# JCIS2：一個分散式持續整合系統
# JCIS2：A Distributed Continuous Integration System

歐伯浩　陳建村　鄭有進　徐天送
國立臺北科技大學資訊工程系
Department of Computer Science and Information Engineering
National Taipei University of Technology
Po-Hao Ou, Chien-Tsun Chen, Yu Chin Cheng, Tien-Song Hsu
Email: { t5598036,s1669021,yccheng,s4599002}@ntut.edu.tw

**摘　要**

在現今的軟體開發程序中，採用持續整合來降低整合錯誤的風險並提昇軟體品質已成為一種標準的作法。雖然在目前市面上有許多工具可支援持續整合，但專案的編譯與測試常相依於特定平台，例如只能在 Windows 或 Linux 上執行。若想要在每種特定平台上建立各自的持續整合環境，工作將非常繁雜且在維護管理上也十分不便。此外，持續整合的執行常會需要很長的時間，這些都會降低開發團隊的使用意願。但我們發現在持續整合流程中其實有許多工作是可同時執行的，例如靜態程式碼分析與程式碼的編譯測試，若能平行執行這些工作，將可縮短整合時間。因此在本論文中我們利用 JavaSpaces 技術，實做一個分散式持續整合系統。藉由分散式運算，將持續整合流程切割成多個較小的子工作，分派到不同的電腦執行，藉此解決專案平台相依性問題並且有效的加快整合速度。

關鍵字：持續整合系統、分散式運算、JavaSpaces

## 一、前言

持續整合（continuous integration）是一種透過自動化的方式，定時的對軟體系統的各模組與元件進行整合，以保持系統穩定度與品質的軟體開發最佳實務作法（best practice）。例如微軟公司的 Daily Build 與 Smoke Test[10]以及 Extreme Programming 所倡導的 Continuous Integration。目前已有許多持續整合軟體工具陸續被開發。例如 ThoughWorks 公司提供的 CruiseControl[6]、Apache 組織所開發的 Gump[2]與 Continuum[3]以及臺北科技大學資工系所開發的 JCIS[12]等。這些工具皆提供自動編譯程式與執行單元測試等基本建構流程。

但這些工具在執行整合時，整個建構流程的工作都在同一台電腦中執行。因此，對於某些相依於特定平台的軟體專案而言，我們必須在不同的平台上分別安裝一套持續整合系統來執行持續整合工作。這些複雜的安裝配置的動作以及系統的管理將會花費非常多的人力與資源。且一個持續整合系統上可能同時間有多個專案需要執行整合工作，若這些專案的整合工作中，需要執行很多的工作項目（如大量的測試案例），或是有非常耗時的工作項目（例如需要等待 I/O 回應的測試案例），則系統將無法頻繁地執行持續整合，而失去了持續整合的用意，也會降低開發團隊使用持續整合系統的意願。

我們建構一個分散式的持續整合系統。如此一來將可利用分散式運算的技術，簡化相依於不同平台軟體專案的整合複雜度，並且藉由整合工作分派至不同的電腦平行執行，縮短持續整合的執行時間。

## 二、相關研究

我們 檢視了幾個知名的開放原始碼持續者整合系統，CruiseControl、Anthill OS、Gump 與 Continuum。

- CruiseControl ： CruiseControl 是由 ThoughtWork 所提供的 Open Source 持續整合系統。CruiseControl 可同時支援 Ant 與 Maven，不過只提供了基本的建構流程與測試流程。CruiseControl 也有 Build Grid 的概念，但並不是真的分散式系統。而是藉由讓各機器上的 CruiseControl 可將其建置的 Logs 與 Artifacts 發佈到同一個 CruiseControl 上，並且透過此 CruiseControl 的 DashBoard 觀看其他 CruiseControl 的建置結果。
- AntHill OS：由商業公司 Urbancode 所提供的開放原始碼持續整合系統，以建構工具 Ant 為基礎。此系統安裝設定都非常簡易，但沒有一個統整性的報告瀏覽介面，不支援將專案分派至不同電腦整合的功能。
- Gump：Apache 的整合工具，可支援 Ant、Maven 等多種建構工具。可提供專案間相依性維護的支援。不支援將專案分派至不同環境整合。
- Continuum：Apache 提出的另一個持續整合工具。可支援 Maven1、Maven2 與 Ant，安裝與配置較簡易。並且有 Web 介面可提供專案內容的修改。不支援將專案分派到不同環境整合。

