<td>5.05</td>
<td>16.19</td>
<td>42.70</td>
</tr>
<tr>
<td>&lt;Step 1 4&gt;</td>
<td>1.00</td>
<td>4.08</td>
<td>3.71</td>
<td>5.92</td>
<td>18.14</td>
<td>34.08</td>
</tr>
<tr>
<td>&lt;Step 1 5&gt;</td>
<td>1.00</td>
<td>2.88</td>
<td>3.36</td>
<td>9.24</td>
<td>15.82</td>
<td>31.85</td>
</tr>
<tr>
<td>&lt;Step 1 6&gt;</td>
<td>1.00</td>
<td>1.94</td>
<td>8.94</td>
<td>7.03</td>
<td>11.91</td>
<td>21.41</td>
</tr>
<tr>
<td>General &lt;Txn&gt;</td>
<td>1.00</td>
<td>3.26</td>
<td>3.59</td>
<td>6.07</td>
<td>16.26</td>
<td>61.63</td>
</tr>
<tr>
<td>Get Acceptance</td>
<td>1.00</td>
<td>1.87</td>
<td>2.13</td>
<td>2.82</td>
<td>5.01</td>
<td>7.41</td>
</tr>
<tr>
<td>Launch &lt;Application name&gt;</td>
<td>1.00</td>
<td>2.18</td>
<td>2.46</td>
<td>14.45</td>
<td>16.55</td>
<td>28.14</td>
</tr>
<tr>
<td>&lt;Step 2 1&gt;</td>
<td>1.00</td>
<td>3.74</td>
<td>3.80</td>
<td>5.45</td>
<td>12.16</td>
<td>18.80</td>
</tr>
<tr>
<td>&lt;Step 2 2&gt;</td>
<td>1.00</td>
<td>2.64</td>
<td>2.31</td>
<td>12.64</td>
<td>13.36</td>
<td>32.96</td>
</tr>
<tr>
<td>&lt;Step 2 3&gt;</td>
<td>1.00</td>
<td>2.15</td>
<td>3.20</td>
<td>3.97</td>
<td>7.39</td>
<td>25.58</td>
</tr>
<tr>
<td>Login Page</td>
<td>1.00</td>
<td>1.50</td>
<td>1.56</td>
<td>1.88</td>
<td>1.75</td>
<td>2.69</td>
</tr>
<tr>
<td>Logoff</td>
<td>1.00</td>
<td>1.31</td>
<td>1.31</td>
<td>1.77</td>
<td>1.69</td>
<td>2.00</td>
</tr>
<tr>
<td>&lt;Txn&gt; Period</td>
<td>1.00</td>
<td>2.22</td>
<td>4.23</td>
<td>4.78</td>
<td>8.64</td>
<td>17.51</td>
</tr>
<tr>
<td>&lt;Txn&gt; Summary</td>
<td>1.00</td>
<td>2.27</td>
<td>2.16</td>
<td>4.30</td>
<td>7.35</td>
<td>19.05</td>
</tr>
<tr>
<td>&lt;Txn&gt; Tab</td>
<td>1.00</td>
<td>9.27</td>
<td>16.36</td>
<td>33.72</td>
<td>30.27</td>
<td>73.27</td>
</tr>
<tr>
<td>Select Account</td>
<td>1.00</td>
<td>3.80</td>
<td>9.40</td>
<td>19.39</td>
<td>42.80</td>
<td>73.90</td>
</tr>
<tr>
<td>Select Business</td>
<td>1.00</td>
<td>1.62</td>
<td>2.21</td>
<td>3.05</td>
<td>8.47</td>
<td>13.85</td>
</tr>
<tr>
<td>Select New Business</td>
<td>1.00</td>
<td>3.21</td>
<td>3.93</td>
<td>3.29</td>
<td>3.00</td>
<td>3.14</td>
</tr>
<tr>
<td>Select Suburb</td>
<td>1.00</td>
<td>3.93</td>
<td>4.57</td>
<td>15.29</td>
<td>33.93</td>
<td>51.79</td>
</tr>
<tr>
<td>Stagger Start</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Submit Business Search</td>
<td>1.00</td>
<td>1.21</td>
<td>1.70</td>
<td>2.23</td>
<td>11.84</td>
<td>23.00</td>
</tr>
<tr>
<td>Submit Login</td>
<td>1.00</td>
<td>1.30</td>
<td>1.35</td>
<td>1.40</td>
<td>1.65</td>
<td>1.65</td>
</tr>
<tr>
<td>&lt;Txn&gt; OK</td>
<td>1.00</td>
<td>5.40</td>
<td>17.39</td>
<td>59.80</td>
<td>70.59</td>
<td>125.40</td>
</tr>
<tr>
<td>&lt;Txn&gt; Questions</td>
<td>1.00</td>
<td>3.67</td>
<td>5.33</td>
<td>6.67</td>
<td>20.44</td>
<td>45.44</td>
</tr>
<tr>
<td><strong>GEO MEAN</strong></td>
<td><strong>1.00</strong></td>
<td><strong>2.54</strong></td>
<td><strong>3.52</strong></td>
<td><strong>6.09</strong></td>
<td><strong>10.68</strong></td>
<td><strong>21.41</strong></td>
</tr>
</tbody>
</table>
### Absolute Performance Comparison

