耗時與耗電量測工具：比較與實例分析

余尚哲 林盈達

國立交通大學資訊科學系

October 15, 2010

摘要

使用耗時與耗電量測工具，來完整量測待測目標的耗時與耗電資資訊，是目前軟硬體量測的首要需求。測量電腦系統所使用的工具，因其測量目的不盡然相同，在測量結果上也會呈現出針對系統上不同範圍、不同層級深度的差異。因此我們需要藉由比較這些不同測量工具的功能，來了解各個測量工具在測量目標上的差異，也方便未來在遇到測量需求時，可以正確的分析出需要使用哪些測量工具較為適合。

本文將討論量測工具時所需要考量的一般性要素、與在耗時與耗電測量上，依照性質需要特別考量的要素，根據這些要素對數項耗時與耗電量測工具列表比較，並據此選出推薦使用的耗時量測工具首選OPrifile，與耗電量測工具首選NIDAQ。最後，我們做一實例測量，以所推薦的耗時量測工具OPrifile測量Firefox瀏覽器內四種不同渲染技術Canvas[1]、Flash[2]、HTML[3]、SVG[4]的執行時間，並分析與討論測量結果。在實例中發現SVG技術效能最慢，而其效能瓶頸是使用到了執行時間極長的外部函式庫libgdk-x11.so，並提出造成此瓶頸的可能原因。

關鍵字：耗時, 耗電, 量測, OPrifile, NIDAQ, Firefox, Rendering

到底可以量多細？

在比較量測工具時有幾個主要考量點，就一般性的論述為：各項工具所支援的作業系統、工具測量的對象空間、工具可測量的最小層級、以及測量該最小層級的所需條件。在接下來的兩小節中將定義並表列出上述四項考量點並加以比較。

因希望在比較後能夠選出量的又細又廣的量測工具：所能量測的層次越小越好，所能量測的系統空間、支援的作業系統越多越好；故比較不同工具的優劣時將優先以最小可測量層級與為主要考量，其次再比較其他各項考量點，並以可測

量的對象空間、支援的作業系統較多者為佳，最後選出理想的量測工具首選。

耗時量測工具

耗時量測的測量結果是在系統上執行一項操作或一系列操作時所花的時間，如：測量開機時間、開啟瀏覽器等待時間、瀏覽網頁時換頁等待時間等，若將上述操作切分成多個測量基本項目，則可用耗時量測工具對各個基本項目加以測量，得出個別項目執行所花時間後，總和即為目標操作的執行時間。

表一列出各項耗時測量工具能測量的最小量測層級、量測的對象空間、可測量的目標作業系統以及最小測量層級之需求。其中測量層級依工具所量測基本項目所在的執行層次由大到小區分為：測量單一程式執行時間(Process)、測量單一函式執行時間 (Function)或測量單一句程式碼執行時間(Statement)；最小測量層級即是指該工具所能測量到的最小層級。工具能夠量測的對象空間分為使用者程式執行空間(user space)與系統核心執行空間(kernel space)兩塊。

工具名稱	最小測量層級	量測空間	可量測作業系統	最小測量層級之需求
OProfile	Statement	K,U	Linux	Debugging info
SystemTap	Statement	K,U	Linux	Debugging info Need to write custom script Need to specify source line
TAU	Statement	U	Linux,Windows	Auto/Manually instrumentation
sysprof	Function	K,U	Linux	Debugging info
LTTNG	Function	K,U	Linux	K: Debugging info U: Manually instrumentation
KFT	Function	K	Linux	Patch kernel
gprof	Function	U	Linux	Debugging info
Android traceview	Function	U	Linux,Windows	Manually instrumentation

K:kernel space U:user space

表一 耗時量測工具與最小測量層級

耗電量測工具

耗電量測工具所測量的數據類型主要為測量裝置耗電量及測量使用電池可攜式裝置的運行時間，例如：EnergyBench測量裝置內CPU執行指令的耗電量、MobileMark 2007測量筆記型電腦在執行一般操作時的電池運行時間。

