一個基於成對取樣之方法用以估計 OSGi 服務能源消耗與卸載決策
A Paired Sampling-Based Approach to Estimating OSGi Service Energy Consumption for Offloading Decision Makings

李信杰[1,2]、林孝融[2]
[1]國立成功大學計算機與網路中心
[2]國立成功大學資訊工程學系
Shin-Jie Lee, Xavier Lin
Email: jielee@mail.ncku.edu.tw, XavierLinX@gmail.com

摘　要

本研究提出一個基於 Paired Sampling 之方法估計一支 OSGi Service 執行時所需消耗的能源，並能夠動態地決定是否要將該 OSGi Service 從某一支手機卸載(Offload)到另一支手機(Mobile-to-Mobile)上執行以節省能源。目前已有許多研究能夠估計一個程式執行時所消耗的能源，然而這些方法大都必須倚賴特定硬體元件的資訊，並且無法在多執行緒的環境下準確地估計執行某一支程式的能源消耗。而我們所提出的方法並不倚賴特定的硬體資訊，也能夠在多執行緒的環境下準確地估計出某一個 OSGi Service 於執行時所消耗的能源，並做出卸載決策。實驗結果顯示出我們的方法能針對 8 個具多種隨機變數設定之 OSGi Services 做出正確的決策，卸載後最多能節省 43.67%的能源(Energy Gain)，且無論是估計本地端或遠端執行 OSGi Service 所需消耗的能源，其相對誤差值(Relative Errors)皆小於 5%。

關鍵字：OSGi service energy consumption estimation, energy-aware decision making, mobile-to-mobile OSGi service offloading

1. 前言

智慧型手機在現代已是日常生活中不可或缺的一個重要工具。隨著時代的演進，智慧型手機的硬體設備越來越進步，而手機上所執行的軟體功能也越來越複雜與多樣化。然而，電池容量的進步速度卻無法跟上手機的其他硬體設備（例如：CPU 的速度、RAM 的容量），因此如何優化手機的能源消耗是近年來十分熱門的一個議題。

將手機上部分計算量繁重的工作卸載(Offload)到遠端的機器上去執行，此概念被稱為 Cyber Foraging [1] [2]，在某些情況下可延長手機電池的使用時間。於本研究中，我們發展出一個以 Paired Sampling 為基礎之 Energy-Aware Decision Model (簡稱為 EA-OSGi)，能夠動態地決定是否要將一個 OSGi Service [3] Offload 到另一台遠端的手機去執行以節省本地端的手機能源，而於此研究中我們不考慮遠端手機的能源消耗。

目前已有許多研究能夠估計一個程式執行時所消耗的能源[4] [5] [6] [7] [8][9]，然而這些方法都必須倚賴特定的硬體元件資訊。例如在[4]中所提出的方法需要以 CPU Cycles 去估算執行一個 Method 所需消耗的能源，而有些程式執行時並不是只需要使用到 CPU，有可能需要做大量的 I/O 運算，因此這些需要倚賴特定硬體元件資訊的方法就無法準確地估計倚賴非考量硬體元件之程式執行時所需消耗的能源。除此之外，在[10]中作者指出這些 Utilization-Based 的方法無法在多個程式同時並行運作的環境(亦即於多執行緒的環境下)，準確地估計某一個程式執行時所需的能源消耗。於此研究中，我們提出以 Paired Sampling 的觀念為基礎來估計一支 OSGi Service 執行時所需消耗的能源，而此方法並不需倚賴特定的硬體資訊，因此能夠應用於各種程式上，實驗結果也呈現在多執行緒的環境下能準確地估計出執行一個 OSGi Service 所需消耗的能源。

於實驗中，我們在 Android 環境下以此研究所提出的方法實作了 8 個具多種隨機變數設定之 OSGi Service，並能夠將手機上的一個 OSGi Service 動態地 Offload 到另一支遠端的手機上去執行。實驗結果顯示出我們的方法能針對此 8 個 OSGi Services 做出正確的決策，卸載後最多能節省 43.67%的能源(Energy Gain)，且無論是估計本地端或遠端執行 OSGi Service 所需消耗的能源，其相對誤差值(Relative Errors)皆小於 5%。

本篇論文之架構如下：首先於第 2 章中進行文獻探討，接著將於第 3 章中詳細說明本研究所提出的系統架構與實作細節，於第 4 章中會呈現實驗評估的數據，最後將於第 5 章中作出總結。