<table>
<thead>
<tr>
<th>Step</th>
<th>1 User</th>
<th>10 Users</th>
<th>20 Users</th>
<th>30 Users</th>
<th>40 Users</th>
<th>50 Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Search</td>
<td>0.08</td>
<td>0.69</td>
<td>0.26</td>
<td>0.43</td>
<td>1.77</td>
<td>5.35</td>
</tr>
<tr>
<td>Close Now</td>
<td>0.47</td>
<td>1.24</td>
<td>1.43</td>
<td>4.01</td>
<td>5.75</td>
<td>8.10</td>
</tr>
<tr>
<td>Complete Tab</td>
<td>0.42</td>
<td>0.77</td>
<td>1.40</td>
<td>2.54</td>
<td>3.85</td>
<td>20.81</td>
</tr>
<tr>
<td>&lt;Step 3 1&gt;</td>
<td>1.07</td>
<td>2.67</td>
<td>3.85</td>
<td>11.15</td>
<td>13.42</td>
<td>26.03</td>
</tr>
<tr>
<td>&lt;Step 3 2&gt;</td>
<td>0.46</td>
<td>1.05</td>
<td>1.39</td>
<td>2.71</td>
<td>4.78</td>
<td>19.45</td>
</tr>
<tr>
<td>&lt;Step 3 3&gt;</td>
<td>0.26</td>
<td>0.43</td>
<td>0.82</td>
<td>2.18</td>
<td>3.15</td>
<td>15.70</td>
</tr>
<tr>
<td>&lt;Step 3 4&gt;</td>
<td>0.21</td>
<td>0.60</td>
<td>1.45</td>
<td>2.46</td>
<td>4.00</td>
<td>6.01</td>
</tr>
<tr>
<td>&lt;Step 3 5&gt;</td>
<td>0.21</td>
<td>0.56</td>
<td>0.73</td>
<td>2.09</td>
<td>4.08</td>
<td>5.86</td>
</tr>
<tr>
<td>End / Save</td>
<td>7.12</td>
<td>9.05</td>
<td>10.19</td>
<td>16.61</td>
<td>32.11</td>
<td>58.56</td>
</tr>
<tr>
<td>&lt;Step 1 1&gt;</td>
<td>1.34</td>
<td>6.69</td>
<td>8.01</td>
<td>11.46</td>
<td>23.25</td>
<td>62.73</td>
</tr>
<tr>
<td>&lt;Step 1 2&gt;</td>
<td>0.43</td>
<td>0.95</td>
<td>1.91</td>
<td>3.16</td>
<td>7.02</td>
<td>15.31</td>
</tr>
<tr>
<td>&lt;Step 1 3&gt;</td>
<td>0.37</td>
<td>0.90</td>
<td>3.47</td>
<td>1.87</td>
<td>5.99</td>
<td>15.80</td>
</tr>
<tr>
<td>&lt;Step 1 4&gt;</td>
<td>0.49</td>
<td>2.00</td>
<td>1.82</td>
<td>2.90</td>
<td>8.89</td>
<td>16.70</td>
</tr>
<tr>
<td>&lt;Step 1 5&gt;</td>
<td>0.33</td>
<td>0.95</td>
<td>1.11</td>
<td>3.05</td>
<td>5.22</td>
<td>10.51</td>
</tr>
<tr>
<td>&lt;Step 1 6&gt;</td>
<td>0.34</td>
<td>0.66</td>
<td>3.04</td>
<td>2.39</td>
<td>4.05</td>
<td>7.28</td>
</tr>
<tr>
<td>General &lt;Txn&gt;</td>
<td>0.46</td>
<td>1.50</td>
<td>1.65</td>
<td>2.79</td>
<td>7.48</td>
<td>28.35</td>
</tr>
<tr>
<td>Get Acceptance</td>
<td>11.82</td>
<td>22.16</td>
<td>25.20</td>
<td>33.29</td>
<td>59.27</td>
<td>87.57</td>
</tr>
<tr>
<td>Launch &lt;Application name&gt;</td>
<td>0.74</td>
<td>1.61</td>
<td>1.82</td>
<td>10.69</td>
<td>12.25</td>
