Equinox ECProxy(EC02)

Version 1.0.0

Summary

|  |  |
| --- | --- |
| Document Reference Number |  |
| Issue Number | 1.0.0 |
| Date | January, 2013 |

Copyright and Trademarks

Copyright

No part of this manual may be copied, photocopied, reproduced, translated or reduced to any medium or machine readable form without prior written consent from AIS PLC, or unless permitted to do so by the terms of the license granted by AIS PLC.

The information in this document is subject to change without notice.

Trademark Notices

Any product names referred to in this document are the property of the respective holders of the trademark or service mark and their rights are acknowledged.

UNIX® is a registered trademark of The Open Group.

Licensed Software

Portions of this solution are provided under license from Nokia Siemens Networks Limited.

Warranty

Although AIS uses all reasonable efforts to ensure the accuracy and completeness of this document, AIS makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. AIS shall not be held liable for errors contained within this manual or direct, indirect or consequential damage as a result of using this manual.

Please refer to your contract with AIS for specific warranty terms for your product.

Copyright © Advance Info Service Public Limited Company, TH, 2009

All rights reserved.

AIS PLC  
414 Phaholyothin Rd,  
Samsen Nai,  
Phayathai,  
Bangkok 10400.  
Thailand.

Document Control

Version History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Description | Author | Date |
| 1.0.0 | Initial draft | Thitipong Jampajeen | January 2013 |
|  |  |  |  |

Document Conventions

The following typefaces are used throughout this guide:

* The ‘Courier’ typeface is used for directory objects and attributes, file names and command line code.
* ‘Italics’ are used for emphasis and for cross references.
* This bold typeface is used to represent information you should type in at the keyboard.

|  |  |
| --- | --- |
| aperto-b | Note: This Note is used to illustrate that you should pay particular attention to its accompanying information. |

References

[1] None

Table of Contents

[Chapter1: Introduction 5](#_Toc332373289)

[Chapter2: Requirement Specification & Design 5](#_Toc332373290)

[Chapter3: Configuration 9](#_Toc332373291)

[Chapter4: Performance Test Result 11](#_Toc332373292)

[Chapter5: Summary 11](#_Toc332373293)

## Chapter1: Introduction

EC02 is an software interface between Application layer and Equinox layer (ES00) , old version was built by integrating Jetty web server that has many problems occur such as high latency , high memory and CPU usage which affect Transaction per second(TPS) and hard to config.

ECProxy is made for solving these problem, concepts are low latency, less resource consumption server.

Requirement Summary

1. Low latency & High throughput
2. Low machine resource consumption.
3. Easy Configuration

## Chapter2: Requirement Specification and Design

Refer to requirements in previous chapter, we research and develop ECProxy that have specification following

1. Develop by using C/C++ & Java language.
2. Use C/C++ to control and manage low level IO such as SOCKET, FILE IO in ECProxy layer.
3. Use Java to communicate between EC02 layer and AF.
4. Use Java Native Interface (JNI) to communicate between Java and C/C++
5. Use Thread Pool Model and POSIX Thread capability together to dispatch Jobs to other affinity CPU.
6. Change methodology for starting application.
7. Change XML parser in Java part from DOM to sTAX.
8. Decrease CPU and Memory usage.
9. Increase TPS.

Concept of new Server stack as picture shown below

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| |  | | --- | |  | |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Figure 1.1 Application server stack.

Software Concept

1. ECProxy layer was designed by focus bottleneck problem in every part of our software as possible. We use threading to increase number of worker for handling incoming jobs which maybe causes race condition we can’t avoid but we can decrease it to become lockless by increase number of queues and set number of responsible thread of each queue as less as possible.

2. We use POSIX PTHREAD API to assign threads to CPUs we want.

3. We use ECProxy for reading and writing packet. After ECProxy received packet from socket, it push to request queue immediately and we use one thread to poll records in our queue and serve to worker thread then worker thread pass message to EC02(Java) by using JNI.

4. When ECProxy obtain result from EC02 (Process by AF), ECProxy will send the result back to sender (ES00).

Conceptual picture.



Figure 1.2 Conceptual of ECProxy Layer mechanisms.

**Approach to Class Design**

We use C++ to implement software structure and class relation design because this way is the balance of software efficiency and software management. Although “C++” is little slower than “C”, but still be very fast language and fast enough for low level IO operation.

1. **Class ECProxy** – Control everything from this class and handle shared resources such a Queue.
2. **Class JavaInterface** – Use java JNI feature for communication between C/C++ & Java.
3. **Class ConnectionPool** – Simple client connection management.
4. **Class Connection** – Just like simple C structure used for keep some necessary information.
5. **Class ThreadPool** – Simple threads management.
6. **Class WorkerThread** – Thread class for handling incoming jobs.
7. **Class Configuration** – This class is responsible for handling configuration both cold and warm configuration.
8. **Class QueueEntry, RequestQueueEntry, ResponseQueueEntry** – Object of incoming request and outgoing response.
9. **Class PrimaryRequestQueue, SecondaryRequestQueue** – Store request object.
10. **Class PreResponseQueue, ResponseQueue** – Store response object.



Figure 1.3 Conceptual & Simple relation class diagram.