表二列出各項耗時測量工具能測量的最小量測層級、可測量的目標作業系統、以及最小測量層級的需求。在測量層級上則可分類為四種層級，以系統運行架構來看由大至小分別為：測量全系統耗電(Whole system)、測量單一程式執行

耗電(Process)、測量單一函示耗電(Function)、以及測量CPU指令耗電(CPU instruction)。

Common tools	最小測量層級	可量測作業系統	最小測量層級需求
EnergyBench	CPU instruction	Any(hardware tool)	Need to connect hardware power pin
PowerTop	Process	Linux	Need Linux kernel 2.6.21
Android Battery Use	Process	Android	Need Android system
NIDAQ	Whole system	Any(hardware tool)	Need to write analysis program Manually Instrument code
SPECpower_ssj2008	Whole system	ALL	Need java virtual machine environment
Battery Life Toolkit	Whole system	Linux	
EEcoMark	Whole system	Windows	
MobileMark 2007	Whole system	Windows	
Battery Eater	Whole system	Windows	
Nokia Energy Profiler	Whole system	Nokia S60	

表二 耗電量測工具與最小測量層級

耗時量測工具首選: OProfile

OProfile在耗時量測工具中是量測層級最細的幾個工具之一，使用的測量方式是抽樣(sampling)取得資料，雖然缺點是使用時會為系統帶來負荷(抽樣時使用預設頻率會造成4%的系統負荷)，但在量測空間上可以量測user space和kernel space的執行耗時情況；執行前準備上也只需要量測目標的除錯資訊就可以進行測量，除錯資訊的準備很方便：對於open source開發的程式，在Debian/Ubuntu環境中通常都可以找到該程式對應的除錯資訊套件包(原始名稱後附加"-dbg", "-dbgsym"字尾)，安裝該套件後即可進行量測；如要量測Linux系統核心也可以在網路找到產生核心除錯資訊檔案(vmlinux)的步驟。

綜合以上條件，OProfile可以讓開發者可以在簡短的準備後開始測量，並且在初期對目標程式的耗時有全系統廣度(user+kernel space)的了解，後期則可調整測量層級細到Statement，了解程式細部的運作情形，因此推薦使用OProfile做為耗時測量的首選工具。

耗電量測工具首選: NIDAQ

NIDAQ是美國國家儀器公司(National Instruments)的資料擷取(Data acquisition)工具。雖然NIDAQ使用硬體儀器測量搭配軟體接收分析數據，會需要使用LabVIEW軟體撰寫自訂的數據分析程式處理測量結果，但也因此提供較高

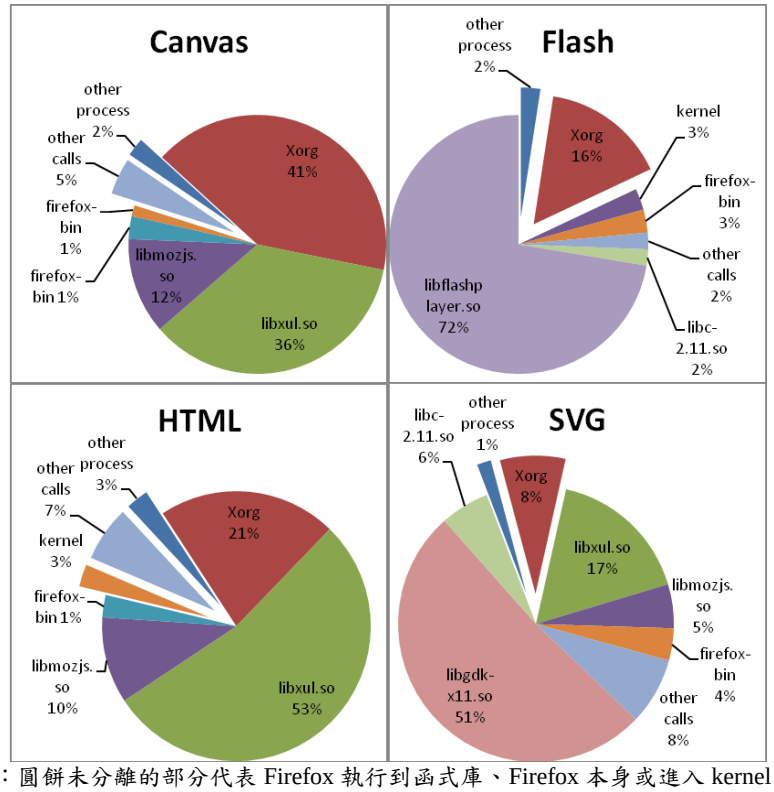
的客製化測量程度與較好的測量精度；反觀EnergyBench、PowerTop、Android Battery Use三款工具雖然測量層次比NIDAQ細，但皆不適用在一般性的耗電數據測量上：EnergyBench是專注在極細CPU指令上的耗電，owerTop與Android Battery Use則僅提供固定格式的耗電資訊報告，並無可供客製化測量的選擇。NI提供了LabVIEW軟體作為配合，讓撰寫客製化數據分析程式的難度大為降低，透過自己做的分析程式也更能夠取得符合期望的耗電測量數據。NIDAQ使用硬體測量儀器測量裝置的實際耗電量，也因此可以測量到全系統廣度(user+kernel space)的耗電資訊。因為NIDAQ支援多種作業系統，所以可以在各種環境中使用此工具，增加測量的便利性。