我們發現大部分的開放原始碼持續整合軟體

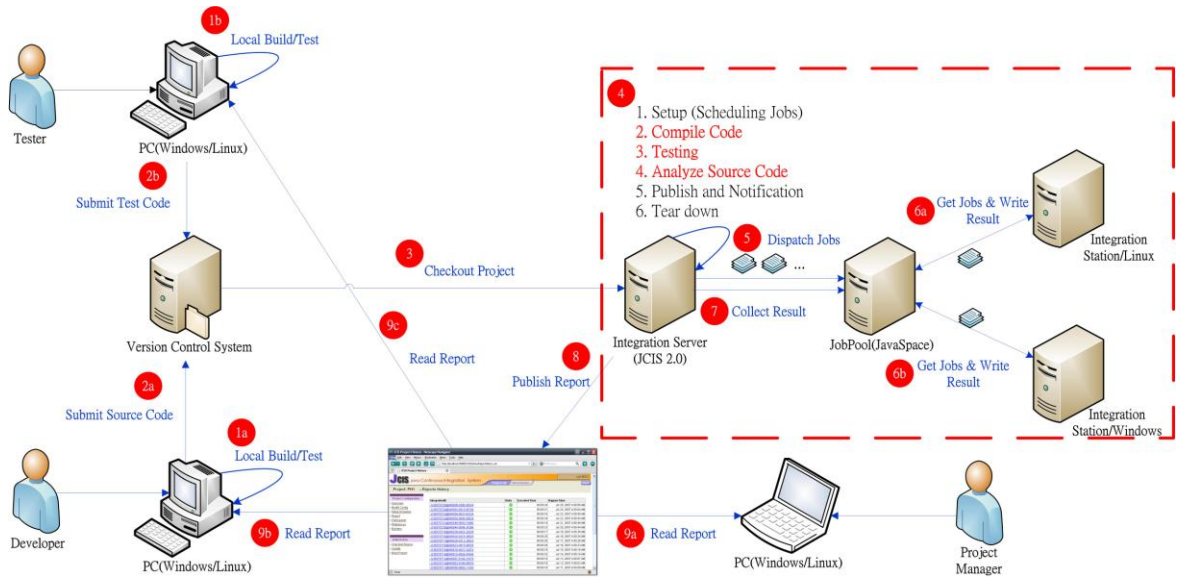


圖 1 分散式持續整合環境示意圖

都無法同時整合不同環境需求的專案，也沒有一個有效的方式可以加快整合的速度。因此我們計畫擴充過去所開發的開放原始碼持續整合軟體 JCIS，利用 JavaSpaces 技術，建構分散式的持續整合環境。

## 三、系統架構與設計

在之前的研究中[12]，我們提出一個持續整合工作應有的環境與運作流程，並且實做了一個支援 Java 與 C/C++的持續整合系統。請參考圖 1 JCIS 的分散式持續整合環境示意圖，虛線所框住的範圍即為我們的分散式持續整合系統。整個整合流程為：

1a. 程式開發人員在自己電腦上撰寫程式，並進行編譯與測試
2a. 程式開發人員把確認完成的原始碼放到一個共用的版本控管系統。
1b. 測試人員從版本控管系統取出專案最新版本的原始碼、撰寫並執行測試程式。
2b. 確認測試完成後，將測試碼放到版本控管系統上。
3. JCIS 自動到版本控管系統取出最新版本的原始碼與測試碼，並進行建構（Build）動作。
4. 建構活動為一個可擴充的建構工作流程，可依據不同專案的需求而加以設定。基本的建構工作包含編譯與測試。在 JCIS 中還包含對於原始碼分析與產生文件檔的建構支援。
5~8. JCIS 將整合工作自動分派到不同的電腦上執行，並且合併各電腦執行後所產生的結果。
8. 完成建構之後，JCIS 將產生的結果，包含目的檔、文件檔與分析報告放到 Web Server 上以提供專案開發人員瀏覽。
9. 專案經理、程式開發人員與測試人員透過瀏覽器觀看持續整合之結果。