## 

## Chapter3: Configuration

Old EC02 configuration consists of three files, EC02\_Configuration.xml, log4jProperties, appConfig.xml , but new E02 Configuration is simplify with only single file. This reduces complication of usage, see examples as shown below.

Old Configuration

1. EC02\_Configuration.xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<configuration>

<!-- time by secs-->

<reload-conf-interval value="1" />

<service-resources>

<service name="PCEF">mPCEFConfig.xml</service>

</service-resources>

</configuration>

1. log4j.properties

log4j.rootLogger=DEBUG,application

#file appender

log4j.appender.application=org.apache.log4j.DailyRollingFileAppender

log4j.appender.application.DatePattern='\_'yyyy-MM-dd

log4j.appender.application.File=./log/${groupName}.EC02.${serviceName}.${instanceEC}.log

log4j.appender.application.layout=org.apache.log4j.PatternLayout

log4j.appender.application.layout.ConversionPattern=%p|%d{HH:mm:ss}|%t|%c{2}.java|%x%m%n

#jetty server log

log4j.logger.org.eclipse.jetty=INFO

1. appConfig.xml

<?xml version="1.0" encoding="UTF-8" ?>

<configuration>

<cold>

<indicator-version1 value="false" />

<appilcation group="vPCEF" service="PCEF" instance="0" />

<host ip="127.0.0.1" port="8888" />

<queue-thread-pool value="2000" />

<library directory="lib/" name="vPCEF.jar" function="mpcef.control.MPCEFMain" />

</cold>

<warm>

<log value="FATAL|ERROR|WARN|INFO|DEBUG" />

</warm>

</configuration>

New Configuration

1. group.EC02.service.instance

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<configuration>

<cold>

<indicator-version1 value="true" />

<application group="pcrf" service="pcrf" instance="0" />

<host ip="0.0.0.0" port="4949" />

<library directory="/opt/equinoxAS/lib/" name="PCRF.jar" function="pcrf.control.PCRFMain" />

<!-- ECProxy -->

<ECReloadConfInterval value="100000" />

<ECThreadPool value="5"/>

<ECAutoCpuUtilize value="true"/>

<ECCpuUtilizeList value="4,5,6,7,8,9,10,11,12,13,14,15,16"/>

<ECRequestBufferSize value="65535"/>

<ECResponseBufferSize value="65535"/>

<ECStatsOn value="true"/>

<ECJavaOption value="-server"/>

<ECJavaOption value="-XX:CompileThreshold=1000"/>

<ECJavaOption value="-XX:ThreadStackSize=65535"/>

<ECJavaOption value="-XX:MainThreadStackSize=65535"/>

<ECJavaOption value="-Xms1G"/>

<ECJavaOption value="-Xmx1G"/>

<!--<ECJavaOption value="-Dlog4j.configuration=/opt/equinoxAS/conf/log4j.properties"/>-->

<ECJavaOption value="-Djavax.xml.xpath.XPathFactory:http://java.sun.com/jaxp/xpath/dom=com.sun.org.apache.xpath.internal.jaxp.XPathFactoryImpl"/>

<ECJavaOption value="-Djava.version=1.6"/>

<!--<ECJavaOption value="-Djava.compiler=NONE"/>-->

<!--<ECJavaOption value="-XX:UseSSE=0"/>-->

<ECJavaLibrary value="ec02\_library\_V2.2.3.jar"/>

<ECJavaLibrary value="eqxmsg\_demo\_v1.0.jar"/>

<ECJavaLibrary value="gson-2.2.2.jar"/>

<ECJavaLibrary value="kryo-2.20-all.jar"/>

<ECJavaLibrary value="gson-2.2.2.jar"/>

</cold>

<warm>

<log value="FATAL|ERROR|WARN|INFO|DEBUG" />

</warm>

</configuration>

## Chapter4: Performance Test Result

This section shows performance test result compared between old and new EC02 benchmark. After we change EC02 to new version we have found something are interesting, performance of system are improved.

Here below are result of testing for two phase of proofing of concept.

In first phase we optimize some code in EC02 and focus in Java technology included XML technologies, Java option tuning and others.

We try to change several XML parser and we got the best one, it is called “sTAX”.

STAX is a modern XML parser for high speed xml parsing, we replace DOM with sTax and get best result as shown in a table below.

**XML Parser (DOM vs STAX)**

|  |  |  |
| --- | --- | --- |
|  | Message (byte) | Time (ms.) |
| DOM | 859 | ~140 |
| sTAX | 859 | ~13 |
| DOM | 3889 | ~300 |
| sTAX | 3889 | ~30 |

Second phase we focus in ECProxy we have designed. At this point we re-engineer EC02 server (Java) by abandon JETTY and integrated with ECProxy instead.

Here below is result of the second phase.

**EC02 (Old vs New)**

|  |  |  |  |
| --- | --- | --- | --- |
|  | TPS | CPU% | Time (sec.) |
| EC02 (Old) | ~200 | 80 - 90 | ~20 |
| EC02 with ECProxy (New) | ~800 | 20 - 30 | ~10 |

**Chapter5: Summary**

ECProxy is flexible in development because we design for the future such separating any risk point maybe cause bottleneck problem and using OOP efficiency, with low coupling design we can cut or paste some module of ECProxy easily.

Every data structures implemented by C++ STL which you can replace them with your custom implementation as well.