

MapReduce and Bigtable

R96922078 王穎杰, R96922079 周錦彥, R96922108 黃禮俊

Abstract

根據調查顯示，Google 週一上午的使用量最大，且當搜尋速度變慢後，使用人數會驟減。Google 為了與其他業者競爭，在各項服務上，必須提供使用者可接受的速度。為了達到此種要求，Google 使用了許多技術。我們在此介紹 MapReduce 以及 Bigtable。

1. Introduction

隨著網路越來越普及，每個人開始對網路的服務越來越依賴，因此服務的品質對每個以網路服務為導向的企業是相當重要的。而在這分秒必爭的社會裡，影響網路服務品質的因素中，「速度」是最被重視的項目，我們以全球最大的網路搜尋引擎 Google 為例，根據 Google 內部統計過後，每個星期一上午是整個禮拜使用 Google 人數最高峰的時間，如果這時候使用者感覺到搜尋速度不符預期之後，使用人數會因此而驟降。由此可看出提升搜尋速度對 Google 來說是隨時隨地都需要關心的問題。許多人好奇 Google 是藉著什麼技術發展成如此龐大的企業呢？Google 一定有其獨到之處，而在眾多 Google 研發的技術當中，我們接下來將介紹 MapReduce 以及 Bigtable 這兩種有趣的技術。

2. MapReduce

2.1 MapReduce Programming Model

MapReduce 是一種 Programming Model，適合用來處理大量的 data。它的特點有四項，分別為 Automatic parallelization and distribution，Fault-tolerance，I/O scheduling，Status and monitoring。這四項特點會在之後的段落裡介紹敘述。

首先我們先介紹 Automatic parallelization and distribution，顧名思義，MapReduce 的整個架構是由 Map 以及 Reduce 兩個 function 所組成，使用者輸入一組 key/value，Map function 產生許多組 intermediate key/value，然後 reduce function 合併具有相同 key 的 intermediate pair，產生最後的結果。

Map function 的 prototype 如下：

```
map (in_key, in_value) → list(out_key, intermediate_value)
```

Reduce function 的 prototype 如下：

reduce (out_key, list(intermediate_value)) →list(out_value)

接下來我們利用 MapReduce 來實作一個例子，這個例子是要計算 input 裡面每個字出現的次數，我們列出自行定義的 Map 和 Reduce function 的 pseudo-code。

```
map(String key, String value):          reduce(String key, Iterator values):
// key: document name                  // key: a word
// value: document contents            // values: a list of counts
    for each word w in value:          int result = 0;
        EmitIntermediate(w, "1");      for each v in values:
                                        result += ParseInt(v);
                                        Emit(AsString(result))
```

Map function 對每一個字都會執行EmitIntermediate 這個function，其中argument “1” 表示出現一次，而reduce function再把具有相同key的value加總起來，即為這個字出現的次數。

由這個例子我們可看出，MapReduce可以把一個具有大量data的計算問題，經由Map function分散切割後，由reduce function合併成結果。以目前CMP架構相當盛行的狀況下來看，因此MapReduce不僅能夠充分利用CPU，也是能夠達到 scalability 的一種 programming model。

2.2. Execution overview

Map operation 會將輸入資料切割成許多個分割(spilt)，並分配給不同的機器，所以這些分割可以在不同的機器上平行處理；而 Reduce operation 則收集在不同機器上的運算後的結果，再合併成最後的結果。

詳細的執行流程如下：

1. 首先使用者程式 MapReduce 函式庫會將輸入資料切割成 M 個分割(spilt)，分割的大小通常是 16~64MB，之後會複製多個使用者程式在一個群集(cluster)的機器上產生準備執行工作。
2. 其中的一份複製是特別的，稱作 master，其他的複製稱作 worker，負責執行 master 分派的工作。在執行過程共有 M 個 Map operation 和 R 個 Reduce operation 要完成，其中 master 挑選閒置的 worker 來負責執行 Map 或 Reduce 的任務。