我們參考了 Replicated-Worker Pattern[7]（如圖 2）來設計此系統架構。Replicate-Worker Pattern 主要用於處理平行計算系統設計的問題。此 Pattern 可分為兩個角色，Master 與 Worker，一個 Master 會配上多個 Worker。Master 負責將工作切割成更小的子工作並且將這些工作寫入 Space 以分派給 Worker，此外還必須將各 Worker 所回傳的工作結果整合。而 Worker 則負責從 Space 中取出工作，並且在執行工作內容之後將工作結果寫回 Space。

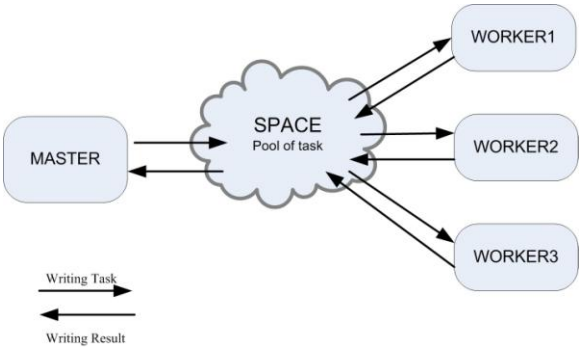


圖 2 Replicated-Worker Pattern

對照圖 3 的 JCIS 系統架構圖，JCIS 在從版本控管系統將專案 Checkout 後。由 Integrator 負責 Master 的角色，而 JavaSpaces 則負責 Space。首先 Integrator 把整合工作以 Builder 為單位切割成數個子工作，將每個子工作包裝成型態為 JobEntry 的工作物件並且透過 JobPool 寫入 JavaSpaces。

JobEntry 物件實做了 JavaSpaces 中的 Entry 介面，主要存放專案資料物件、工作內容物件與其他相關的整合設定物件。使用物件作為溝通方式有一個很大的好處，佈署在遠端 PC 的 Integration Station 只要知道物件的 Interface 即可，並不需要

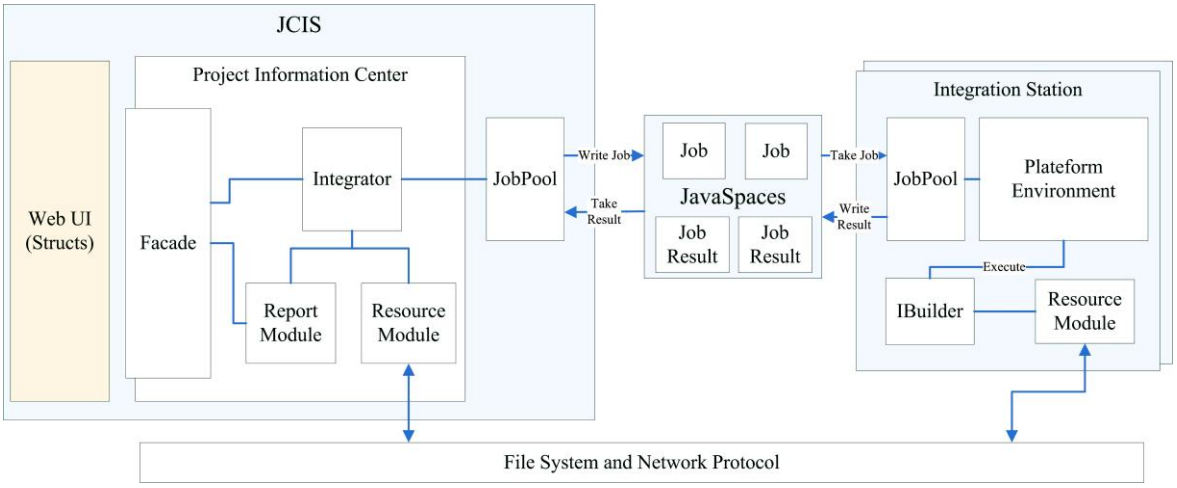


圖 3 JCIS 系統架構圖

擁有實做物件的程式碼。例如 Builder，在 JCIS 中每一個 Builder 即代表一種整合工作，例如 GCCcompile Builder、StatSVN Builder 等，所有的 Builder 都必須實做 IBuilder 介面。因此 Integration Station 只要知道 IBuilder 此 Interface 即可。Java 會利用網路自動去指定的路徑下載所需要的 Classes 檔案，此路徑稱為 Codebase。所以若往後 JCIS 的 Builder 內容有更改，我們並不需要讓所有的 Integration Station 都進行更新，只需要更換 Codebase 路徑上的檔案即可。