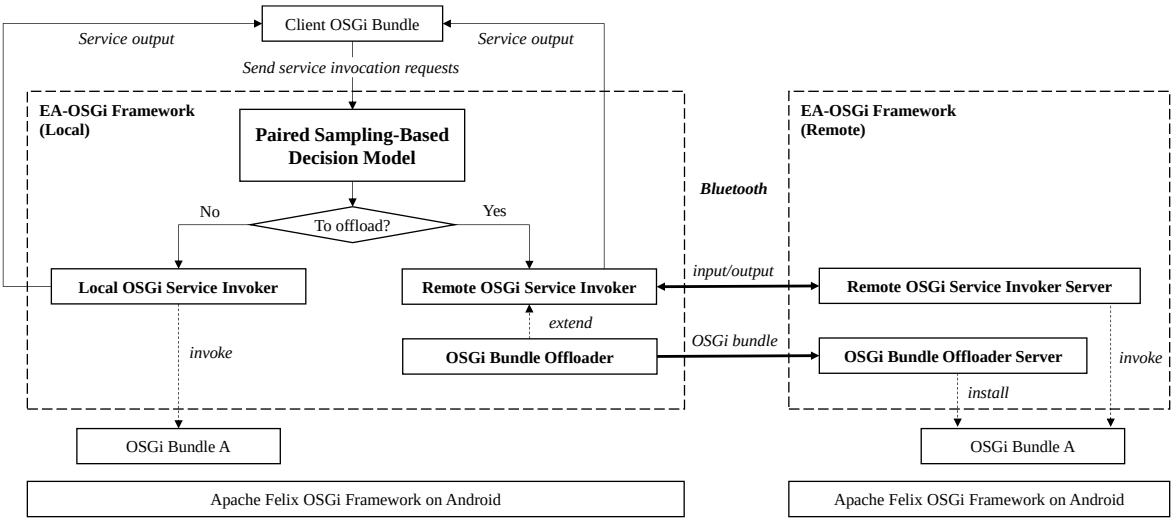

圖 1 系統架構圖

2. 文獻探討

MAUI [4]為一個能夠將手機上的程式碼 Offload 到雲端上的系統。MAUI 使用了 Monsoon [11]去測量一台手機的能源消耗，並能夠決定有哪些 Methods 應該被 Offload 到雲端去執行，或是應該在本地端被執行。Serendipity [5]為一個能夠使用其他手機的計算資源以達到加速運算以及節省能源的系統。此系統以類似 PowerBooter [13]的技術去測量能源，並決定有哪些 Methods 應該要被 Offload 到遠端的手機上去執行。

ThinkAir [6]為一個將 Method 卸載到雲端執行的系統，此系統實作了類似 PowerBooter [13]的 Energy model，能夠動態地預測執行中的 Method 的能源消耗。M. Akram 等人[14]則是擴充了 ThinkAir 的方法以支援 Delayed Offloading。Phone2Cloud [7]為一個基於 Computation Offloading 以達到節省手機能源的系統，並同時能夠增進手機程式的執行效能。此系統使用 PowerUsage [15]去測量能源消耗。

Y. D. Lin 等人[8]提出了一個 Offloading Framework，能夠同時提升執行效率與減少能源消耗。他們使用了 Android 內的電池記錄去取得能源資訊，並做出決策有哪些 Module 要被 Offload 到雲端或是 GPU 上去執行。F. A. Ali 等人[9]提出了一個手機硬體元件的 Power Model，並使用 AIOLOS [11]去動態測量能源，以及做出決策判斷是否要 Offload 一個 OSGi Service 到雲端上執行。

表 1 為相關研究之比較表。決定要將應用程式的哪個部分 Offload 到遠端去執行稱之為 Partitioning [16]，而於本研究中的 Partition Granularity 為一個 OSGi Bundle。本研究使用 Android 的 API [17]去測量能源，不需要額外的硬體設備，因此是屬於 Online 的能源測量方式，比 Offline 的測量方式更加來得有彈性[18]。除此之外，在測量於本地端執行一個 OSGi Service 所需消耗的能源時，我們並不倚賴任何硬體的資訊，而上述所提到的其他研究都需測量如 CPU、RAM 等等的硬體資訊，因此他們所提出的方法並不能適用於那些有使用到其他硬體的程式，而我們所提出的方法則沒有這個限制，可以應用於任何類型的程式。

3. EA-OSGi Framework

本研究所提出的系統架構圖如圖 1 所示，其中有四個主要元件，包括：(1) Paired Sampling-Based Decision Model，此為本系統中最核心的元件，用來估計執行一個 OSGi Service 所需花費的能源，以及做出是否要將此 Service offload 到另一支手機

表 1 相關研究的比較表

	評估 Local 能源消耗時所使用的 Power Model			Offloading 策略	
	Core Concept	Online/Offline	Hardware Component-Specific	Offloading Target	Partition Granularity
MAUI [4]	Utilization-Based	Offline (Monsoon)	Yes (CPU)	雲端伺服器	Method
Serendipity [5]	Utilization-Based	Online (like PowerBooter)	Yes (CPU)	行動裝置	Method
ThinkAir [6]	Utilization-Based	Online (likePowerBooter)	Yes (CPU, LCD, WiFi, Celllar)	雲端伺服器	Method
Phone2Cloud [7]	Utilization-Based	Online (PowerUsage)	Yes (CPU)	雲端伺服器	Application
Y. D. Lin et al. [8]	Utilization-Based	Online (Android battery log)	Yes (CPU)	雲端伺服器/GPU	Module
F. A. Ali et al. [9]	Utilization-Based	Offline (Monsoon)	Yes (CPU, RAM, LCD)	雲端伺服器	OSGi Bundle
Proposed Approach	Pairwise Sampling-Based	Online (Android API)	No	行動裝置	OSGi Bundle

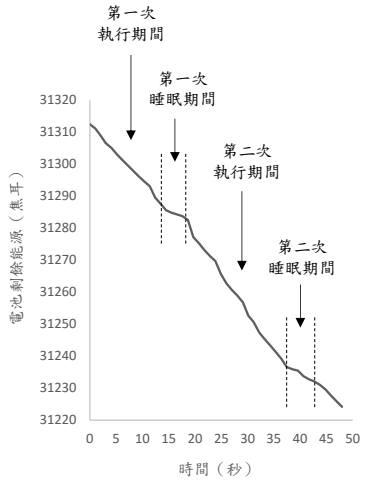

圖 2 執行一個 OSGi Service 所消耗的能源

的決策；(2) Local OSGi Service Invoker，此元件主要用來執行本地端的 Service Invocations；(3) Remote OSGi Service Invoker，此元件主要用來執行遠端的 Service Invocations；(4) Bundle Offloader，此元件主要用來將執行 OSGi Service 所需的 OSGi Bundle 傳輸到遠端的手機並進行安裝。