<td>20.82</td>
</tr>
<tr>
<td>&lt;Step 2 1&gt;</td>
<td>1.58</td>
<td>5.91</td>
<td>6.00</td>
<td>8.61</td>
<td>19.21</td>
<td>29.72</td>
</tr>
<tr>
<td>&lt;Step 2 2&gt;</td>
<td>0.28</td>
<td>0.74</td>
<td>0.65</td>
<td>3.54</td>
<td>3.74</td>
<td>9.23</td>
</tr>
<tr>
<td>&lt;Step 2 3&gt;</td>
<td>0.71</td>
<td>1.53</td>
<td>2.27</td>
<td>2.82</td>
<td>5.25</td>
<td>18.16</td>
</tr>
<tr>
<td>Login Page</td>
<td>0.16</td>
<td>0.24</td>
<td>0.25</td>
<td>0.30</td>
<td>0.28</td>
<td>0.43</td>
</tr>
<tr>
<td>Logoff</td>
<td>0.13</td>
<td>0.17</td>
<td>0.17</td>
<td>0.23</td>
<td>0.22</td>
<td>0.26</td>
</tr>
<tr>
<td>&lt;Txn&gt; Period</td>
<td>1.20</td>
<td>2.66</td>
<td>5.08</td>
<td>5.73</td>
<td>10.37</td>
<td>21.01</td>
</tr>
<tr>
<td>&lt;Txn&gt; Summary</td>
<td>1.52</td>
<td>3.45</td>
<td>3.29</td>
<td>6.55</td>
<td>11.17</td>
<td>28.96</td>
</tr>
<tr>
<td>&lt;Txn&gt; Tab</td>
<td>0.11</td>
<td>1.02</td>
<td>1.80</td>
<td>3.71</td>
<td>3.33</td>
<td>8.06</td>
</tr>
<tr>
<td>Select Account</td>
<td>0.10</td>
<td>0.38</td>
<td>0.94</td>
<td>1.94</td>
<td>4.28</td>
<td>7.39</td>
</tr>
<tr>
<td>Select Business</td>
<td>1.18</td>
<td>1.91</td>
<td>2.61</td>
<td>3.60</td>
<td>10.00</td>
<td>16.34</td>
</tr>
<tr>
<td>Select New Business</td>
<td>0.14</td>
<td>0.45</td>
<td>0.55</td>
<td>0.46</td>
<td>0.42</td>
<td>0.44</td>
</tr>
<tr>
<td>Select Suburb</td>
<td>0.14</td>
<td>0.55</td>
<td>0.64</td>
<td>2.14</td>
<td>4.75</td>
<td>7.25</td>
</tr>
<tr>
<td>Stagger Start</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Submit Business Search</td>
<td>0.43</td>
<td>0.52</td>
<td>0.73</td>
<td>0.96</td>
<td>5.09</td>
<td>9.89</td>
</tr>
<tr>
<td>Submit Login</td>
<td>0.20</td>
<td>0.26</td>
<td>0.27</td>
<td>0.28</td>
<td>0.33</td>
<td>0.33</td>
</tr>
<tr>
<td>&lt;Txn&gt; OK</td>
<td>0.05</td>
<td>0.27</td>
<td>0.87</td>
<td>2.99</td>
<td>3.53</td>
<td>6.27</td>
</tr>
<tr>
<td>&lt;Txn&gt; Questions</td>
<td>0.09</td>
<td>0.33</td>
<td>0.48</td>
<td>0.60</td>
<td>1.84</td>
<td>4.09</td>
</tr>
<tr>
<td><strong>SUM</strong></td>
<td><strong>34.64</strong></td>
<td><strong>74.88</strong></td>
<td><strong>96.15</strong></td>
<td><strong>160.44</strong></td>
<td><strong>290.14</strong></td>
<td><strong>598.77</strong></td>
</tr>
</tbody>
</table>
6.3 Detailed Performance Results for Individual Changes

