基於時間預估之雲端算圖系統

范登凱　蔡建麟　許家彰　袁嘉彬
中華電信研究院-雲端運算所
tengkaifan@cht.com.tw (gmail.com)　bentsai@cht.com.tw
kennyshiu@cht.com.tw　dandyyuan@cht.com.tw

摘要

算圖 (rendering) 在動畫製作過程中 (pipeline) 係最耗費硬體資源之步驟。倘若沒有大量的運算資源，動畫師 (artist) 需等上許久時間才能得到算圖結果。此外大量的算圖任務，往往對於算圖時間的預估較難掌控。本論文主要提出一套基於算圖時間預估之智慧型雲端算圖系統。本系統會依據使用者設定的期望結果時間，計算出符合此算圖任務相對應的節點數量。藉由運算資源的配置，期望可在限定時間內完成算圖任務，以達到高的服務品質 (Quality of Service)。此外本系統主要建置在雲端虛擬環境，方便節點的動態增減。實驗結果顯示，本論文所提出的算圖時間預估方法，可精確地配置算圖資源，並於期望結果時間內完成算圖任務。

關鍵詞：雲端服務、算圖系統、算圖預估、資源配置、虛擬機器。

Abstract

Rendering is the most time-consuming process in the graphics pipeline. Generally, the artists need to wait for a long time to get the rendering results if there are not a lot of computing resources. Besides, among a large number of rendering jobs, the tasks of rendering time estimations are difficult to predict. When artists submit jobs to render, they would be more willing to assign the expected time of getting results. In other words, the artists can be promised to obtain rendering results after waiting for a period of time. Thus, in this paper, we propose a time-based estimation method of cloud rendering system. Our proposed rendering system based on users' expected time of getting outcomes to calculate the requirement of rendering nodes. By rendering resources allocation to provide QoS (Quality of Service), our system ensures that users can obtain the rendering result within a limited time. Furthermore, our rendering system is built on cloud virtualization environments which facilitate to dynamic scale in & scale out of compute nodes. We experimentally validate our approach using a set of real rendering materials. The results indicate that our proposed rendering system could effectively estimate the rendering time, allocate the corresponding rendering resources, lastly complete render job within in users' expected time.

Keywords: Software as a Service (SaaS), Rendering System, Rendering Estimation, Resource Allocation, Virtual Machine.

1. 前言

算圖 (Rendering) [5] 在電腦繪圖領域，泛指利用軟體從模型生成影像的過程。模型 (Model) 是用嚴格定義的語言或者資料結構對於三維物體的描述，它包括幾何 (geometry)、視點 (viewpoint)、紋理 (texture) 以及照明 (lighting)…等資訊。算圖這個術語類似於「動畫師 (artist) 對於場景的渲染」。另外算圖也常用於描述：計算視訊檔案中的效果，以產生最終視訊輸出的過程 [1]。算圖是 3D 圖學中最重要的研究課題之一，在圖形流體中，算圖是最後一項重要步驟，透過它得到模型與動畫最終顯示效果。隨著電腦圖形的不斷複雜化，算圖也越來越成為一項重要的技術。

算圖的應用領域有：電腦與視訊遊戲、模擬、電影或者電視特效以及視覺化設計。算圖主要可分為預先算圖 (pre-rendering 或 offline rendering) 與及時算圖 (real-time rendering 或 online rendering)。前者常用於電影製作、電視特效；後者常應用於即時遊戲，其中又以預先算圖所需資源較多 [1]。例如皮克斯動畫電影“玩具總動員”使用一台工作站計算一張圖片耗費超過三個小時。為提升算圖效率，皮克斯公司建置算圖農場 (render farm) 進行平行運算。最後總共計算 114,000 張圖 (共 77 分鐘)，仍需耗費兩年時間。再以更高品質動畫電影“魔戒-雙城奇謀”為例，算圖總時數超過 400 萬小時 [6]。

除了透過大量運算資源加速算圖效率外，算圖時間預估往往亦是動畫師所關心的議題。例如：某專案時程較急迫，動畫師希望可以在最短時間內得到結果。倘若當動畫師提交算圖素材後，能夠自行設定期望結果時間，勢必更能夠精確地掌握專案進度與品質。