將所有的工作都包裝完之後，Integrator 必須設定工作的執行順序。因為一個專案的整合可能必須要參考其他專案的整合結果，在 JCIS 中稱之為 Reference Project。所以 JCIS 提供了一個 XML 格式的設定檔讓使用者設定專案與其 Reference Project 之間整合工作的執行順序。我們將工作分成兩種類型，一種為可平行處理的工作，另一種為必須循序處理的工作。可平行處理的工作我們會在設定檔中用 type 為 parallelGroup 的 work tag 將其包起來，則 Integrator 在分派工作時，就會同時將他們放入 JavaSpaces 中讓 Integration Station 去取得。而必須循序處理的工作則是用 type 為 sequenceGroup 的 work tag 包起來，JCIS 會依照設定的順序一個個將工作放入 JavaSpaces 中，只有當此工作被執行完畢且有相對應的工作結果也就是 JobResult 物件被回傳後，才會將下一個工作放入 JavaSpaces。Integrator 等到所有的 JobResult 都被回傳，就開始進行整合結果的合併並且產生整合報表。而 Integration Station 則擔任 Worker 角色，透過 JobPool 從 JavaSpaces 取出 JobEntry，並從 JobEntry 中取出 Builder 物件來執行。執行完畢後再將執行結果包裝成 Entry 寫回 JobPool。

圖 4 為 Integrator 與 Integration Station 之間利用 JavaSpaces 與 Http 網路協定通訊的溝通流程。在 Integrator 與 Integration Station 之間，除了利用 JavaSpaces 將工作內容分派到 Integration Station 之外，還必須將專案真正的檔案內容，例如專案原始碼、相依專案的整合執行結果等，分派到 Integration Station 上。但 JavaSpaces 只能傳送物件形式的資料，所以我們將專案檔案壓縮包裝成 ZIP 檔放在 Web Server 上，讓 Integration Station 在取得 JobEntry 之後可以根據 JobEntry 內所給的網址，自動的透過 Http 網路協定下載所需要的檔案。在工作執行完畢之後，Integration Station 也會根據 JobEntry 內所指定的網路路徑，將整合工作執行後的產出檔案壓縮後上傳。

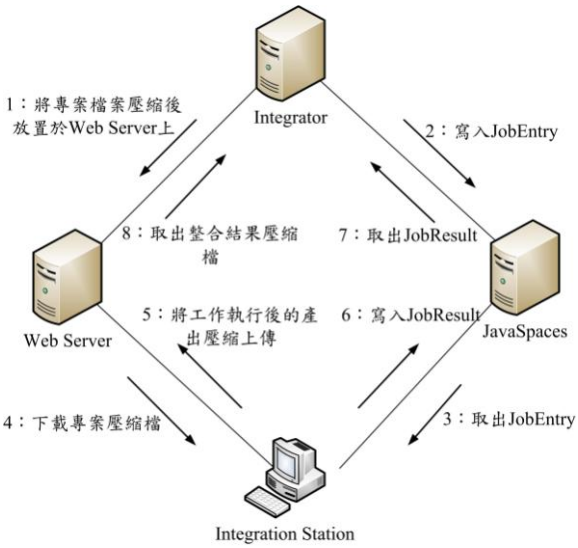


圖 4 Integrator 與 Integration Station 溝通流程

## 四、應用範例

系統擴充之後，如圖 4 所示，在忽略 JobEntry 與 JobResult 寫入 JavaSpaces 時間的情況下，每次專案整合最少會增加一次專案檔案與整合結果的壓縮、解壓縮的時間、一次專案壓縮檔下載與一次專案整合結果上傳的時間。因此我們希望能透過實驗觀察系統擴充之後，能提昇多少執行速度，或者會因為這額外的負擔反而降低了速度。