This section shows comparative results from TestStudio showing the impact of each of the changes identified in section 3.1. Generally the comparisons show the specific “before and after” impact of the individual change. Changes were applied cumulatively, so the “after” results for one graph generally correspond to the “before” data for the next – although due to the amount of change completed in a very compressed timeframe, it is possible that some other minor environmental changes may have been included in some of the results shown, and the order in which results are presented may not exactly match the order of events.

Note that a few anomalies are evident in some of the results presented. It is likely that most of these were caused by factors outside of the controlled test environment (for example the shared Oracle database and the shared MQ interface to the test mainframe environment for <transaction>).

Some of the changes affected both single-user and multi-user performance (for example adding a missing database index), whilst others are only evident in a multi-user test (for example contention for the Start<Application component> servlet). The results are presented as comparisons of single-user runs where possible, or 20-user runs if these are necessary to show the impact.

The comparisons are shown as “relative” and “absolute” charts of the average response times recorded for each of the timing points defined in the workload model in section 6.1. The “geometric mean” of the ratios of the response times for the relative report and the total “system” time taken by all timing points for a single transaction, for the absolute reports, are also shown.
6.3.1 Missing Index on Post Code Table

The graphs show a comparison of single-user response times, before and after adding the index on the pcode1 column of the Post Code table.

Relative Performance Comparison

<table>
<thead>
<tr>
<th>Ratio</th>
<th>Average Response Time</th>
<th>Absolute Performance Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.65</td>
<td></td>
<td>Total Response Time (secs)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Before: 85.74</td>
</tr>
</tbody>
</table>
6.3.2 Impact of Logging

The graphs show a comparison of single-user response times, before and after reducing the logging level from DEBUG to ERROR throughout the application. Note that separate tests showed that the bulk of the improvement came from the reduction in logging within <Application component>Server. The “baseline” for this comparison shows the performance after the creation of the Post Code index.

Relative Performance Comparison

![Graph showing relative performance comparison between Perf 1 user baseline and Perf 10 no logging.]

**Ratio**

Average Response Time 0.85

Absolute Performance Comparison

![Graph showing absolute performance comparison between Total Response Time (secs) before and after logging level reduction.]

**Total Response Time (secs)**

Before 48.00  After 44.83
6.3.3 MQ Polling Speed in Printing

Specific test results showing the impact of the polling speed on the time required to print are not available. Observations during testing indicated that this reduced the time taken for the Get Acceptance transaction step by around 2 to 3 seconds per document selected for printing. This change will be superseded by the proposed change in printing architecture, described next.

6.3.4 Printing Architecture

The graphs show a comparison of single-user response times, between a run for which one document was selected for printing and a run with no documents selected. Not selecting a document for printing is almost equivalent to the proposed change in the printing architecture, since the <Application name> application completes nearly all of the work involved in requesting a print job anyway. Not selecting any prints avoids the need to wait for the response, which will also be avoided by the proposed architectural change.

Relative Performance Comparison

<table>
<thead>
<tr>
<th>Ratio</th>
<th>Average Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.95</td>
<td></td>
</tr>
</tbody>
</table>

Absolute Performance Comparison

<table>
<thead>
<tr>
<th>Total Response Time (secs)</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>39.35</td>
<td>30.98</td>
</tr>
</tbody>
</table>
6.3.5 Repeated Loading of Quote

Specific test results showing the impact of eliminating an unnecessary loading of the <transaction> during acceptance processing are not available. Observations during testing indicated that this reduced the time taken for the Get Acceptance transaction step by around 2 seconds per transaction.

6.3.6 Impact of Explicit Garbage Collection

The graphs show a comparison of 20-user response times, before and after disabling explicit Garbage Collection. Note that the impact of this option would have been more pronounced if the comparison had been done after the elimination of some of the other application issues affecting multi-user performance.

Relative Performance Comparison

Ratio

Average Response Time 0.57

Absolute Performance Comparison

Total Response Time (secs) Before 130.49 After 83.85
6.3.7 JVM Server Option

The graphs show a comparison of 20-user response times, before and after enabling the “-server” JVM option. Note that the impact of this option would have been more pronounced if the comparison had been done after the elimination of some of the other application issues affecting multi-user performance.

Relative Performance Comparison

![Graph showing relative performance comparison]

**Ratio**

**Average Response Time**

0.77

Absolute Performance Comparison

![Graph showing absolute performance comparison]

**Before**

**After**

**Total Response Time (secs)**

83.85 73.86
6.3.8 Start<Application component> Serialisation

The graphs show a comparison of 20-user response times, before and after the Start<Application component> serialisation issue had been resolved.

Notice that the average response time for most transaction steps actually increased with this change. This is simply because removing the bottleneck in the Start<Application component> component allowed multiple Get Acceptance transactions to occur simultaneously. This had the effect of significantly reducing the elapsed time to complete a transaction, hence increasing the throughput of the system. As the test scripts were not set up to maintain a constant transaction rate, but a constant (average) user “think time” after receipt of each response from the server, this had the effect of increasing the overall transaction rate and hence the load on the servers.

Relative Performance Comparison

<table>
<thead>
<tr>
<th>Ratio</th>
<th>Get Acceptance</th>
<th>0.66</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Launch &lt;Application name&gt;</td>
<td>0.26</td>
</tr>
<tr>
<td>Average Response Time</td>
<td>1.12</td>
<td></td>
</tr>
</tbody>
</table>

Absolute Performance Comparison

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Acceptance</td>
<td>33.03</td>
</tr>
<tr>
<td>Launch &lt;Application name&gt;</td>
<td>6.40</td>
</tr>
<tr>
<td>Total Response Time (secs)</td>
<td>73.86</td>
</tr>
</tbody>
</table>
6.3.9 Impact of JVM Version

The graphs show a comparison of 20-user response times, for the 1.3.1_09 and 1.4.2 JVM versions. Both tests have explicit Garbage Collection disabled and the “-server” option selected.