在執行一支 OSGi Service 時，Client 首先會送出執行此 OSGi Service 的請求，接著 Decision Model 會判斷是否要 Offload 此 OSGi Service Invocation。若做出的決策是不進行 Offload，那麼該 OSGi Service 就會在本地端被執行，因此會由 Local OSGi Service Invoker 去執行此 OSGi Service；反之，若做出的決策為要進行 Offload，那麼便會使用 Remote OSGi Service Invoker 去遠端地執行此 OSGi Service。除此之外，在遠端執行 OSGi Service 之前，會使用 Bundle Offloader 元件將所需的 OSGi Bundle 安裝到遠端的手機上。

在接下來的章節，我們首先會於 3.1 節中去探討如何估計執行一支 OSGi Service 所需消耗的能源，並做出是否要 Offload 此 OSGi Service 的決策。接著，將分別於 3.2 節與 3.3 節中探討本地端與遠端執行 OSGi Service 的實作細節。

3.1 Paired Sampling-Based Decision Model

在本研究中我們發展了一個 Power Model，用來估計執行一個 OSGi Service 所需消耗的能源，接著再由 Decision Model 判斷在本地端執行此 OSGi Service 會比較省能源、或是要 Offload 到遠端的手機上執行才會比較省能源。而在介紹 Power Model 之前，我們首先介紹我們所提出的這個方法的動機。圖 2 所表示的為執行一個 OSGi Service 所消耗的能源。在執行此 OSGi Service 時，背景同時有許多程式並行地在執行。第一次的 Service Invocation 發生於 0 秒至 14 秒，而第二次的 Service invocation

表 2 執行 OSGi Service 時的能源資料

i	Δt_R^i (s)	Δe_R^i (J)	Δt_S^i (s)	Δe_S^i (J)	is^i (byte)	os^i (byte)
1	14.2	26.88	2.0	1.03	5226	126350
2	18.6	44.8	2.0	1.33	5575	130406
3	21.6	36.98	2.0	1.15	6150	138710
…	…	…	…	…	…	…
19	24.8	50.48	2.0	1.84	6439	140826
20	24.5	47.86	2.0	2.16	6502	156086

發生於 19 至 37 秒。第一次與第二次的 Service Invocation 之間有 5 秒的間隔，而第二次與第三次的 Service Invocation 之間有 6 秒的間隔。

由圖 2 中可看出，於執行期間的線段比非執行期間(在此論文中稱之為睡眠期間)的線段還要來得陡，亦即在執行 OSGi Service 的期間所消耗的電池能源、比沒有執行時所消耗的電池能源還要來得劇烈。藉由此觀察，我們推斷出一個現象：儘管在背景同時有許多程式並行地在執行，只要額外多執行一個 OSGi Service 其電池能源就會消耗的比較快，而沒有執行該 OSGi Service 時電池能源就會消耗的比較慢。因此在概念上，我們藉由將執行 OSGi Service 時（執行期間）所消耗的能源，減去沒有執行時（睡眠期間）所消耗的能源，以得到執行該 OSGi Service 所消耗的「純」能源。

接下來，我們將正式地定義執行一個 OSGi Service 時的執行期間、以及於執行期間時所消耗的能源。

Definition 1 (執行一個 OSGi Service 的執行期間以及睡眠期間). *令* $\Delta t_R^i = t_{R.end}^i - t_{R.start}^i$ *為於第* i $(1 \le i \le n)$ *次執行一個 OSGi Service 的執行期間。令* $\Delta t_S^i = t_{S.end}^i - t_{S.start}^i$ *為執行該 OSGi Service 後的睡眠時間，其中* $t_{S.start}^i = t_{R.end}^i$ *以及* $t_{S.start}^i = t_{R.end}^i + \alpha$ *。*

表 2 列出執行某一個 OSGi Service 時的能源資料，而圖 3 則是其示意圖。以第一次執行該 OSGi Service 為例，執行期間的起點與終點分別為 $t_{R.start}^1$ 與 $t_{R.end}^1$，而其執行時間為 $\Delta t_R^1 = 14.2$ 秒。α 代表的是一個期間，於實驗中 α 的預設值為 2 秒，因此 $\Delta t_S^1 = 2$ 秒。換句話說，在每次的 Service Invocation 結束之後，都會休息 2 秒。

Definition 2 (於執行期間與睡眠期間每秒所消耗的能源). *令* $e_{R.start}^i$ *與* $e_{R.end}^i$ *分別代表於時間點* $t_{R.start}^i$ *與* $t_{R.end}^i$ *所測量得到的電池剩餘能源。令* $e_{S.start}^i$ *與* $e_{S.end}^i$ *分別代表於時間點* $t_{S.start}^i$ *與* $t_{S.end}^i$ *所測量得到的電池剩餘能源。於* Δt_R^i *期間中每秒所消耗的能源的計算方式為* $e_{R/t}^i = \Delta e_R^i / \Delta t_R^i$ *，而於* Δt_S^i *期間中每秒所消耗的能源的計算方式為* $e_{S/t}^i = \Delta e_S^i / \Delta t_S^i$ *，其中* $\Delta e_R^i = e_{R.start}^i - e_{R.end}^i$ *，且* $\Delta e_S^i = e_{S.start}^i - e_{S.end}^i$ *。*