我們以本校軟體中心提出的通訊軟體整合型

計畫「WiMAX 無線通訊系統軟體與工具開發」(以下稱為 WiMAX 計畫) [14]作為我們的實驗對象。WiMAX 計畫在 Linux 上使用 C/C++語言開發，主要包含了六個 WiMAX 通訊軟體子計畫，分別為 SYS、VIDEO、MAC、PHY1、PHY2 與 CHN。各計畫的檔案資料如表 1。

表 1

| 子計畫名稱 | 總檔案數量/個 | 總檔案大小/KB | 總 LOC 數 |
|---|---|---|---|
| SYS | 188 | 545 | 6568 |
| VIDEO | 839 | 8386 | 15839 |
| MAC | 210 | 1444 | 16472 |
| PHY1 | 77 | 7188 | 3437 |
| PHY2 | 89 | 319 | 4112 |
| CHN | 66 | 365 | 4622 |

我們會進行程式編譯、測試與 StatSVN (Subversion 活動分析工具) 的整合工作。在這些整合動作之中，VIDEO、MAC、PHY1、PHY2、CHN 的編譯與 StatSVN 的整合動作是可平行處理的。而 SYS 因為必須整合其他五個子計畫，所以需要其他五個子計畫的編譯結果。等 SYS 編譯完成後，才會產生一個包含六個子計畫所有測試案例的 CppUnit 測試檔案，此時我們才能進行測試。整個執行流程如圖 5。