The relative response report shows that on average there is no difference between the two versions, whilst the absolute report shows a small reduction in overall response times with 1.4.2. However the sample size is such that these differences may be largely due to random variations, and several anomalies were noted in the <transaction> times recorded during the 1.3.1_09 test run, probably due to external (test environment) factors.

Note that little difference was expected between 1.3.1_09 and 1.4.2 with the options selected. Based on testing in other environments, it is likely that more substantial gains could be achieved using other features introduced in 1.4.2, however, as discussed in section 4.1, there was not time to explore these options during the course of this exercise.

Relative Performance Comparison

<table>
<thead>
<tr>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Response Time</td>
</tr>
</tbody>
</table>

Absolute Performance Comparison

<table>
<thead>
<tr>
<th>Total Response Time (secs)</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>56.96</td>
<td>53.43</td>
</tr>
</tbody>
</table>
6.4 Details of JVM Configuration Changes

The exact changes needed to implement the server JVM and disable explicit garbage collection are shown below. Note that the highlighted differences reflect the changes made in the test environment; the -Xmx and -Xms settings shown (1600m) are understood to be in production already.

Change:

```
SET JRE_PATH=D:\win32app\jdk1.3.1_09\jre\bin\hotspot
```
To:

```
SET JRE_PATH=D:\win32app\jdk1.3.1_09\jre\bin\server
```

Change:

```
JavaService.exe -install "<Customer application> AppServer"
%JRE_PATH%\jvm.dll -Xmx512m -Xms128m -
Djava.class.path=%CLASS_PATH% -
Djavax.xml.transform.TransformerFactory=org.apache.xalan.xslt.trax.TransformerFactoryImpl -start %MAIN_CLASS% -params
%CFG_PATH% -out %HOME%\stdout.log -err %HOME%\stderr.log -
current %HOME%
```
To:

```
JavaService.exe -install "<Customer application> AppServer"
%JRE_PATH%\jvm.dll -XX:+DisableExplicitGC -Xmx1600m -Xms1600m -
Djava.class.path=%CLASS_PATH% -
Djavax.xml.transform.TransformerFactory=org.apache.xalan.xslt.trax.TransformerFactoryImpl -start %MAIN_CLASS% -params
%CFG_PATH% -out %HOME%\stdout.log -err %HOME%\stderr.log -
current %HOME%
```
6.5 Possible Issues Requiring Further Investigation

6.5.1 Zero Impact Queries

Details of the 3 most expensive queries observed in a half hour sample of production SQL transactions are shown below, with brief comments on possible approaches to tuning. However, it should be noted that there are only relatively small gains to be made in these areas, compared with the major gain from the index on the Post Code table, which had already been implemented in production by the time this sample was collected.

<table>
<thead>
<tr>
<th>#</th>
<th>Count</th>
<th>Average Response</th>
<th>Total Response</th>
<th>Average Rows</th>
<th>Total Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>627</td>
<td>0.096</td>
<td>59.9</td>
<td>1</td>
<td>627</td>
</tr>
<tr>
<td>2</td>
<td>46</td>
<td>0.302</td>
<td>13.9</td>
<td>11</td>
<td>514</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>0.508</td>
<td>10.7</td>
<td>54</td>
<td>1140</td>
</tr>
</tbody>
</table>

Query #1

\[
\text{insert into detail_solution (id_solution, name_parameter, no_sequence, value_parameter, no_bucket) values ()}
\]

A large number of individual rows are inserted consecutively in each transaction (two sets of the order of 80 rows each). There is possible scope for the use of array inserts to minimise the network latency and other overheads associated with these operations. No investigation has been done of the work involved in such a change – in particular, it is not known whether the JDBC classes in use support array inserts.

A brief inspection of the SQL showed that the application appears to insert 80 odd rows (mainly in detail_solution, but also in other tables), before immediately deleting and reinserting (apparently) equivalent rows. It is worth investigating whether there is any scope to eliminate this behaviour.

Query #2

\[
\text{select id_solution, name_parameter, value_parameter, no_sequence, last_update_date, no_bucket, reserve1, reserve2 from detail_solution where id_solution='\text{\textless \text{param}\textgreater}' order by name_parameter asc,no_sequence asc,no_bucket asc}
\]

The average response of 0.3 seconds seems surprisingly high considering the average number of rows returned and the presence in the WHERE clause of a
column that is likely to be indexed. As this is a large table it is worth reviewing the indexes and physical table structure in the database.