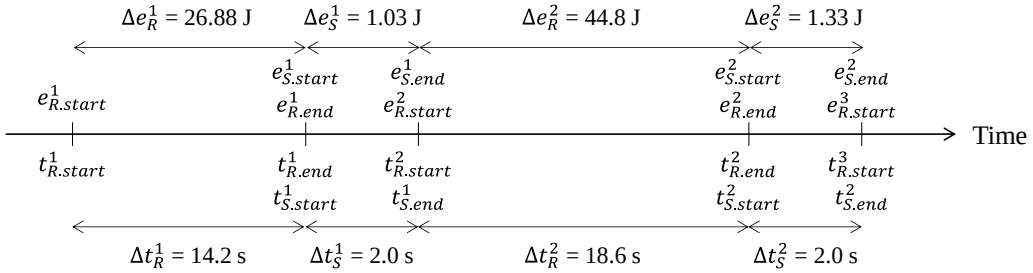

圖 3 表 2 中一個 OSGi Service 執行與非執行時所紀錄之時間與能源消耗

同樣以表 2 作為例子，對於第一次執行該 OSGi Service 而言，於時間點$t_{R.start}^1$與$t_{R.end}^1$所測量得到的電池剩餘能源分別為 31312.56 焦耳與 31285.68 焦耳，因此於執行期間中每秒所消耗的能源為 $e_{R/t}^1 = \Delta e_R^1/\Delta t_R^1 = (31312.56 - 31285.68)/14.2 = 1.89$（焦耳/秒）。同樣地，於睡眠期間中每秒所消耗的能源為$e_{S/t}^1 = \Delta e_S^1/\Delta t_S^1 = 1.03/2.0 = 0.52$（焦耳/秒）。

在定義了 OSGi Service 於執行期間每秒所消耗的能源（$e_{R/t}^i$）、以及於睡眠期間每秒所消耗的能源（$e_{S/t}^i$）之後，接著我們將採用最新幾次的能源資料分別去計算$e_{R/t}^i$與$e_{S/t}^i$的平均，以及使用 Paired Sample T-Test 去檢驗此兩平均值是否有顯著差異。

Definition 3 (執行一個 Local OSGi Service 每秒所消耗的估計能源). *令$P_m = \{\langle e_{R/t}^{n-m+1}, e_{S/t}^{n-m+1}\rangle, \dots, \langle e_{R/t}^n, e_{S/t}^n\rangle\}$為一個最新的 m 個 Pairs 的集合。令$\bar{e}_{R/t} = \frac{1}{m}\sum_{i=n-m+1}^{n} e_{R/t}^i$ 以及$\bar{e}_{S/t} = \frac{1}{m}\sum_{i=n-m+1}^{n} e_{S/t}^i$。Null Hypothesis 表示$\bar{e}_{R/t}$與$\bar{e}_{S/t}$之間的差等於 0，而 Alternative hypothesis 表示$\bar{e}_{R/t}$與$\bar{e}_{S/t}$之間的差不等於 0。*

$$H_0: \bar{e}_{R/t} - \bar{e}_{S/t} = 0$$
$$H_1: \bar{e}_{R/t} - \bar{e}_{S/t} \neq 0$$

執行一個 OSGi Service 每秒所消耗的能源估計為

$$e_K = \begin{cases} \bar{e}_{R/t} - \bar{e}_{S/t}, & if\ H_0\ is\ rejected \\ N/A, & else \end{cases}$$

Definition 4 (執行一個 Local OSGi Service 所消耗的估計能源). *於第i次執行一個 OSGi Service 所消耗的能源估計為$e_{local}^i = e_k \times \Delta t_R^i$。*

Definition 3 定義了執行一個 OSGi Service 所消耗的估計能源（記作e_K），而於 Definition 4 中則定義了如何使用e_K去估計每一次的 OSGi Service Invocation 所會消耗的能源。在實驗中，我們將 m 的值設定為 25，亦即我們會使用最新的 25 筆能源記錄來估計 OSGi Service 的能源消耗。以表 2 的例子而言，$P_m = \{\langle \frac{26.88}{14.2}, \frac{1.03}{2.0}\rangle, \dots, \langle \frac{47.86}{24.5}, \frac{2.16}{2.0}\rangle\}$，而$\bar{e}_{R/t} = 2.02$（焦耳/秒）、$\bar{e}_{S/t} = 0.92$（焦耳/秒）。以 Pair Sample T-Test（自由度為 19，信心水準為 95%）檢驗P_m的結果，t-value 的值為 7.060（> 2.100），這代表$\bar{e}_{R/t}$與$\bar{e}_{S/t}$之間有顯著的差異。由於H_0被 Rejected，我們可以計算$e_k = 1.1$（焦耳/秒），代表執行該 OSGi Service 每秒會消耗 1.1 焦耳的能源。e_k計算出來之後，我們便可以反推之前每一次 OSGi Service Invocation 所消耗的能源。例如，第一次執行 OSGi Service 所消耗的能源為$e_{local}^1 = 1.1 \times 14.2 = 15.62$焦耳，而第二次執行 OSGi Service 所消耗的能源為$e_{local}^2 = 1.1 \times 18.6 = 20.46$焦耳，依此類推。

到目前為止，我們已經探討完如何估計一個 OSGi Service 於本地端執行所會消耗的能源。接下來，我們會詳細地定義如何估計遠端執行一個 OSGi Service 所會消耗的能源，以及作出是否需要 Offload 此 OSGi Service 的決策。

Definition 5 (估計執行一個 Remote OSGi Service 所需消耗的能源). *執行一個 Remote OSGi Service 所需消耗的能源估計為$e_{offload}^i = (is^i + os^i) \times e_{BT}$，其中$is^i$與$os^i$分別代表第$i$次的 Service Invocation 的輸入與輸出資料量大小，而e_{BT}代表以藍芽傳輸一個位元組所需消耗的能源。*