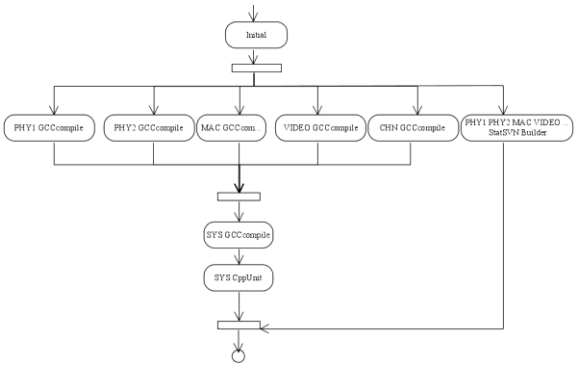


圖 5 工作流程圖

依照圖 5 的工作流程我們 XML 的整合工作執行順序設定檔如圖 6。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:BuilderworkFlow xmlns:ns2="http://www.example.org/workflow/">
    <work type="parallelGroup">
        <work type="builderWork" builderName="statsvn" projectName="phy1"/>
        <work type="builderWork" builderName="statsvn" projectName="phy2"/>
        <work type="builderWork" builderName="statsvn" projectName="channel"/>
        <work type="builderWork" builderName="statsvn" projectName="mac"/>
        <work type="builderWork" builderName="statsvn" projectName="video"/>
        <work type="builderWork" builderName="statsvn" projectName="sys"/>
        <work type="sequenceGroup">
            <work type="parallelGroup">
            <work type="builderWork" builderName="GCCcompile" projectName="phy1"/>
            <work type="builderWork" builderName="GCCcompile" projectName="phy2"/>
            <work type="builderWork" builderName="GCCcompile" projectName="channel"/>
            <work type="builderWork" builderName="GCCcompile" projectName="mac"/>
            <work type="builderWork" builderName="GCCcompile" projectName="video"/>
            </work>
            <work type="builderWork" builderName="GCCcompile" projectName="sys"/>
            <work type="builderWork" builderName="Cppunit" projectName="sys"/>
        </work>
    </work>
</ns2:BuilderworkFlow>
```

圖 6 整合工作執行順序設定檔

我們用來測試的電腦主要有兩種，作為 Integration Station 的 Linux 系統電腦其規格為：

| Integration Station | |
|---|---|
| 作業系統 | Fedora 8 |
| CPU | Intel Core2 6320 1.86GHz |
| 記憶體 | 1024 MB |

與作為 JCIS Server 的 Windows 系統電腦規格為：

| JCIS Server | |
|---|---|
| 作業系統 | Windowns XP |
| CPU | Intel Core2 T7200 2.00GHz |
| 記憶體 | 2048 MB |

我們在 JCIS Server 的電腦上先測試每個專案壓縮所花費的時間，並且在 Integration Station 測試解壓縮的時間。其結果如表 2，我們可以發現壓縮與解壓縮的時間花費並不高，但是專案約縮小成原來一半以下的大小，可有效的減少傳輸的時間，尤其在同一個專案需要分散到很多個不同 Integration Station 的情況下。

表 2

| 子計畫名稱 | 壓縮時間/秒 | 解壓縮時間/秒 | 壓縮後大小/KB |
|---|---|---|---|
| SYS | 0.1 | 0.2 | 226 (41%) |