**Query #3**

```sql
SELECT ANZSICID || '|' || TRIM(LT01E.OCCUPATION) || '
'|| OCCUPATIONDESCRIPTOR || '|' || OCCUPATIONAUTOFLAG
FROM LT01E
WHERE UPPER(LT01E.OCCUPATION) LIKE UPPER('%'||:1||'%')
AND TO_DATE(SUBSTR(:2,1,10),'MM/DD/YYYY')
between ValidFromDate and ValidToDate
order by LT01E.OCCUPATION
```

Bind variables: 1 'BANNER' 2 '11-30-2003 00:00:00'

It appears that this query may not be able to use an index due to the `UPPER()` function. There is possible scope for storing `UPPER(OCCUPATION)` redundantly and defining an index on the uppercased column, or possibly defining a functional index on `UPPER(OCCUPATION)` (although with the plan to migrate <Application name> to DB2, it would be necessary to confirm whether DB2 supports functional indexes).
6.5.2 XSLT Re-compilation

XSLT compilation, required for the XML transformation processing involved in transaction acceptance, is a resource hungry process.

From discussions with the <Application name> project team, it is understood that the design of the transaction acceptance processing includes a mechanism to allow XSLT compilation results to be cached, to avoid the overhead of XSLT compilation for every XML transformation. From a brief analysis of stack traces it is clear that the code refers to an XSLT cache, however there is some doubt as to whether repeated XSLT compilation is actually being avoided, as intended. This observation was only discovered during analysis after the onsite tuning exercise had been completed; so it has not been possible to investigate further to confirm whether or not this issue is real.

The frequency with which calls to the XSLT compiler appear in HPROF output, and the fact that XSLT compilation is seen during a stack dump taken towards the end of a 5-user test run (when the result should have been compiled and cached even before sampling began), suggests that the caching of the compilation results may not be working as intended. If this is indeed an issue, the two possibilities appear to be: a) it simply isn't caching, or b) it is cached on a per-thread basis, and the samples analysed happen to have picked up the first calls for a particular thread.

It is worth confirming this behaviour, as there could be a significant CPU reduction available if the first interpretation is right (i.e. it is not caching at all).

The stack dump showing the compilation occurring towards the end of the 5-user run is shown below.

Thread dump (start 1068784298671ms, stop 1068784693546ms, duration 394875ms)
@1068784603265ms
...
Thread-1:
[1] java.lang.String.replace (String.java:1672)
[2] org.apache.bcel.generic.ObjectType.<init> (ObjectType.java:73)
[3] org.apache.bcel.generic.Type.getType (Type.java:184)
[4] org.apache.bcel.generic.Type.getReturnType (Type.java:198)
[5] org.apache.bcel.generic.InvokeInstruction.getReturnType
(InvokeInstruction.java:118)
(InvokeInstruction.java:120)
(Mode.java:1047)
(Stylesheet.java:449)
(Stylesheet.java:548)
[12] org.apache.xalan.xsltc.compiler.XSLTC.compile
(XSLTC.java:359)
(XSLTC.java:434)
(TransformerFactoryImpl.java:623)
[15] <customer.app>.XSLHelper.getTransformerFromCachedTemplate
()
[16] <customer.app>.XSLHelper.performXSLTransformation ()
[17] <customer.app>.XSLHelper.performXSLTransformation ()
[18] <supplier.app>.applicationbeans.XMLBean.getXML ()
[19] <supplier.app>.applicationbeans.<customer>.PrototypeBean.getXMLString ()
[21] <supplier.app>.applicationbeans.<customer>.PrototypeBean.save ()
[22] <supplier.app>.STLDBplugin.SessionContext.saveModel ()
[23] <supplier.app>.STLDBplugin.SessionContext.save ()
[24] <supplier.app>.STLDBplugin.STLDBplugin.onUserFunction ()
[26] <package>.commands.Cx<package>PluginFunctionCommand.execute (Cx<package>PluginFunctionCommand.java:374)
[27] <package>.commands.Cx<package>Command.execute (Cx<package>Command.java:678)
[28] <package>.framework.CxCommandExecutor.execute (CxCommandExecutor.java:338)
[29] <package>.framework.CxCommandExecutorRunnable.run (CxCommandExecutorRunnable.java:282)

@1068784606250ms:
Thread-1: 
[16] <customer.app>.XSLHelper.getTransformerFromCachedTemplate ()
[17] <customer.app>.XSLHelper.performXSLTransformation ()
[18] <customer.app>.XSLHelper.performXSLTransformation ()
[19] <customer.app>.XMLTransform.transformXML ()
[22] <supplier.app>.applicationbeans.<customer>.PrototypeBean.save ()
[23] <supplier.app>.STLDBplugin.SessionContext.saveModel ()
[25] <supplier.app>.STLDBplugin.STLDBplugin.onUserFunction ()
[27] <package>.commands.Cx<package>PluginFunctionCommand.execute (Cx<package>PluginFunctionCommand.java:374)
[28] <package>.commands.Cx<package>Command.execute (Cx<package>Command.java:678)
[29] <package>.framework.CxCommandExecutor.execute (CxCommandExecutor.java:338)
[31] java.lang.Thread.run (Thread.java:479)
6.5.3 Scalability of <transaction>

As can be seen from the results of the 50 user run presented in section 3.2.2 and appendix 6.2, the performance of the <transaction> steps deteriorated quite significantly under multi-user load. It is quite likely that this is a reflection of the shared infrastructure (MQ and mainframe) used for <transaction> in the test environment.