綜合以上所述，NIDAQ有較好的測量精度、客製化的測量細度與廣度、以及使用上跨平台的便利性，因此推薦使用NIDAQ做為耗電測量的首選工具。

實例：四種瀏覽器動態渲染效能分析

Mozilla Firefox瀏覽器是一個開放原始碼的瀏覽器，在Mozilla Firefox中可以使用Canvas[1]、Flash[2]、HTML[3]、SVG[4]四種渲染方式生成影像。本實例將使用Ubuntu 10.04作業系統，在Mozilla Firefox 3.6.11中分別使用上述四種渲染方式撥放四個網頁動畫[5]；這些頁面使用了相同的演算法，計算畫面上多個彈跳圓點的渲染資訊，但在產生影像時呼叫了不同的渲染引擎。本例使用耗時量測工具OProfile測量四次動畫撥放的耗時資料，並輸出Function層級的執行資訊後。

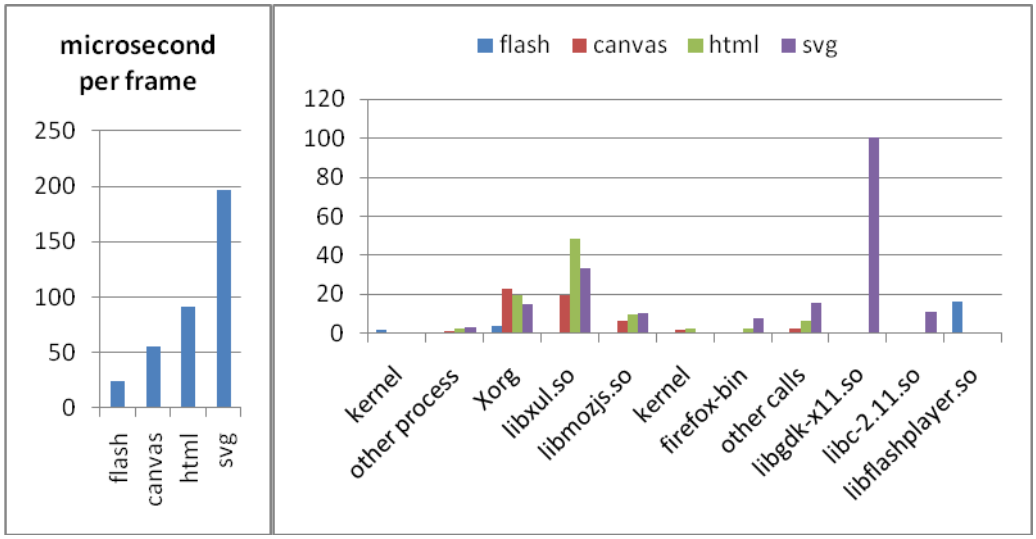
有了耗時資料後，我們利用其中提供的各函式所屬檔案資訊，依據這些檔案的檔名，合併同檔案內所屬函式的執行時間，即可知道四種渲染方式使用各類函式庫的時間比例，並使用此項資料製作出圖一。