| VIDEO | 1 | 2 | 2109 (25%) |
| MAC | 0.2 | 0.25 | 391 (27%) |
| PHY1 | 0.75 | 0.25 | 801 (11%) |
| PHY2 | 0.1 | 0.1 | 108 (33%) |
| CHN | 0.1 | 0.1 | 162 (44%) |

因為 WiMAX 計畫在 Linux 上開發，所以我們先不使用分散式架構，在作為 Integration Station 的電腦上循序的執行所有的整合工作。表 3 為 100 次實驗中各 Builder 執行的平均時間與整個整合所花費的總時間，這時間包括報表的產生，所以會比各 Builder 的執行時間總和還多。

表 3

| | GCCcompile/秒 | StatSVN/秒 | CppUnit/秒 | 總計 |
|---|---|---|---|---|
| CHN | 3 | 21 | | 24 |
| MAC | 45 | 23 | | 68 |
| PHY1 | 4 | 20 | | 24 |
| PHY2 | 15 | 22 | | 37 |
| VIDEO | 86 | 23 | | 109 |
| SYS | 46 | 24 | 60 | 130 |
| 整合花費時間 | 422 | | | |

實驗一：

我們以 JCIS Server 與一台 Integration Station 實驗分散式整合流程，此 Integration Station 負責所有的 Builder 動作，包括 GCCcompile、CppUnit 與 StatSVN。實驗結果如表 4：

表 4

| | GCCcompile | StatSVN | CppUnit | 總計 |
|---|---|---|---|---|
| CHN | 6 | 24 | | |
| MAC | 49 | 27 | | |
| PHY1 | 7 | 23 | | |
| PHY2 | 21 | 25 | | |
| VIDEO | 106 | 25 | | |
| SYS | 57 | 27 | 68 | |
| 整合花費時間 | 522 | | | |

由表我們可以觀察到，在只有一個 Integration Station 的情況下，整合時間反而多了將近 100 秒。這是因為各子計畫的 GCCcompile 編譯出來的.o 檔與執行檔六個子計畫加起來總共有 71 檔案佔 38432KB。所以因為這些檔案的壓縮、解壓縮與傳輸，而造成 Builder 執行時間與最後整合工作結果的時間增加，造成總整合時間為未使用分散式架構時的 123%。

實驗二：

以 JCIS Server 與兩台 Integration Station 進行實驗，兩台 Integration Station 皆可執行 GCCcompile、CppUnit 與 StatSVN。因為受限於 SYS 的編譯與測試必須依賴其他子計畫執行結果，且這兩個工作執行時間都很長。因此在理論上兩台 Integration Station 最佳的工作分配情況是讓其他五個子計畫的 GCCcompile Builder 先平行執行完畢，讓 SYS 的編譯與測試可以盡早開始（如圖 7）。我們根據實驗一中各 Builder 執行的平均時間再加上整合工作結果的時間，預估平均整合時間約為 285 秒。

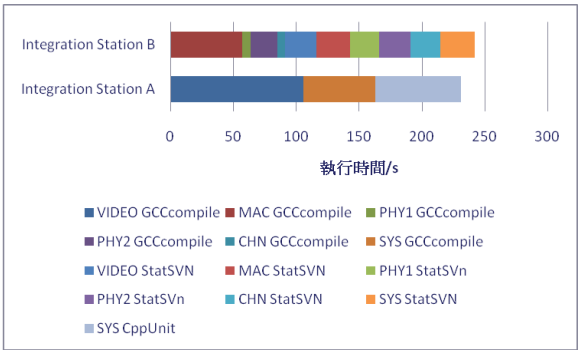


圖 7 兩台 Integration Station 下工作的最佳分配

但是由於 JCIS 目前是讓 Integration Station 自由爭取工作，所以會有不好的工作分配情況產生如圖 8。在這種情況下會有一台電腦因為必須要等待其他工作的整合結果而造成閒置，總整合時間可能會超過 370 秒。

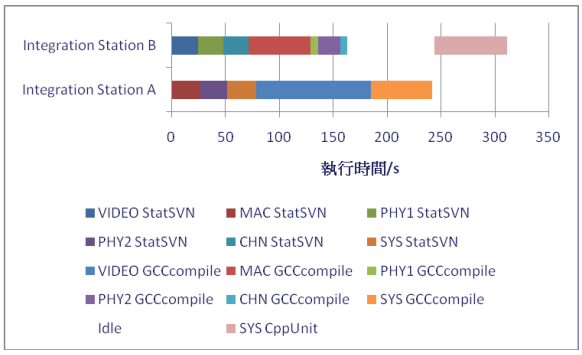


圖 8 兩台 Integration Station 下工作的最差分配

實驗結果如圖 9，平均的整合時間約在 280 秒，幾乎跟最佳時間相等，為未使用分散式架構的 66%。但由於工作分配的方式不固定，因此在不佳的工作分配情況下整合時間最高曾到 380 秒，但依然比未使用分散式架構時快。

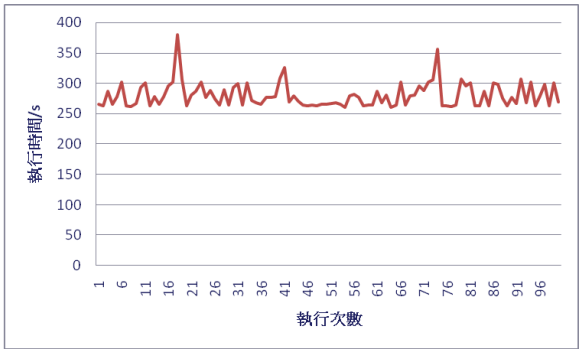


圖 9 兩台 Integration Station 的執行結果

實驗三：

在實驗三我們使用三台 Integration Station 進行實驗。在此情況下，最佳的執行情況跟實驗二時大致相同，依然受限於 SYS 的 GCCcompile 與 CppUnit，最佳的整合時間平均依然需要 270 秒（如圖 10），而因為 Integration Station 增加所以最差的情況可縮短到 330 秒（如圖 11）。圖 12 為此實驗的統計結果。