However, as the architecture has a single set of components responsible for <transaction>, it is worth confirming the throughput of these components in the production configuration, as this could potentially become another bottleneck.

6.5.4 Further Reduction in Logging

Even with the Log4J logging level set to ERROR, large volumes of log files were still generated by the application. A three-hour run, increasing from 5 to 50 users generated over 200 MB of log files.

It is worth investigating whether the level of this type of logging can be reduced.

6.5.5 Thread Configuration and Missing Threads

Analysis of the stack traces captured via the Java debugging interface on the <Application component>Server JVM during a 5-user test run gave rise to the following observations.

1) There are gaps in the sequence numbers of the java threads:

   Thread-17
   Thread-15
   Thread-14
   Thread-13
   Thread-11
   Thread-10
   Thread-9
   Thread-8
   Thread-7
   Thread-6
   Thread-5
   Thread-4
   Thread-3
   Thread-2
   Thread-1
   Thread-0

2) Late in the sample, a <Package> thread named Thread-18 appears, executing com.<Package>.server.framework.CxClientServiceReader.run (listening on a socket). It is not clear what this thread is or why it only appeared at this stage.

There was no opportunity to investigate further, however the following possibilities have been identified.

- This may be perfectly normal <Package> thread-pool behaviour (although it is not consistent with our previous observations of <Package> behaviour in other sites).
• The missing threads may possibly have crashed.
• The fact that new threads are starting during the run might be an indication that the thread pool is not configured correctly.
• Why isn’t the thread pool configured to start with more threads?

6.5.6 Intermittent <transaction> Error

An intermittent error was observed very occasionally during several of the multi-user tests. On displaying the <Transaction> Summary screen a <transaction> shows as "yet to be processed", however by the time the script gets to the acceptance stage, the <transaction> is saved and accepted successfully, and looking up the <transaction> details after the test completes, the <transaction> is processed correctly. This appears to be some sort of race condition being exposed under stress. Details of the exact symptoms have been supplied separately to the <Application name> team. This appears to be a minor issue, but is worth noting as intermittent, stress related issues such as this can sometimes be a symptom of a more significant issue.

6.5.7 Exceptions in logs

Analysis of java.lang.Exception.<init> by callee indicates that a number of exceptions occurred during the test run monitored with HPROF.

The number of exceptions reported in a comparatively short test is surprising. Exception handling and associated error logging may be contributing significantly to the measured response times.

The following two exceptions were related to <Customer> package code.

• java.io.FileNotFoundException: 
  <customer.app>.<System>InteractionPlugin.hasSessionTimedOut

• java.lang.RuntimeException: 
  <customer.app>.<Package>Interaction.isNumeric

Note - as this analysis is based on sampling, it does not represent all exceptions that may have occurred.

Analysis of the <Application component> server.log file (for this short test run) shows 584 lines containing the word "Exception".

HPROF callee graph for java.lang.Throwable.<init>:

```
1: java.lang.Throwable.<init>: 0.07% (5 inclusive)
2: java.lang.Exception.<init>: 0.07% (5 inclusive)
3: java.io.IOException.<init>: 0.04% (3 inclusive)
4: java.io.InterruptedIOException.<init>: 0.03% (2 inclusive)
5: java.net.PlainSocketImpl.socketAccept: 0.03% (2 inclusive)
6: java.net.PlainSocketImpl.accept: 0.03% (2 inclusive)
7: java.net.ServerSocket.implAccept: 0.03% (2 inclusive)
8: java.net.ServerSocket.accept: 0.03% (2 inclusive)
9: <package>.framework.CxSocketRequestProcessor.run: 0.03% (2 inclusive)
10: java.lang.Thread.run: 0.03% (2 inclusive)
```