3. 被分派的 Map 任務的 worker，首先從相對應的分割(split)讀取資料，並從中分析出一組對應的 key/value，然後在傳送給使用者定義的 Map 函數處理。Map 函數產生的 intermediate key/value 會先暫存在記憶體中。
4. 暫存在記憶體裡的 intermediate key/value 會週期性地被寫入當地的磁碟，而被寫入的磁碟位置會被傳回給 master，master 在依據這些位置分派 Reduce 任務給 worker 執行。
5. 負責 Reduce 任務的 worker 會使用 RPC(remote procedure call)來讀取在當地磁碟中的暫存資料，而當讀取完所有的 intermediate key/value 後，worker 會對它做排序來把有相同 intermediate key 的 value 集合起來。
6. 負責 Reduce 任務的 worker 會對每個不同的 intermediate key/value 執行步驟 5.，然後把對應的 intermediate key/value 傳給使用者定義的 Reduce 函數。而 Reduce 函數會把此分割的結果附加在最終的結果裡。
7. 當所有的 Map 和 Reduce 任務完成後，master 會喚醒使用者程式，而使用者程式可繼續執行下去。

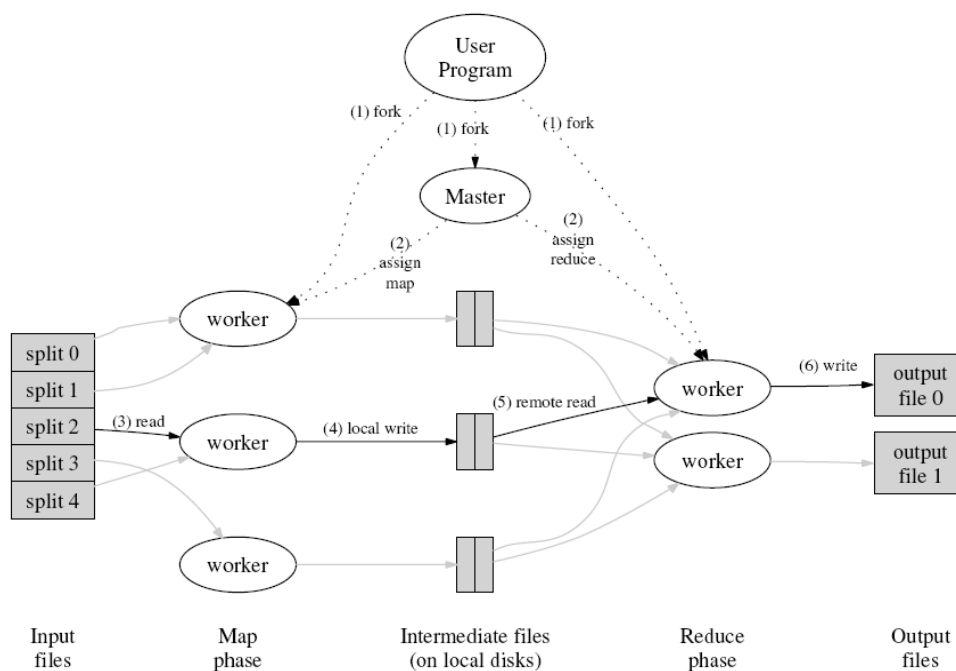


Figure1. Execution overview

2.3. Fault tolerance

因為 MapReduce 是設計來在多數機器間幫助大量資料運算的處理，當有機器發生錯誤時，MapReduce 必須提供良好的容錯機制，並減少效能的降低。

1. Worker failure

Master 會週期性的 ping worker，如果超過定義的時間沒有收到 worker 的回覆，則 master 判斷此 worker 執行失敗。如果 worker 負責的是 Map 任務，則此 worker 必須重新執行，但如果 worker 負責的是 Reduce 任務，則此 worker 不必重新執行，理由是 Reduce operation 產生的結果是儲存在 Google 信任的檔案系統(GFS, Google file system)，但 Map operation 產生的結果是儲存在當地端的磁碟。