雲端運算 (Cloud Computing) 近幾年的興起，也帶動了資訊科技產業另一波風潮。它是一種基於網際網路的運算方式，特色主要包含有快速動態擴充 (scale in, scale out)、虛擬化資源 (virtualization)、監控與量測服務…等。服務模式可分為軟體即服務 (SaaS, Service as a Service)、平台即服務 (PaaS, Platform as a Service)、及基礎架構即服務 (IaaS, Infrastructure as a Service)。部署模型可區為成：公用雲 (Public Cloud)、私有雲 (Private Cloud)、社群雲 (Community Cloud、及混合雲 (Hybrid Cloud) [11]。

本論文提出一套基於算圖時間預估方法之雲端算圖系統，主要應用於預先算圖。算圖系統會依據使用者輸入的期望結果時間，來計算出此算圖任務所需配置的相對應資源 (本論文指運算節點數量)。此外本系統主要建置在雲端虛擬環境，可享有動態擴充資源與集中式監控之特性。暫定以公有雲模式部署本系統，方便使用者透過網路即可取得雲端算圖服務。實驗結果顯示，本論文提出以算圖時間預估方法之雲端算圖系統，可精確地計算出期望結果時間內所需運算資源，進而有效地分派與完成算圖任務，以達到高的服務品質 (Quality of Service)。

本論文的編排如下：第二章介紹相關文獻，主要探討目前關於算圖系統建置的方法與比較；我們提出的雲端算圖系統將呈獻於第三章。第四章將透過實驗來證明本論文方法之有效性。結論與未來研究方向被放置於最後一章。

2. 文獻探討

本章我們首先花一點篇幅介紹製作視覺動畫的基本流程 (Visual Effects Pipeline, VFX Pipeline)；接著探討算圖系統相關研究。

2.1 Visual Effects Pipeline Overview

根據 [2] 指出 3D 動畫製作可以分為三大步驟：(1) 前置作業 (Pre-Production). (2) 動畫製作 (Production). (3) 後置作業 (Post-Production). 其中 Pre-Production 主要討論動畫的構想與腳本 (script)，必要時可透過簡單的圖像 (storyboard) 加以輔助。Production 階段為動畫師花費最多時間與消耗最多運算資源的過程，可細分為： (*i*) 分鏡構圖 (Layout), (*ii*) 模型製作 (Modeling), (*iii*) 貼圖材料製作 (Texturing), (*iv*) 骨架設定 (Rigging), (*v*) 動做設計 (Animation), (*vi*) 特效 (Effects), (*vii*) 燈光 (Lighting), (*viii*) 算圖 (Rendering). 最後 Post-Production 階段負責合成 Production 程序產生的 2D 或 3D 圖像，待進行顏色校正後，正式輸出成視覺動畫。

2.2 Cloud-based Rendering System & Grid-based Rendering System

2006~2009 期間，分散式算圖系統大多以網格方式 (Grid-based) 建置 [3][4][12]。Chong [4] 等人利用開放原始碼 (Open Source Software) 建置網格式算圖系統。有鑒於算圖素材的檔案容量會影響傳輸效率，文中亦提出資料壓縮技術，將使用者檔案進行壓縮後傳輸，以節省網路頻寬。此外作者將平行算圖區分為兩類：(1) networking rendering: 圖片集可被平行計算，算圖任務彼此間是獨立的。換言之，算圖任務具有基元性 (atomic) 最小單為 frame。(2) distributed rendering: 單一圖片可以依解析度 (resolution) 細分成若干單位進行算圖。本論文提出之算圖服務係以 frame 為最小算圖單位，屬於 networking rendering system。Ijaz [10] 等人同樣為降低網路流量，亦提出資料壓縮方法。文中採用 MPEG-4 Part 25 (MP25) 技術，針對三種特定格式 (i.e., COLLADA, X3D, and XMT) 之 3D 物件進行壓縮。