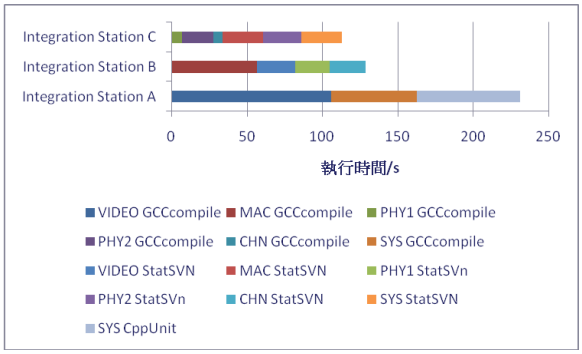


圖 10 三台 Integration Station 下工作的最佳分配

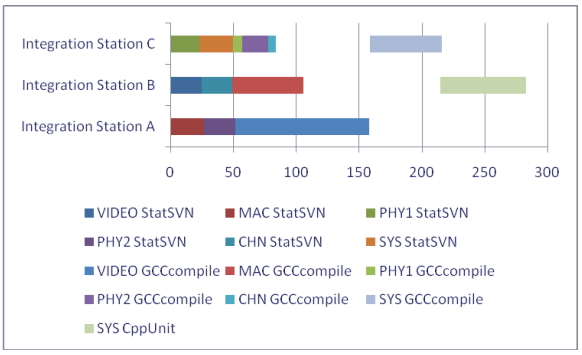


圖 11 三台 Integration Station 下工作的最差分配

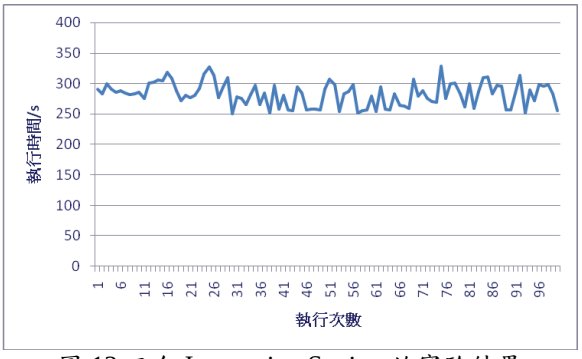


圖 12 三台 Integration Station 的實驗結果

實驗的結果平均 281 秒，因為 SYS 的 GCCcompile 與 CppUnit 的相依性關係，所以結果跟實驗二差不多。但是由統計圖表我們可以發現，實驗的時間變得較為穩定，不像實驗二會有突然超過 350 秒的情況。雖然沒有辦法在增加整合速度，但是可以減少不好的工作分配情況發生，讓整合時間變得較為穩定。

## 五、結論與未來展望

在本論文中，我們利用 JavaSpaces 技術擴充過去的研究成果—JCIS，實做了一個分散式的持續整合系統。在分散式的持續整合環境下，專案可以被分派到各式各樣特殊的電腦環境進行整合，因此我們只需要一個持續整合系統，就可整合各種不同環境需求的專案，也可將一個跨平台專案分派到不同平台的電腦進行測試，降低了在整合相依於特殊環境需求的專案時的複雜度。除此之外由於將整合工作分派到不同電腦上同時執行，因此也有效的加快了專案整合速度。

我們以本校軟體中心提出的 WiMAX 整合型計畫進行實驗，從實驗的結果中我們發現在 Integration Station 數量足夠的情況下，分散式的持續整合系統能有效的減少專案整合所花費的時間。雖然在專案有相依性的情況下，Integration Station 的增加無法再加快整合速度，但卻可讓整合的時間趨近穩定，減少工作分配不良的狀況。

JCIS 未來的發展方向將繼續以分散式架構為基礎，應用 Virtualization 技術開發一個支援虛擬化技術的持續整合環境。將各種不同的整合環境透過虛擬化技術儲存起來，使其可在實際執行整合建構工作時，能夠自動佈署及動態調適各種不同的整合環境，以順利在高度複雜的環境下完成整合工作。

本研究由國科會計劃 NSC 96-2218-E-027-017 補助，特此致謝。

## 參考文獻

[1] Anthill, http://www.urbancode.com/projects/anthill/default.jsp.
[2] Apache Gump, http://gump.apache.org/.
[3] Apache Continuum, http://maven.apache.org/continuum/.
[4] Apache Ant, http://ant.apache.org/.
[5] CppUnit, http://sourceforge.net/projects/cppunit.
[6] CruiseControl, http://cruisecontrol.sourceforge.net/.
[7] E. Freeman, S. Hupfer, and K. Arnold, JavaSpeces Principles, Patterns, and Practice, Addison Wesley, 1999.
[8] GCC, the GNU Compiler Collection, http://gcc.gnu.org/
[9] Jini Network Technology, http://java.sun.com/developer/products/jini/index.jsp.
[10] S. McConnell, *Rapid Development*, Microsoft Press, 1996.
[11] StatSVN, http://www.statsvn.org/.
[12] 鄭有進，一個支援Java應用程式的開放原始碼持續整合系統，國科會自由軟體專案研究計劃，計畫編號 NSC 93-2218-E-027-038.
[13] 鄭有進，WiMAX無線通訊系統軟體與工具開發－子計劃八：通訊軟體發展之持續整合系統，國科會自由軟體專案研究計劃，計畫編號 NSC 95-2218-E-027-016.
[14] 陳偉凱，楊士萱，吳和庭，劉玉蓀，尤信程，林丁丙，劉傳銘，謝金雲，鄭有進，WiMAX無線通訊系統軟體與工具開發，國科會自由軟體專案研究計畫，計畫編號 NSC 95-2218-027-009.