2. Master failure

讓 master 週期性寫入檢查點(checkpoint)來紀錄執行狀態，當發生錯誤時，master 即可很快地回溯至上個檢查點再重新執行，但根據 Google 的經驗，master 出錯的機率相當低。

1.4 Refinement

1. Task granularity

設計讓 Map 任務的數量多於機器的數量，可以減少復原錯誤的時間、管線(pipeline)執行 Map 任務和 Reduce 任務，以增加吞吐量(throughput)。

2. Locality optimization

把具有相同 Key 的分割(spilt)分派至較近的群集中，可以減少 Map 分配任務和 Reduce 任務收集結果的時間。

3. Backup tasks

在 Map/Reduce operation 中有少部分的 operation 會形成 straggler，造成整體執行時間大幅拉長，其原因是一些 Map/Reduce 也必須在當地端競爭資源(CPU, Memory, local disk, and network bandwidth, etc.)，因此造成 delay latency 的增加。Google 提出一個簡單的方法來減輕 straggler 帶來的影響，就是每一份 Map/Reduce 都同時有 2 個 copies 在執行，而只要其中一項完成就可以回傳結果，發現重複執行的 overhead 很少，但卻能使整體執行時間縮短。

4. Skipping bad records

有時 Map/Reduce 有 bug 造成 record 的損壞，造成 Map/Reduce 無法順利完成，若是修復這些 record 的 cost 太高，則會有機制來判定是否要略過這些損壞的 record。

5. Ordering guarantee

讓 intermediate key/value 依據 intermediate key 的大小依遞增的方式處理，易於產生每個分割的結果。

1.5 Performance: MR_Sort

在 Google 提出的報告中，使用 MapReduce 在處理對 1 TB 的資料做排序來測量其效能。

(a) Normal condition

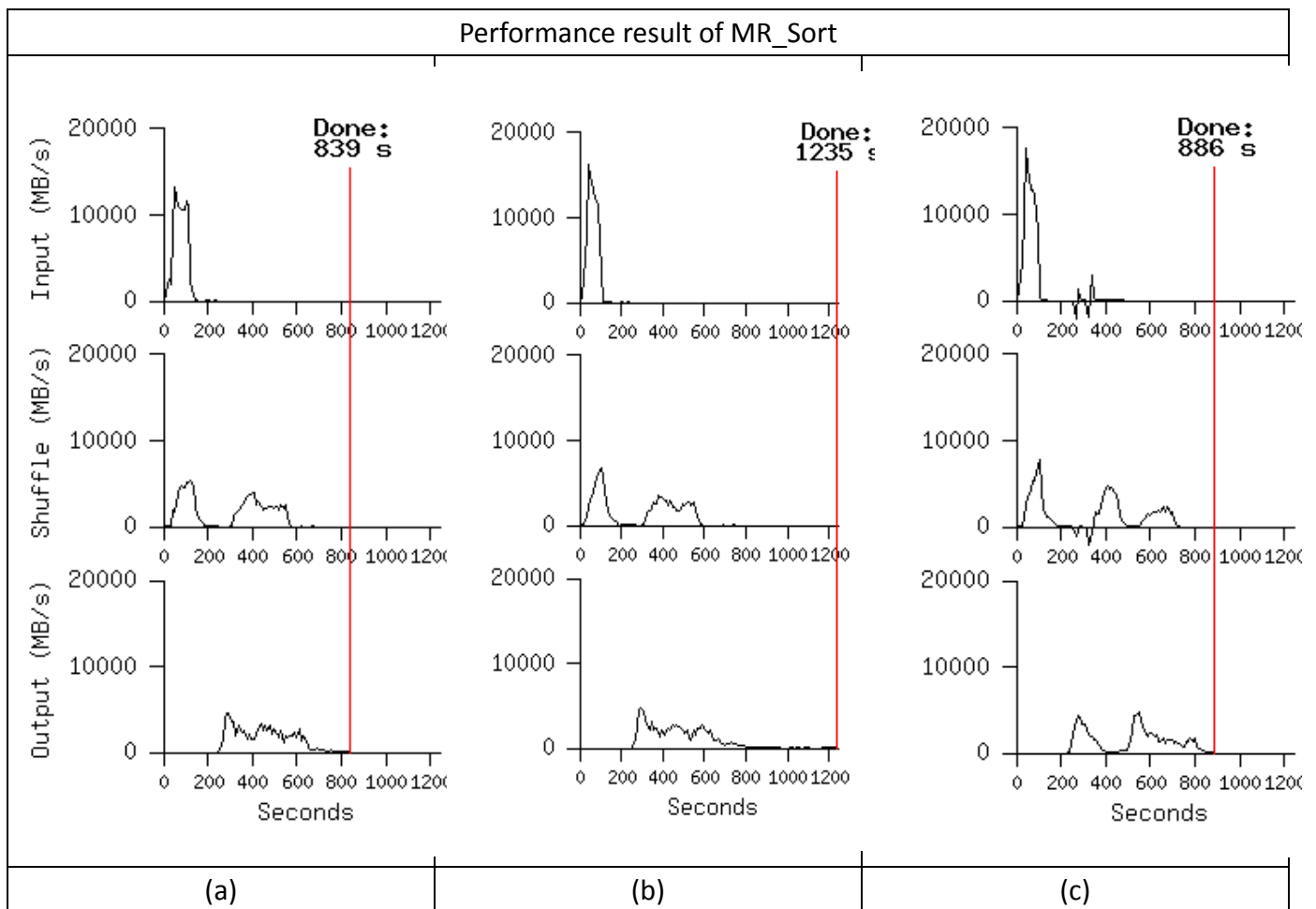
在有“backup tasks”的改進，及沒有錯誤發生的情況下，可看出 MapReduce 處理大量資料的速度是很驚人的。