在 [3] 研究中，Carpegna 等人提出的網格式算圖系統具有網頁介面，因此在實作上較強調安全機制。文中提到除了採用網路安全協定(https) 外，在中介軟體 (middleware) 部分，認證均透過 GSI (Globus Security Infrastructure)。GSI 管理存取與認證使用以 X.509 與 SSL (Secure Socket Layer Protocol) 為基準的公鑰機制。Glez-Morcillo 等人 [8] 結合 P2P 與網格技術，提出 P2P-based 網格算圖系統。透過 P2P broker 管理檔案交換程序，每個 P2P broker 會經由 Session Manager 來交換資訊。一旦某個算圖節點透過 P2P 方法取得所需算圖素材後，將啟動算圖程序。本論文提出算圖系統具有網頁操作介面，基於安全性考量同樣採用 GSI 安全機制與 https 通訊協定。

雲端近幾年的興起，許多應用系統紛紛建置在雲端主機上。Zhou [14] 等人利用超級電腦 (super computer) 打造雲端算圖系統，稱為 Golden Farm Cluster Rendering Platform。此平台建置 2000 算圖節點，共有 24,000 CPU cores. 本論文和 [14] 最大不同處在我們提出的雲端算圖系統係建置在虛擬機器上，可享有雲端既有的特性。並以公用雲模式部署，具有網頁圖型化操作介面。反觀 Golden Farm Cluster Rendering Platform 係建立在實體主機上，以應用程式方式部署供終端使用者操作。

Rajendran [13] 探討雲端服務應用於動畫領域之益處。動畫公司除可以取得大量運算資源外，對於量化分析 (例如：Return On Investment, Service Level Agreement) 也較能掌握。文章最後利用兩個個案來探討動畫公司引進雲端服務。(1) 夢工廠 (DreamWorks Studios) 和雲端服務提供者 (Cerelink) 簽數年合約。夢工廠利用雲端資源製作出 3D 電影 “史瑞克 3D”。(2) Crest Animation Studio 採用 Computational Research Laboratories (CRL) 的 “EKA” 雲端系統，製作出印度第一部 3D 電影 “Alpha and Omega”。數據指出該公司藉由雲端資源來進行算圖，可減少超過 50% 以上算圖時間。

許多研究 [7][9] 探討網格運算與雲端運算間的差異。除了資源部署方式 (雲端運算強調虛擬化) 相異外，雲端運算係以服務導向 (services oriented) 偏向集中化管理；網格運算則屬應用導向 (application oriented) 偏向分散式管理。如圖 1 所示 [7]。無論是網格式算圖服務或雲端式算圖服務，大多數的研究均指向如何提供分散式平行運算及賦予大量運算資源。顯少有研究討論如何有效地預估算圖時間。本論文除了將系統部署於雲端虛擬化環境外，並提出算圖時間預估之方法。

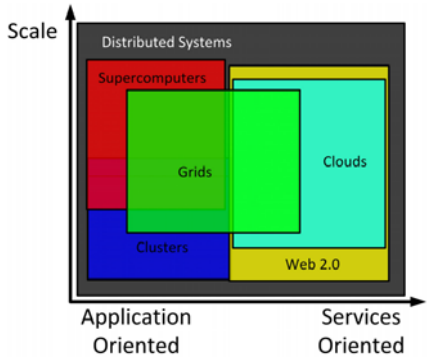

圖 1 網格與雲端概觀

3. 智慧型雲端算圖系統

本論文提出的智慧型雲端算圖系統主要聚焦在如何於使用者限定算圖時間內，提供算圖結果。因此，給定一個算圖任務 (通常包含一張以上圖片素材)，包括起始圖片編號、結束圖片編號、間隔距離，及期望結果時間四個基本參數。透過我們提出的預估方法可計算出此素材所需基本算圖總時間，進而配置相對應算圖機器，來達到使用者預期目標。本章節首先介紹整體雲端算圖系統架構，接著將著重在算圖任務資源配置模組的詳細步驟。