21 December 2003 <Application> Performance Tuning Page 42
inclusive)
11: <package>.configmgr.CxFileConfigurationMgr.processImportFile: 0.01% (1 inclusive)
12: <package>.configmgr.CxFileConfigurationMgr.readConfigFile: 0.01% (1 inclusive)
13: <package>.configmgr.CxFileConfigurationMgr.<init>: 0.01% (1 inclusive)
14: <customer.app>.<System>InteractionPlugin.hasSessionTimedOut: 0.01% (1 inclusive)
15: <customer.app>.<System>InteractionPlugin.onGarbageCollect: 0.01% (1 inclusive)
16: <package>.CxRegistry.removeSession: 0.01% (1 inclusive)
17: <package>.CxGarbageCollector.run: 0.01% (1 inclusive)
18: java.lang.Thread.run: 0.01% (1 inclusive)
3: java.lang.ClassNotFoundException.<init>: 0.01% (1 inclusive)
4: java.net.URLClassLoader$1.run: 0.01% (1 inclusive)
5: java.security.AccessController.doPrivileged: 0.01% (1 inclusive)
6: java.net.URLClassLoader.findClass: 0.01% (1 inclusive)
7: java.lang.ClassLoader.loadClass: 0.01% (1 inclusive)
8: java.lang.ClassLoader.loadClass: 0.01% (1 inclusive)
9: sun.misc.LauncherAppClassLoader.loadClass: 0.01% (1 inclusive)
10: java.lang.ClassLoader.loadClass: 0.01% (1 inclusive)
11: java.util.ResourceBundle.loadBundle: 0.01% (1 inclusive)
12: java.util.ResourceBundle.findBundle: 0.01% (1 inclusive)
13: java.util.ResourceBundle.getBundleImpl: 0.01% (1 inclusive)
14: java.util.ResourceBundle.getBundle: 0.01% (1 inclusive)
15: java.text.DateFormatSymbols.getZoneData: 0.01% (1 inclusive)
16: java.text.DateFormatSymbols.cacheLookup: 0.01% (1 inclusive)
17: java.text.DateFormatSymbols.initializeData: 0.01% (1 inclusive)
18: java.text.SimpleDateFormat.<init>: 0.01% (1 inclusive)
20: java.util.Date.toString: 0.01% (1 inclusive)
21: <package>.logging.CxStdFileDiagnosticLogger.formatMessage: 0.01% (1 inclusive)
22: <package>.logging.CxStdFileDiagnosticLogger.logInformation: 0.01% (1 inclusive)
23: <package>.CxGarbageCollector.run: 0.01% (1 inclusive)
3: java.lang.RuntimeException.<init>: 0.01% (1 inclusive)
4: java.lang.NullPointerException.<init>: 0.01% (1 inclusive)
5: java.lang.Float.readJavaFormatString: 0.01% (1 inclusive)
6: java.lang.Double.parseDouble: 0.01% (1 inclusive)
7: <customer.app>.<Package>Interaction.isNumeric: 0.01% (1 inclusive)
8: <customer.app>.<Package>Interaction.setParameterValue: 0.01% (1 inclusive)
9: <customer.app>.<Package>Interaction.setParameterValue: 0.01% (1 inclusive)
10: <customer.app>.<Package>Interaction.onUserTag: 0.01% (1 inclusive)
11: <package>.gen2tags.CxPluginTag.executeFunction: 0.01% (1 inclusive)
12: <package>.gen2tags.CxPluginTag.fieldLookup: 0.01% (1 inclusive)
13: <package>.gen2tags.CxTagUtils.resolveTagCast: 0.01% (1 inclusive)
14: <package>.gen2tags.fieldparser.CxFPCastElement.resolve: 0.01% (1 inclusive)
15: <package>.gen2tags.CxServerTag.getAttributeValue: 0.01% (1 inclusive)
16: <package>.gen2tags.CxOpVarTag.expandTag: 0.01% (1 inclusive)
17: <package>.gen2tags.CxServerTag.expandTagChildren: 0.01% (1 inclusive)
18: <package>.gen2tags.CxPluginTag.expandTag: 0.01% (1 inclusive)
19: <package>.gen2tags.CxServerTag.expandTagChildren: 0.01% (1 inclusive)
20: <package>.gen2tags.CxSubTag.expandSubTag: 0.01% (1 inclusive)
21: <package>.gen2tags.CxCallSubTag.expandTag: 0.01% (1 inclusive)
22: <package>.gen2tags.CxServerTag.expandTagChildren: 0.01% (1 inclusive)
23: <package>.gen2tags.CxGen2RootTag.expandTag: 0.01% (1 inclusive)
6.6 Other Observations

6.6.1 TE and <External user> System Issues

Whilst the <External user system> Executive and the <External user> system used as part of the test environment are outside the scope of this exercise, and outside <Customer>’s responsibility in production, it is worth noting that some of the failure modes that occurred during the tests gave rise to knock-on problems in these environments. Specifically, two separate database connection pools, one associated with logon processing and one with the <end user> system ran out of pooled connections.

It is likely that these failures were the result of the test scripts “blindly continuing”, even after a failure that would deter a real user from carrying on, so these issues probably would not manifest themselves in the real world.