(b) No backup tasks condition

主要是說明“backup tasks”對執行速度有顯著的影響。在沒有“backup tasks”的改進下，straggler 對整體執行時間的影響，執行速度與(a) Normal condition 相比慢了 44%。

(c) 200 processes killed

說明 MapReduce 有良好的容錯機制。在有 200 個 process 突然被中止的情況下，經過容錯機制的處理後，完成的時間只比(a) Normal condition 增加了 5%。



2. Bigtable

Bigtable 為 Google 為了非常大量的結構性資料而設計的分散式儲存系統，許多 Google 的服務都利用 Bigtable 來儲存所需的資料。總而言之，可說是一個資料庫系統，其特性有：

1. 壓縮
2. 高性能
3. 建構於 GFS(Google File System)之上
4. 私有的、不公開的(Google 自行發展)
5. column-oriented，與一般常見的 row-oriented 資料庫系統不同
6. 並非關聯式資料模型。
7. 資料的 index 有 row key 以及 column key，都是任意的字串。

3.1 資料模型

為稀疏且分散式的多維度 sorted map。可使用 row key、column key 以及時間戳記(timestamp)作為索引。Bigtable 中的每筆資料都是以此格式儲存：(row, column, timestamp) -> data，除了時間戳記使用 int64 之外，row key、column key 以及 data 都是以字串的方式儲存。

3.1.1 列(Row)

在 table 中的 row key 都是任意的字串，對於每個單一的 row 做讀寫都是不可中斷的(atomicly)。使用 row key 將資料以字典順序排序，這使得相似 row key 的資料將會被儲存在鄰近的 row 中。表中的 row range 以動態的方式分割，每個 row range 被稱為 tablet，為分散式以及負載平衡的最小單位。

3.1.2 行(Column)