3.1 系統架構概述

智慧型雲端算圖系統架構可分為五個階層：資訊基礎、算圖服務、算圖引擎、服務管理及存取介面，如圖 2 所示。

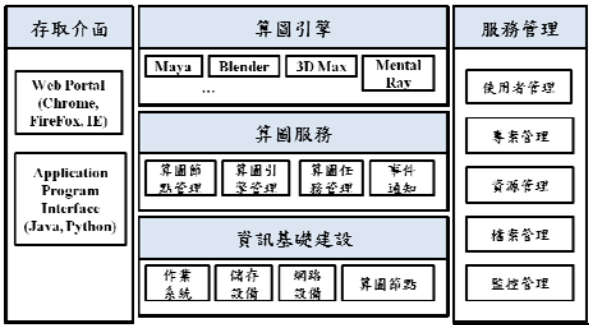

圖 2 智慧型雲端算圖系統架構

(i) 資訊基礎建設主要提供硬體資源及作業系統。儲存設備負責檔案儲存服務，用以儲存算圖素材及算圖系統產生出來的結果。算圖節點提供實際算圖資源。系統間的通訊溝通主要係透過網路設備來支援。資訊基礎建設全部均部署在雲端虛擬化服務環境。

(ii) 算圖服務主要掌管算圖任務，提供提交/啟動/暫停/回復/刪除任務…等基本操作。當此算圖任務所需的算圖引擎 (例如: Maya) 需要有許可證 (License) 時，亦可透過算圖引擎管理模組進行許可證調度。最後會依據算圖節點管理模組提供的資源有效性，將此算圖任務分派給閒置狀態之算圖資源。算圖任務結束或失敗，則會透過事件通知模組發佈訊息。

(iii) 算圖引擎：為因應不同的動畫公司採用多樣化的動畫軟體。本系統提供許多較常被使用的算圖引擎。例如：Maya, Blender 及 Mental Ray. 未來將支援其他更多算圖軟體。

(iv) 服務管理提供許多背景的相關服務與存取服務。此階層包括使用者管理、專案管理、資源管理、檔案管理及監控管理。

(v) 存取介面：使用者除了可透過網頁操作本系統外，亦可透過 API 來和公司內部系統進行橋接。目前支援 Java、Python，未來將支援更多高階語言。

系統實作部分：Web Portal 以 JavaScript (i.e., ExtJS[1]) 開發、服務管理層係以 Java 實作、算圖引擎介接採用 Python 語言；算圖服務層主要是透過 C++ 實作。此外資料庫與網頁伺服器分別以 MySQL 6.0 與 Tomcat 7.0 為主。

我們透過簡單的基本流程圖，來說明使用者操作情境。首先使用者必須透過帳號/密碼登入本系統；接著上傳算圖素材，上傳期間可觀看上傳進度，並可進行基本操作，例如：暫停/刪除/回復。產生算圖任務時預設為先前上傳的素材，並設定相關參數，包括起始圖片編號、結束圖片編號、間隔距離、期望結果時間。其中間隔距離參數表示圖片間的距離，例如圖片編號：1, 4, 7，間隔距離為 3。期望結果時間為使用者希望多久之後可以獲得算圖結果；例如：1 hr, 5 hrs, 12 hrs, 1 day, 或指某個特定時間點 “2013/07/31 14:00:00”。最後提交此算圖任務。當算圖服務主機接收到任務時，會進行時程預估並取得相對應算圖資源 (詳如 3.2)。最後分派此算圖任務至各節點，並驅動每個節點上的算圖引擎實際算圖。算圖過程中，使用者亦可透過網頁來監看算圖狀況。

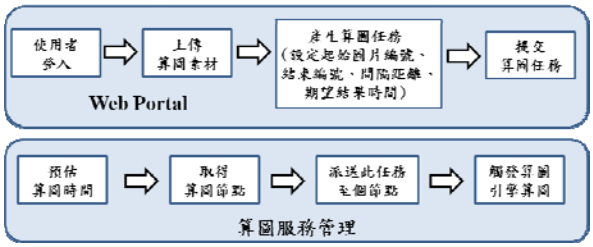

圖 3 使用者操作情境流程圖

3.2 算圖時間預估與資源配置

本系統主要透過算圖時間預估，來配置適當數量的算圖資源。本論文提出一套直覺式方法來預估算圖所需時間。當算圖服務管理模組接收到算圖任務 *Job* 時，會依據使用者輸入的圖片起始編號 f_s、結束編號 f_e 、及間隔距離 $f_{interval}$，計算出總圖片數量 f_{total} (透過 $\frac{(f_e - f_s)}{f_{interval}}$)。同時會將此素材第一張圖片 f_{first} (i.e., f_s) 指派給某台算圖節點 n_i 進行算圖，可得到 n_i 運算 f_{first} 所需花費的時間 t_{first} 。最