以表 2 所列出的能源資料為例，第一次 Service Invocation 時的輸入與輸出資料量大小分別為 5226 個位元組以及 126350 個位元組，而於實驗中e_{BT}所測出來的值為8×10^{-7}（焦耳/位元組）；因此，我們可以估計於第一次遠端執行一個 OSGi Service 所需消耗的能源為$e_{offload}^1 = 131576 \times 8 \times 10^{-7} = 0.11$焦耳。

由 Definition 4 與 5，我們已經能夠估計分別於本地端與遠端執行一個 OSGi Service 所需消耗的能源。接著，我們便可以比較在本地端執行此 OSGi Service 會比較省能源，亦或是 Offload 此 OSGi Service 到遠端的手機上執行才會比較省能源，並做出決策。

Definition 6 (是否要 Offload 一個 OSGi Service Invocation 的決策). *令最新 m 筆於本地端執行一個 OSGi Service 的能源的平均值為$\bar{e}_{local} =$*

$$Decision = \begin{cases} To\ offload, & if\ \bar{e}_{local} > \bar{e}_{offload} \\ Not\ to\ offload, & otherwise \end{cases}$$

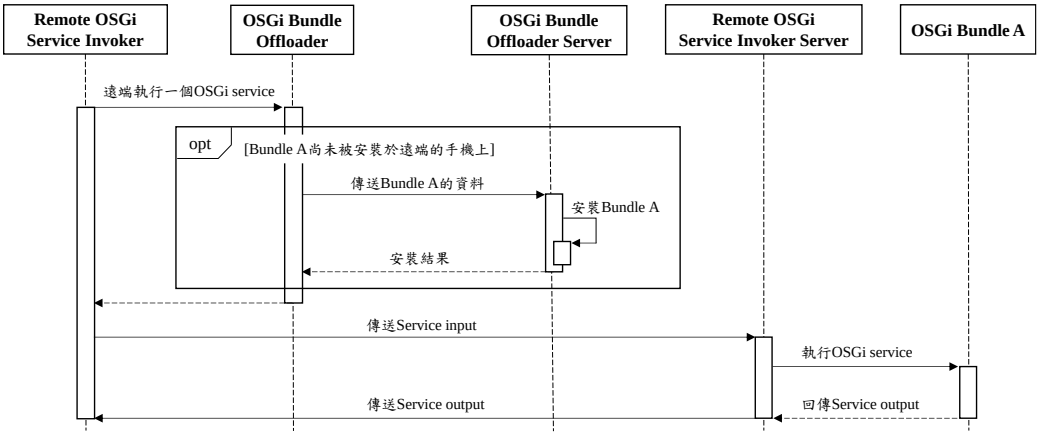

圖 4 Remote OSGi Service Invoker 與 OSGi Bundle Offloader 的 Sequence Diagram

$\frac{1}{m}\sum_{i=n-m+1}^{n} e_{local}^{i}$。*令最新 m 筆於遠端執行一個 OSGi Service 的能源的平均值為* $\bar{e}_{offload} = \frac{1}{m}\sum_{i=n-m+1}^{n} e_{offload}^{i}$ *。是否要 Offload 此 OSGi Service Invocation 的決策為*

以表 2 的資料為例，$\bar{e}_{local}$與$\bar{e}_{offload}$計算出來的結果分別為 24.46 焦耳與 0.12 焦耳，因為$\bar{e}_{local}$的值比$\bar{e}_{offload}$還大，因此最後做出的決策為要 Offload 此 Service Invocation 至遠端的手機上執行以節省能源。

3.2 Local OSGi Service Invoker

Local OSGi Service Invoker 的主要功能為於本地端執行一個 OSGi Service，表 3 列出了在執行 OSGi Service 時的 Java 程式碼片段。我們首先利用 OSGi 所提供的 API *getServiceReference()* 以及 *getService()*去取得 OSGi Service 的物件，接著再利用 Java 的 Reflection 機制取得該物件的類別所宣告的所有方法，並使用 Apache Commons ClassUtils 所提供的 API *isAssignable()*去搜尋符合所傳入的 *args* 的方法，最後再次利用 Java 的 Reflection 的方法 *invoke()*去動態地執行該 OSGi Service。

3.3 Remote OSGi Service Invoker 與 OSGi Bundler Offloader

Remote OSGi Service Invoker 的主要功能為遠端地執行一個 OSGi Service，其中 OSGi Bundle Offloader 擴充了 Remote OSGi Service Invoker，而其主要功能為將執行 OSGi Service 所需要的 OSGi Bundle 安裝到遠端的手機上。圖 4 顯示的是 Remote OSGi Service Invoker 與 OSGi Bundle Offloader 的 Sequence Diagram。當 Decision Model 做出的決策為要 Offload 一個 OSGi Service，則會透過 Remote OSGi Service Invoker 去遠端地執行此 OSGi Service。在執行之前，OSGi Bundle Offloader 會先確認遠端手機上是否已經有安裝所需的 OSGi Bundle，例如圖 4 中的 Bundle A。若遠端手機上沒有安裝，OSGi Bundle Offloader 則會先將所需的 OSGi Bundle 的資料透過藍芽傳送至遠端手機並進行安裝。

在確認遠端的手機上有所需的 OSGi Bundle 之後，Remote OSGi Service Invoker 會將執行此 OSGi Service 所需的輸入資料傳送至遠端的手機，並透過遠端手機上的 Remote OSGi Service Invoker Server 去執行該 OSGi Service，最後會再將 OSGi Service 的執行輸出回傳給本地端的手機。

4. 實驗評估

我們於實驗中評估了使用我們所開發的 Framework 可以省下多少能源，以及檢驗我們所提出的方法在計算能源時的誤差值。於 4.1 節中，首先會說明我們的實驗設計；於 4.2 節中，將會呈現節省的能源數據；最後於 4.3 節中，將會檢驗估計能源的誤差值。