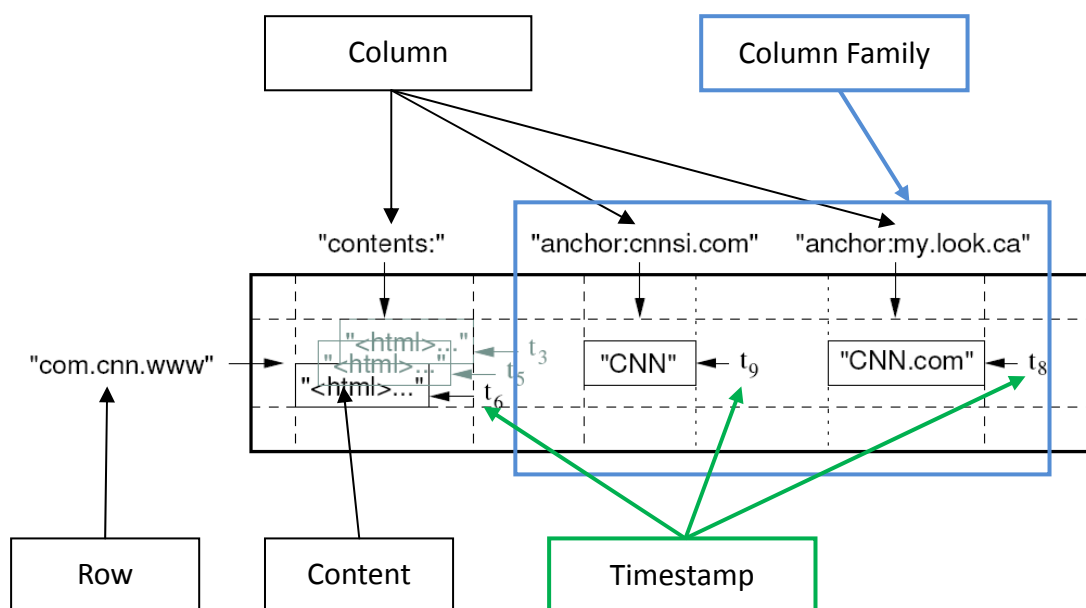
Column 可以被聚集成一個 group，稱作 column family，通常是具有相同性質的才會被聚集成一個 column family。Column family 是作為存取控制的基本單位。Column family 的產生必須在資料被儲存之前，而在 column family 產生之後，其下的 column key 才可以被使用。Column key 的命名方式為：family: qualifier，其中 family 必須是可見、可列印的，而 qualifier 可為任意的字串。

3.1.3 時間戳記(Timestamp)

在 Bigtable 中所儲存的每個基本儲存單位(Cell)都有時間戳記，這讓儲存的資料可以有多种版本，也可以避免碰撞的產生。由於是 64-bit 的整數，所以可以精確地表示實際的時間到百萬分之一秒。時間戳記可以被 Bigtable 或是客戶端的應用程式所指定，客戶端可以指定保留最新的幾份資料。

3.1.4 範例：Webtable

在此 table 中，以 URL 作為 row name，以網頁的各項性質當成 column name(contents, anchor)，同樣特性的內容聚集成一個 column family(anchor)。



3.2 APIs

Bigtable 的 API 提供以下功能：

1. 建立、刪除 table 與 column family
2. 改變 cluster server、table
3. 修改 column family 的 meta data
4. 讓使用者以更複雜的方式操作資料
 - i. 支援單一系列的處理(single-row transactions)，對於儲存在單一系列上的

資料提供 atomic read-modify-write sequences

- ii. 允許 cell 被當成整數的計數器使用
 - iii. 支援客戶端在伺服器端執行其所撰寫的 script，這些 script 是由 Google 為了處理資料所發展的語言(Sawzall)所撰寫的。使得客戶端的應用程式可以在 Bigtable 中寫入或刪除資料。
5. 讓 MapReduce 所使用