[1] http://www.sencha.com/products/extjs

後透過 $t_{first} \times f_{total}$ 來估計整體算圖時間 t_{total}。t_{total} 為單一節點所需花費的預估時間，可以拿此時間跟使用者期望時間 $t_{expected}$ 做比較。倘若 $t_{total} \leq t_{expected}$，則算圖服務模組僅需配置一台機器；否則本系統會依據 t_{total} 與 $t_{expected}$ 間的比例，來配置相對應的節點數量。圖 4 為算圖任務資源配置模組的虛擬碼。

Algorithm: Resource allocation strategies
Input: A render job j_i with material and some user's parameters including start frame f_s, end frame f_e, frame interval $f_{interval}$, and expected time $t_{expected}$.
Output: the amount of rendering nodes N.
1. Calculate f_{total} by $\frac{(f_e - f_s)}{f_{interval}}$;
2. Calculate t_{first} by using one render node to render f_s;
3. If (f_{total} = = 1) return 0; the j_i is completed.
4. Calculate t_{total} by $t_{first} \times (f_{total} - 1)$; //1st frame has been rendered.
5. int N = 0; //節點資源數量
6. $N = \text{Ceiling}\left(\frac{t_{total}}{t_{expected}}\right)$; // 確保 $N \geq 1$
7. Return N.

圖 4 算圖時間預估虛擬碼

本系統會透過上述演算法取得此算圖任務之節點數量後，將此算圖任務放置任務串列中 Job_{List}。Job_{List} 依據使用者提交算圖任務當下時間作排序。算圖任務分派執行緒 (Job Dispatcher) 會依序消費 Job_{List} 中的 job。當發現此任務所需要的機器大於算圖池的數量 (render pool) 時，會呼叫自動供裝程序 (auto-provision procedure)，來取得足夠的算圖節點。最後將此算圖任務中的圖片均勻地指派至算圖節點進行算圖。相對應虛擬碼如下圖所示。

Algorithm: Dispatch job to render node(s).
Input: Sorted Job_{List}.
Output: Dispatching job.
1. while (true) {
2. Get the j_i from Job_{List}.
3. Get the figure of required render nodes $\boldsymbol{N}$ from j_i;
4. Detect whether the render pool $\boldsymbol{P}$ has enough render nodes:
   - if true, get the render nodes set $\boldsymbol{N}$ directly.
   - else, get the idle render node $\boldsymbol{N}_{\text{idle}}$ from $\boldsymbol{P}$ and get $(\boldsymbol{N} - \boldsymbol{N}_{idle})$ by auto-provision procedure.
5. After getting $\boldsymbol{N}$, then dispatch j_i on $\boldsymbol{N}$ uniformly.
6. }

圖 5 運算資源取得與任務分派虛擬碼

事實上在 auto-provision procedure 所花費的時間 $t_{provision}$ 可以納入 $t_{expected}$，如此一來我們可以輕易地解決 $t_{provision}$ 所造成的時間誤差。例如 $t_{expected}$ 為 1 小時，$t_{provision}$ 為 10 分鐘，我們在算圖時間預估時可將 $t_{expected}$ 往前位移 10 分鐘，即 $t_{expected} = t_{expected} - t_{provision}$。

4. 實驗與探討

本章節主要探討兩個實驗：算圖時間預估與算圖節點配置實際運算。

4.1 算圖時間預估

實驗主要執行在 20 台算圖節點 (共 80 cores)，CPU 為 Intel Xeon CPU X7560 2.27 GHz 搭配 8 GB 記憶體。每個虛擬主機均安裝免費算圖引擎 Blender，並部署於 Linux (CentOS 6, 64 bits) 作業系統。算圖素材方面，我們共蒐集了 12 個實際動畫素材。為了證明我們透過單一圖片運算時間，來推估整個素材的整體計算之可行性。我們首先將全部動畫素材分派給算圖節點進行運算。結果顯示於表 1。