表 3 執行一個 OSGi 服務的 Java 程式碼片段

```
1  void invokeService(String serviceName, String methodName, Object[] args) {
2    ServiceReference ref = getServiceReference(serviceName);
3    Object serviceObject = getService(ref);
4    Method[] methods = serviceObject.getClass().getDeclaredMethods();
5    for (Method method : methods)
6      if (method.getName().equals.(methodName))
7        if (isAssignable(toClass(args), method.getParameterTypes())
8          method.invoke(serviceObject, args);
9  }
```

表 4 於實驗評估時所使用的 OSGi Services

Service #	Service Name	Method Name	Input (Arguments)	Output
1	SimRank	compute	1 個包含多個點的圖	點的排序結果
2	ZipService	zip	1 個 Zip 檔案	處理之後的 Zip 檔案
3	AudioConverter	convert	1 個 Wave 音訊檔	1 個 Mp3 音訊檔
4	MatrixCalculator	multiply	2 個矩陣M_1與M_2	1 個矩陣$M_3 = M_1 M_2$
5	Dijkstra	calculate	1 張圖與 1 個起點s	從s到其他點的最短路徑
6	ImageFactory	resize	1 張圖片	調整過大小後的圖片
7	Downloader	download	1 個網路連結	1 個檔案
8	Uploader	upload	1 個檔案	沒有任何輸出

4.1 實驗設計

於實驗中，我們以我們所提出的 Framework 開發了一個 Android App，並實作了 8 個 OSGi Services，詳細的內容列於表 4。我們以 Google Nexus 6 作為使用者的手機（Client），而以 Sony Xperia Z3 作為一台遠端的手機（Server）。

第 1 個 OSGi Service 的功能為以 SimRank 演算法計算一個圖中任兩點之間的相似度。第 2 個 OSGi Service 的功能為處理一個含有大量檔案的 Zip 壓縮檔。第 3 個 OSGi Service 的功能為將一個 Wave 音訊檔轉換成 Mp3 音訊檔。第 4 個 OSGi Service 的功能為做矩陣相乘的運算。第 5 個 OSGi Service 的功能為以 Dijkstra 演算法計算一個圖中兩點的最短路徑。第 6 個 OSGi Service 的功能為調整一張圖片的大小。第 7 個 OSGi Service 的功能為下載一個檔案。最後是第 8 個 OSGi Service，其功能為上傳一個檔案。

由於在本地端執行時不需使用藍芽傳輸資料，因此於本地端執行剩下的 Service Invocations 的時間會相對地比於遠端執行還要來得短。在此實驗中，我們將比較兩者於做完決策後相同時間之內所消耗的能源差距，觀察是否將 OSGi Service offload 到遠端執行會比較省能源。為了評估使用我們所開發的 Framework 能夠節省多少能源，我們的實驗流程如下：(1)首先某個 OSGi Service 在本地端先執行一定次數以蒐集能源資料；(2)在本地端執行一定次數之後，Decision Model 會做出決策判斷該 OSGi Service 是否要被 Offload；(3)做出決策之後，該 OSGi Service 會額外被執行 5 次。我們所比較的能源為步驟(3)所消耗的能源。例如，於步驟(2)中所做出的決策為要 Offload 該 OSGi Service，則我們會先測量遠端執行 5 次 OSGi Service 的能源；接著，我們會再重複執行一次步驟(1)，但省略步驟(2)，刻意讓該 OSGi Service 在本地端執行，接著會於步驟(3)中測量於本地端執行 5 次 OSGi Service 的能源。最後，我們會去計算兩者能源的差距，以評估使用我們開發的 Framework 所能省下的能源。

4.2 節省的能源 (Energy Gain)

在提出實驗數據結果之前，我們會先正式的定義於實驗中所節省的能源的計算方式。

Definition 7 (在做完決策之後完成剩下的 Remote Service Invocations 的時間). *令* $\Delta t_{offload}^{remain^i} = t_{offload.end}^{remain^i} - t_{offload.start}^{remain^i}$ *為在做出決策後於第*i*次執行 Remote OSGi Service 的時間，其中*$n+1 \le i \le n+5$*，且*n*表示在做出決策之前於本地端執行 OSGi Service 的次數。令*$\Delta t_{bundle} = t_{bundle.end} - t_{bundle.start}$*為將 OSGi bundle offload 到遠端手機的時間，其中*$t_{bundle.end} = t_{offload.start}^{remain^{n+1}}$*。在做完決策之後完成剩下的 Remote service invocations 的時間的計算方式為* $\Delta t_{offload}^{remain} = t_{offload.end}^{remain^{n+5}} - t_{bundle.start}$ *。*

Definition 8 (在做完決策之後完成剩下的 Local Service Invocations 的時間). *令* $\Delta t_{local}^{remain^i} = t_{local.end}^{remain^i} - t_{local.start}^{remain^i}$ *為在做出決策後於第*i*次執行 Local OSGi Service 的時間。在做完決策之後完成剩下的 Local service invocations 的時間的計算方式為*$\Delta t_{local}^{remain} = (t_{local.start}^{remain^{n+1}} + \Delta t_{offload}^{remain}) - t_{local.start}^{remain^{n+1}}$ *。*

Definition 9 (在做完決策之後完成剩下的 Remote Service Invocations 所需消耗的能源). *令* $e_{offload.start}^{remain}$ *與* $e_{offload.end}^{remain}$ *分別代表於時間點* $t_{bundle.start}$ *與*$t_{offload.end}^{remain^{n+5}}$*所測量得到的剩餘電池能源。在做完決策之後完成剩下的 Remote service invocations 所需消耗的能源的計算方式為* $\Delta e_{offload}^{remain} = e_{offload.start}^{remain} - e_{offload.end}^{remain}$ *。*