圖說明：圓餅未分離的部分代表 Firefox 執行到函式庫、Firefox 本身或進入 kernel mode，分離的部分代表其他執行中的 process，或作業系統執行進入 kernel mode。

圖一 四種渲染方式(Canvas, Flash, HTML, SVG)下各函式庫使用時間比例圖

將四次撥放動畫取得的畫面更新率(frame rate)資料換算成每畫面毫秒數(microsecond per frame)，配合圖一的函式庫使用比例即可計算得到四種渲染方式渲染每幅畫面時各項函式庫的平均執行時間，製成圖二：



圖二 四種渲染技術每畫面渲染時間(ms)與使用動態函式庫時間(ms)

觀察圖一、圖二可以得到以下幾項資訊：

一、Canvas與HTML的函式庫比例是較為相似的，兩者都使用約10%的時間

處理Javascript程式碼(libmozjs.so[1])、約75%的時間在Xorg[2]與libxul.so[3]渲染；但在渲染的細部時間比重上有所差異，Canvas使用較少Mozilla的Gecko[4]/XUL[5]技術做渲染(函式在libxul.so內)，而使用多一倍的時間在Xorg－Linux視窗顯示的基礎程式上做渲染，所以Canvas效能會比HTML好。

二、SVG渲染技術主要使用GDK[6]函式庫(libgdk-x11.so)實現，但可能是使用的不好，或是GDK本身效能就較低，在圖二中GDK函式庫的執行時間遠超過其他函式庫，也連帶使的SVG的每畫面渲染時間是最長的。相比之下Flash主要的渲染函式庫libflashplayer.so使用比率更高，但在圖二中此函式庫的執行時間卻相當短，顯示libflashplayer.so函式庫渲染畫面較有效率。

結論

今日多種電腦應用技術百家爭鳴，程式與其他程式或動態連結函式庫互動的次數大幅增加，互動越緊密其界線也越模糊，測量時需要考慮的測試範圍也越加龐大複雜。因此要正確理解單一程式使用了多少CPU資源或電力，選出合適的量測工具是有關鍵影響的。例如本文實例中發現了SVG渲染的效能瓶頸是在程式外部的GDK函式庫，若只著重於改善SVG本身效能，對整體的效能提升是事倍功半的。本文選出的OProfile與NIDAQ都是可以具有足夠測量細度，且支援大範圍綜合量測統計的工具，適合作為目前電腦環境下的首選量測工具。

參考資料

[1] Canvas, https://developer.mozilla.org/en/HTML/Canvas, http://en.wikipedia.org/wiki/Canvas_element

[2] Adobe flash player, http://www.adobe.com/products/flashplayer/

[3] HTML, http://en.wikipedia.org/wiki/HTML

[4] SVG, https://developer.mozilla.org/en/SVG

[5] "HTML5" versus Flash: Animation Benchmarking, http://themaninblue.com/writing/perspective/2010/03/22/

---

[1] libmozjs.so: Mozilla 的 JavaScript 引擎函式庫

[2] Xorg: X Window 系统的實做，一種以點陣圖方式顯示的軟體視窗系統。

[3] libxul.so: 提供基於 Gecko/XUL 技術介面渲染 API 的函式庫。

[4] Gecko: Mozilla 開發的跨平台渲染引擎。

[5] XUL: 基於 XML、跨平台的使用者介面定義技術，並且使用 Gecko 引擎實做定義的介面渲染。

[6] GDK: GIMP Drawing Kit，包裝低層繪圖函式的繪圖函式庫，屬性介於 X server 和 GTK+間