$\boldsymbol{t_{total}}$：計算所花費的整體時間。
$\boldsymbol{t_{avg}}$：平均計算一張圖所花費的時間。
$\boldsymbol{t_{first}}$：計算第一張圖片所花費的時間。
σ (sigma)：標準差常被機率統計使用，作為度量母體樣本間的離散程度。可透過下列公式取得：

$$\sigma = \sqrt{\frac{1}{F}\sum_{i=1}^{F}\left(t_i - t_{avg}\right)^2}$$

其中，$\boldsymbol{F}$ 為算圖素材的總圖片數、t_i 為第 i 張圖片所需的算圖時間。

$\boldsymbol{t_{|avg-first|}}$：$t_{avg}$ 與 t_{first} 之間的差值。

表 1 算圖時間統計資訊

No.	$\boldsymbol{F}$	t_{total} (d:hh:mm:ss)	t_{avg} (hh:mm:ss)	σ (mm)	t_{first} (hh:mm:ss)	$t_{\lvert avg-first\rvert}$ (mm:ss)
1	100	0:21:43:47	**00:13:10**	0.46	**00:13:26**	00:16
2	100	0:00:27:44	**00:00:17**	0.01	**00:00:17**	00:00
3	100	0:05:19:11	**00:03:14**	0.13	**00:03:39**	00:25
4	100	0:04:34:34	**00:02:46**	0.28	**00:02:17**	00:29
5	100	3:16:03:47	**00:53:22**	1.49	**00:52:30**	00:52
6	100	0:23:28:53	**00:14:14**	0.62	**00:15:02**	00:48
7	250	5:23:21:32	**00:34:16**	1.88	**00:34:23**	00:07
8	250	3:02:05:10	**00:17:47**	0.60	**00:18:01**	00:14
9	250	0:02:46:22	**00:00:40**	0.08	**00:00:34**	00:06
10	350	1:08:47:49	**00:07:54**	0.13	**00:07:48**	00:06
11	350	1:23:00:22	**00:11:30**	0.47	**00:11:34**	00:04
12	400	18:08:27:37	**01:06:04**	2.99	**01:08:03**	01:59

藉由表 1 結果明顯指出，$t_{|avg-first|}$ 差距非常小。最大值不超過 2 分鐘。此外 σ (標準差) 值也反應出該母體中，圖片所需計算時間之離散程度較穩定。例如：素材 9 的標準差和 $t_{|avg-first|}$ 為分別為 0.08 及 6 秒。然而 σ 與 $t_{|avg-first|}$ 最大值則出現在素材 12，我們透過下段篇幅來說明此現象。

我們利用皮爾斯相關係數 (Pearson Correlation) 來探討 σ 與 F 及 t_{avg} 之間是否具有相關性。由表 2 數據顯示，σ 和 t_{avg} 呈現高度正相關，相關係數高達 0.95；反觀 σ 和 F 僅呈現中(弱)相關，相關係數僅呈現在 0.40。因此，我們推論越長的平均算圖時間，其素材內的變異程度相對不穩定。此項發現對於日後校正系統有很大的幫助。最後，我們可根據 σ 與 $t_{|avg-first|}$ 統計值合理的推斷：一般算圖素材其每張圖所需的計算時間變異程度相對穩定。因此透過取樣出來的樣本 (本論文係依據使用者設定的起始圖片) 所產生的實際運算時間，來估計此素材的整體運算時間是具可行性的。

表 2 σ 和 F 及 t_{avg} 間相關係數

	F	t_{avg}
σ	0.40	**0.95**

雖然表 2 顯示 σ 與 F 呈現中弱相關。事實上未來或許有可能遇到非常大的算圖素材 (例如：超過 10,000 張圖片)，圖片中的 3D 物件可能不是呈現均勻分佈。換言之，每張圖片計算時間變異程度較大。此時我們可藉由其他抽樣方法 (例如：random sampling, systematic sampling, and cluster sampling) 來取得算圖時間預估基準。

4.2 基於時間預估算圖評估

本節主要評估本論文提出之方法，是否有效地藉由時間預估方法取得相對應運算資源後進行算圖，最終達到使用者預期需求。我們從蒐集的素材中隨機產生 600 個算圖任務。換言之，每個算圖任務所需參數 (i.e., 圖片起始編號、結束編號、間隔距離、及期望結果時間) 均不盡相同。此外將未採用時間預估方法視為比較對象 (baseline)。換句話說，當系統接收到算圖任務時，分派至算圖池中所有閒置機器，而不指派特定計算節點數量。為求公平性，我們固定採用 20 台算圖機器，而不透過 auto-provision procedure 來取得額外機器。評估度量衡則採用任務達成率 (即是否在使用者限定時間內完成算圖任務)。實驗結果如表 3 所示。其中 "# of failure jobs" 欄位表示未在使用者期望時間內完成任務數量。