Definition 10 (在做完決策之後完成剩下的 Local Service Invocations 所需消耗的能源). *令* $e_{local.start}^{remain}$ *與*$e_{local.end}^{remain}$ *分別代表於時間點*$t_{local.start}^{remain^{n+1}}$ *與*$(t_{local.start}^{remain^{n+1}} + \Delta t_{offload}^{remain})$*所測量得到的剩餘電池能源。在做完決策之後完成剩下的 Local service invocations 所需消耗的能源的計算方式為* $\Delta e_{local}^{remain} = e_{local.start}^{remain} - e_{local.end}^{remain}$ *。*

在正式定義完於做完決策之後完成剩下的 OSGi Service Invocations 所需消耗的能源之後，我們以下列公式計算使用我們的 Framework 所能節省的能源：

$$\text{Energy Gain} = \frac{\Delta e_{local}^{remain} - \Delta e_{offload}^{remain}}{\Delta e_{local}^{remain}}.$$

表 5　執行 Local OSGi Services 與 Remote OSGi Services 所消耗的能源比較表

Service #	$\Delta e_{offload}^{remain}$	$\Delta e_{local}^{remain}$	$\Delta e_{local}^{remain} - \Delta e_{offload}^{remain}$	Energy gain	Service #	$\Delta e_{local}^{remain}$	$\Delta e_{offload}^{remain}$	$\Delta e_{local}^{remain} - \Delta e_{offload}^{remain}$
1	171.94	305.23	133.29	43.67%	5	78.36	87.63	-9.27
2	138.32	222.13	83.81	37.73%	6	169.28	184.54	-15.26
3	198.99	265.84	66.85	25.15%	7	105.96	116.32	-10.36
4	84.53	95.73	11.2	11.70%	8	122.35	140.62	-18.27

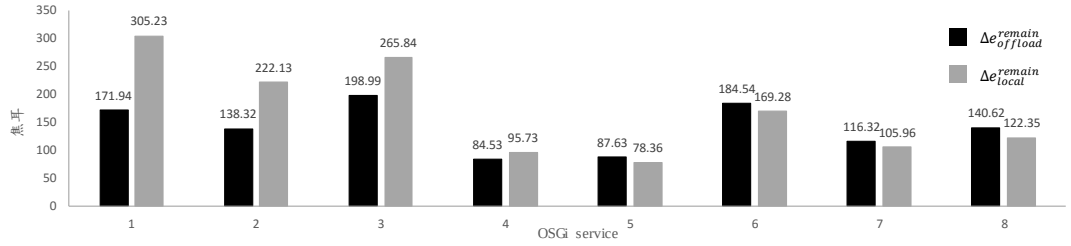

圖 5 在做完決策後完成剩下的 OSGi Service invocations 所需消耗的能源比較圖

表 5 列出執行 Local OSGi Services 與 Remote OSGi Services 所消耗的能源比較表，而圖 5 則是以長條圖的方式呈現。對於前 4 個 OSGi Services，根據 Power Model 所估計出來的能源，Decision Model 所做出的決策是要 Offload，而由圖 5 中可看出，前 4 個 OSGi Services 其$\Delta e_{offload}^{remain}$的值都比$\Delta e_{local}^{remain}$還要低，代表將 OSGi Service offload 到遠端手機去執行的確比較省能源，且最高可以節省 43.67%的能源。另一方面，對於後 4 個 OSGi Services 而言，其$\Delta e_{offload}^{remain}$的值都比$\Delta e_{local}^{remain}$還要高，代表將 OSGi Service offload 到遠端手機去執行反而比較耗電，而我們所做出來的決策是不 Offload。因此，針對這 8 個 OSGi Service 所作出的決策皆是正確的。

4.3 估計能源的誤差值 (Relative Errors)

在這個章節中，我們會檢驗我們所估計的能源與實際測量的能源兩者之間的誤差值。在檢視實驗數據之前，我們先正式地定義估計 Local 與 Remote 的 OSGi Service Invocation 能源誤差值的計算方式。

Definition 11 (估計 Local OSGi Service Invocation 能源的誤差值). *令*$\Delta\hat{e}_R^i = (e_K + \bar{e}_{S/t}) \times \Delta t_R^i$*為第*$i$*次執行 Local OSGi Service invocation 的估計能源。估計第*i*次的 Local OSGi Service invocation 能源的誤差值的計算方式為*$\delta_{local}^i = \frac{\left|\Delta\hat{e}_R^i - \Delta e_R^i\right|}{\Delta e_R^i}$*。*

Definition 12 (估計 Remote OSGi Service Invocation 能源的誤差值). *令*$\Delta\hat{e}_{offload}^{remain^i} = e_{BT} \times (is^i + os^i) + \bar{e}_{S/t} \times \Delta t_{offload}^{remain^i}$*為第*$i$*次執行 Remote OSGi Service invocation 的估計能源。估計第*i*次的 Remote OSGi Service invocation 能源的誤差值的計算方式為*$\delta_{offload}^i = \frac{\left|\Delta\hat{e}_{offload}^{remain^i} - \Delta e_{offload}^{remain^i}\right|}{\Delta e_{offload}^{remain^i}}$*。*

表 6 估計 Local 與 Remote OSGi Service invocation 能源的誤差值（總表）

	Service 1	Service 2	Service 3	Service 4	Average
$\bar{\delta}_{local}$	1%	5%	5%	3%	3.5%
$\bar{\delta}_{offload}$	4%	4%	3%	5%	4%