表 3 基於時間預估算圖之評估

Methods	**# of jobs**	**# of failure jobs**	**Reaching rate (%)**
Baseline	600	33	94.5
Time-based	600	**0**	**100**

結果顯示我們提出的基於時間預估方法達成率為 100 %，優於 baseline 產生的 94.5 %。換言之，失敗任務數量相較之下，也低於 baseline 製造的 33。

雖然本論文提出的方法可以有效地在限定時間內完成算圖任務。我們進一步深入去探討每個任務的完成時間 $t_{completed}$ 與 $t_{expected}$ 間的差距。礙於篇幅我們僅取前六個差距最小的任務來說明，如表 4 所示。數據指出任務均在 $t_{expected}$ 之前 30 秒完成，其最大值為 0.483 分鐘；最小值為 0.166 分鐘。事實上 600 個算圖任務中，計有 18 個任務的完成時間與預期時間差距落入一分鐘內。一分鐘作為長時間算圖任務的緩衝時間略顯不足。我們未來可調整 $t_{expected}$ 時間軸，來克服此問題。例如使用者將 $t_{expected}$ 設定為 2 小時，系統接收此任務時，刻意地將 $t_{expected}$ 往前調整至 1 小時 50 分。此做法雖然配置了較多的計算節點數量，但卻可提供高品質服務。當然有更聰明的作法，根據 4.1 節實驗得知，t_{first} 越高表示算圖時間變異程度較高，因此我們可依據 t_{first} 大小來動態調整 $t_{expected}$ 位移量。此方法可避免每個算圖任務均配置相同緩衝區間。

表 4 $t_{completed}$ 與 $t_{expected}$ 之比較

No.	$t_{expected-completed}$ (mm)	rendering total time (mm)
1	**0.166**	40
2	0.216	57
3	0.266	65
4	0.283	73
5	0.400	73
6	**0.483**	49

4.3 討論

本論文提出基於時間預估之雲端算圖系統，目前仍然具有幾個限制條件，如下：

(1) $t_{expected} \geq (2 \times t_{first})$：使用者期望得到結果的時間必須至少大於等於 2 倍第一張算圖時間。然而有可能出現算圖素材運算量非常龐大，連計算一張圖片所需的時間都超過使用者預期的時間，即 $t_{first} > t_{expected}$。

(2) 運算節點均相同硬體規格：本實驗所採用的虛擬機器規格均相同。但未來勢必會添加不同規格的硬體資源。如此一來每台機器運算能力亦會隨著硬體規格不同而有所差異。

針對上述兩個限制，我們提出因應的對策。針對限制一，最直接的方法莫過於如何快速取得 t_{first}。這裡提出兩個概念做法：(*i*) 本論文算圖的最小單位為圖片 (frame)，也是一般動畫公司常用的基本單位。未來可以引入 distributed rendering (可參考第 2.2 節)，將某張圖片依據解析度進行分割後平行運算。此方法雖然運用較多的算圖資源，但是必定可大幅降低原本計算單一圖片所需時間。(*ii*) 無論是 networking rendering (圖片不可分割) 或是 distributed rendering 均需等待該圖片被計算完成後

才能得到運算時間 (i.e., t_{first})。事實上我們甚至可以預估 t_{first}。一般算圖引擎在計算某張圖片時，會輸出目前處理進度 (progress message)。然而本論文暫不採用 progress message 來估計的原因有: (a) 並非所有的算圖引擎均提供 progress message。(b) progress message 間的離散程度較難掌握。例如：計算 1% 時花費 1 分鐘;計算 2% 時花費 10 分鐘。