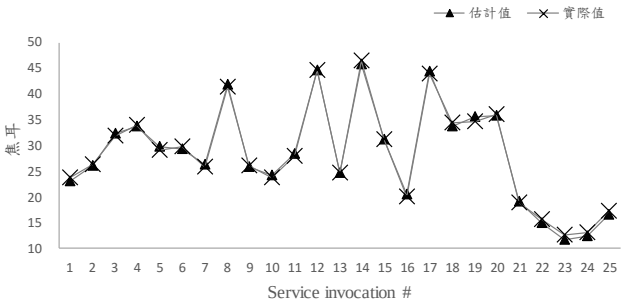

圖 6 估計 Local 與 Remote OSGi Service invocation 能源的誤差值（SimRank OSGi Service）

圖 6 描繪出了執行 SimRank OSGi Service 時，估計 Local 與 Remote service invocation 能源的誤差值。由圖中可以看出，估計值與實際值所描繪出的線條幾乎重疊，代表我們所估計的值其誤差相當的低。而表 6 則是列出了估計值與誤差值的總表，其中$\bar{\delta}_{local}$和$\bar{\delta}_{offload}$分別代表各次的δ_{local}^i與$\delta_{offload}^i$的平均值。由表中可以看出，無論是估計 Local OSGi Service Invocation 的能源，或是估計 Remote OSGi Service Invocation 的能源，我們所提出的方法的誤差值都在 5%以內。

5. 結論

本研究提出了一個基於 Paired Sampling 方法，能夠在多執行緒的環境下估計一個 OSGi Service 執行時所需消耗的能源，同時解決了目前許多研究所提出的方法需倚賴特定硬體元件資訊、而無法於多執行緒的環境下準確地估計執行程式所需消耗的能源的缺點。除此之外，本研究所提出的 Decision Model 能夠估計遠端執行一個 OSGi Service 所需消耗的能源，並動態地決定是否要將某一個 OSGi Service Offload 到遠端的手機上執行以節省能源。最後於實驗結果中驗證了我們所提出的方法最多能節省 43.67%的能源，且無論是估計本地端執行 OSGi Service 所需消耗的能源、或是估計遠端執行 OSGi Service 所需消耗的能源，其能源估計的誤差值皆小於 4%。

致　謝

感謝科技部編號 MOST 103-2221-E-006-218 之計畫對本研究之經費提供與技術支援，由於科技部的支持，使本研究得以順利進行，特此致上感謝之意。

參考文獻

[1] M. Satyanarayanan, "Pervasive computing: vision and challenges," *IEEE Personal Communications*, Volume 8, Issue 4, pp. 10-17, 2001.

[2] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, Hen-I Yang, "The case for cyber foraging," *EW 10 Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pp. 87-92, 2002.

[3] OSGi, https://www.osgi.org/

[4] E. Cuervo, A. Balasubramanian, Dae-ki Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 49-62, 2010.

[5] C. Shi, V. Lakafosis, M. H. Ammar, E. W. Zegura, "Serendipity: Enabling Remote Computing among Intermittently Connected Mobile Devices," *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*, pp. 145-154, 2012.

[6] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic Resource Allocation and Parallel Execution in the Cloud for Mobile Code Offloading," *INFOCOM, 2012 Proceedings IEEE*, pp. 945-953, 2012.

[7] F. Xia, F. Ding, J. Li, X. Kong, L. T. Yang, J. Ma, "Phone2Cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing," *Information Systems Frontiers, Volume 16, Issue 1*, pp. 95-111, 2014.

[8] Y. D. Lin, E. T. H. Chu, Y. C. Lai, T. J. Huang, "Time-and-Energy-Aware Computation Offloading in Handheld Devices to Coprocessors and Clouds," *IEEE Systems Journal*, Volume 9, Issue 2, pp. 393-405, 2015.

[9] F. A. Ali, P. Simoensa, T. Verbelena, P. Demeestera, B. Dhoedta, "Mobile device power models for energy efficient dynamic offloading at runtime," *Journal of Systems and Software, Volume 13*, pp. 173-187, 2016.

[10] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, Y. Wang, "Fine-grained power modeling for smartphones using system call tracing," *Proceedings of the sixth conference on Computer systems*, pp. 153-168, 2011.

[11] Monsoon Solutions Inc. http://www.msoon.com

[12] T. Verbelen, P. Simoens, F. D. Turck, B. Dhoedta, "AIOLOS: Middleware for improving mobile application performance through cyber foraging," *Journal of Systems and Software, Volume 85, Issue 11*, pp. 2629-2639, 2012.

[13] L. Zhang, B. Tiwana, R. P. Dick, Z. Qian, Z. M. Mao, Z. Wang, and L. Yang, "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones," *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 105-114, 2010.

[14] M. Akram and A. ElNahas, "Energy-Aware Offloading Technique for Mobile Cloud Computing," *3rd International Conference on Future Internet of Things and Cloud*, pp. 349-356, 2015.

[15] F. Ding, F. Xia, W. Zhang, X. Zhao, C. Ma, "Monitoring energy consumption of smartphones," *Proceedings of the 2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*, pp. 610-613, 2011.

[16] K. Kumar, J. Liu, Y. Lu, B. Bhargava, "A Survey of Computation Offloading for Mobile Systems," Mobile Networks and Applications, Volume 18, Issue 1, pp. 129-140, 2013.

[17] Android BatteryManager API, http://developer.android.com/reference/android/os/BatteryManager.html

[18] W. Jung, C. Kang, C. Yoon, D. Kim, H. Cha, "DevScope: a nonintrusive and online power analysis tool for smartphone hardware components," *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 353-362, 2012.