至於第二個限制式，未來我們可以透過校正的方式來將每台機器的運算能力做標準化。首先將某台機器視為是 benchmark m_{base}。不同規格的機器 m_{diff} 和 benchmark 計算相同的素材，得到兩個相異的運算時間。我們可將此算圖時間之比例，來當做是校正因子。簡單舉例： m_{base} 計算某素材需要 1 小時；而 m_{diff} 只花費 30 分鐘。我們可以推論機器 m_{diff} 的運算能力是 m_{base} 的 2 倍。此資訊可記錄起來，供分派算圖資源之用。透過圖 4 虛擬碼得到某算圖任務需要配置的節點數量後，在實際取得資源時 (可見圖 5) 可參考機器校正資訊，來取得等效能的運算資源。藉此來解決日後不同規格機器所產生的變因。

5. 結論與未來方向

本論文主要提出一套基於算圖時間預估方法之雲端算圖系統。我們提出的算圖系統會依據使用者輸入的期望算圖結果時間，來計算出此算圖任務所需配置的相對應資源。此外本系統主要建置在雲端虛擬環境，方便算圖節點的動態擴張。我們首先經由實驗來證明利用單張算圖時間來預估整體算圖時間是具可行性的。結果顯示，產生出來的算圖時間標準差平均落在 0.01 ~ 2.99。接下來我們進行實際算圖試驗。結果顯示，本論文提出的基於算圖時間預估方法之算圖系統，能精確地在期望結果時間內完成算圖任務。

未來我們計畫將朝向導入 distributed rendering 來強化算圖系統之效能，以解決嚴苛的期望時間議題。此外我們將整合更多的算圖引擎 (例如: Houdini, Cinema 4D, RenderMan…等)，並持續校正算圖時間預估演算法。

參考文獻

[1] https://zh.wikipedia.org/wiki/%E6%B8%B2%E6%9F%93

[2] A. Beane, 3D Animation Essentials, 1 ed., pp. 350: John Wiley & Sons, Inc., 2012.

[3] Y. Carpegna, M. Pissardo, B. Montrucchio, and A. Sanna, "A Grid Computing-Based Architecture for on Demand Movie Rendering," in 7th World Multiconference on Systemics, Cybernetics and Informatics: Technologies and Applications, Orlando, 2007, pp. 7-13.

[4] A. Chong, A. Sourin, and K. Levinski, "Grid-based computer animation rendering," in 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia, Malaysia, 2006, pp. 39-47.

[5] J. Dollner, and K. Hinrichs, "A Generic Rendering System," IEEE Transactions on Visualization and Computer Graphics, vol. 8, no. 2, pp. 99-118, June, 2002.

[6] G. Dolbier, and V. Megler, Building an Animation and Special Effects Studio from the Ground Up, IBM, 2005.

[7] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in Grid Computing Environments Workshop, Austin, 2005, pp. 1-10.

[8] C. Glez-Morcillo, D. Vallejo, J. Albusac, L. Jimenez, and J. Castro-Schez, "A New Approach to Grid Computing for Distributed Rendering," in P2P, Parallel, Grid, Cloud and Internet Computing, Barcelona, 2011, pp. 9-16.

[9] S. M. Hashemi, and A. K. Bardsiri, "Cloud Computing Vs. Grid Computing," APRN Journal of Systems and Software, vol. 2, no. 5, pp. 188-194, May, 2012.

[10] U. Ijaz, E. Quacchio, and D. Alfonso, "A Web-based Framework for Compressed 3D Objects: Downloading and Rendering," in 10th International Conference on Frontiers of Information Technology, Islamabad, 2012, pp. 129-133.

[11] P. Mell, and T. Grance, "The NIST Definition of Cloud Coomputing," 2011.

[12] W. Premchaiswadi, A. Tungkasthan, and N. Jongsawat, "Using Grid Computing for Rendering to Support 3d Animation Rraining Courses," in World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education Vancouver, Canada, 2009, pp. 1409-1418.

[13] L. Rajendran, "Investigating the Cloud Services in Animation Field," Internatinoal Journal of Engineering Science and Technology, vol. 4, no. 3, pp. 818-822, March, 2012.

[14] W. Zhou, Y. Lu, P. Gao, C. Qiu, and Q. Qi, "A New Software Architecture for Ultra-Large-scale Rendering Cloud," in 11th International Symposium on Distributed Computing and Applications to Business, Engineering & Science, Guilin, 2012, pp. 196-199.