3.3 Building Blocks

Bigtable 是建立於其他 Google 所發展的幾個基礎架構之下：

使用 GFS(Google File System)，儲存紀錄以及資料，為分散式檔案系統。GFS 中的每個基本單位(chunk)為 64MB。

使用 Google SSTable 檔案格式儲存 Bigtable 的資料。SSTable 提供了有順序且不可改變的映射(key->value)，key 以及 value 都是任意的字串。並可進行以下操作：針對特別的 key 尋找其 value、在特定的 key 範圍中反覆查詢其 key/value 對。

Bigtable 還依賴了高度可靠性且穩固的分散式資料鎖定服務：Chubby。其主要功能是提供鬆散接合的分散式系統(loosely-coupled distributed system)一個介面，使其可以通知系統其他節點以達到同步的作用，設計重點在於可靠性以及可及性，但這通常與高性能相對。

3.4 實際的應用例子

在 2006 年的八月，總計有 388 個 Bigtable cluster 在大量的 Google cluster 上執行，其中大約有 24500 台桌上型電腦。

每個 Cluster 的桌上型電腦數量	Cluster 數量
20-49	47
50-99	20
100-499	20
>500	12

2.4.1 Google Analytics

此服務提供了網站管理員分析使用者瀏覽其網站的各種資料，包含使用者的來源，以及使用者如何瀏覽網頁，以便修改網站的內容。這個服務主要使用兩個 Bigtable 來紀錄，其中一個叫做 Raw Click Table，其每個 row 就是每位使用者瀏覽該網站的一個 session，其 row key 為 web site name + 該 session 所建立的時間(而 column family 就是使用者瀏覽該網站的各種行為)

2.4.2 Google Earth

此服務提供使用者可以瀏覽地球上任意位置的平面，可觀看地圖以及各種解析度的衛星影像。使用了一個 table 來放置前置處理資料以及其他不同的 table 來存放 client 資料。在存放影像資料的 table 中，每一個 row 都對應到單一的地理區段，且鄰近地點的影像資料會被確保放在鄰近的 row 中以增加存取的速度。Table 中包含一個 column family 來聯繫每個地理區段的整體來源資料。這個 column family 具有大量的 columns。此 column family 相當的稀疏，因為每個地理區段的影像資料並不多。

2.4.3 Personalized Search

此服務提供使用者紀錄其使用網頁的行為，例如網頁、影像、新聞、圖片等查詢。Table 中的 row key 就是每個使用者的 User ID，而 column family 就是每種動作。

3. Conclusion

在此報告中，我們知道 MapReduce 是一個強大的 programming model，使用者能夠藉此處理大量資料的運算並自動平行分配運算，達到了加速的效果。MapReduce 的應用相當廣泛，例如：分散式擷取(distributed grep)、分散式排序(distributed sort)、文件叢集(document clustering)、網路存取記錄統計(web access log stats)、機器學習(machine learning)，以及反轉式索引建置(inverted index construction)。至於 Bigtable 提供分散式的結構來存取所需資料，在一定的可靠性(reliability)之下發揮其效能並且具備了良好的 scalability。兩者皆是 Google 網路服務的重要技術之一。

4. 參考資料

1. MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat, *Google, Inc.*
2. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber, Bigtable: A Distributed Storage System for Structured Data, OSDI, 2006.
3. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, The Google File System, ACM Symposium on Operating Systems Principles 2003
4. Rob Pike, Sean Dorward, Robert Griesemer, Sean Quinlan, Interpreting the Data: Parallel Analysis with Sawzall, *Scientific Programming Journal*, Special

Issue on Grids and Worldwide Computing Programming Models and Infrastructure 13:4, pp. 227-298.

5. <http://en.wikipedia.org/wiki/BigTable>
6. David F. Carr, How Google Works,
<http://www.baselinemag.com/article2/0,1397,1985040,00.asp>
Behind The Google File System,
<http://www.baselinemag.com/article2/0,1540,1985047,00.asp>
7. Greg Linden, Google's BigTable,
<http://glinden.blogspot.com/2005/09/googles-bigtable.html>
8. Greg Linden, Google Bigtable paper,
<http://glinden.blogspot.com/2006/08/google-bigtable-paper.html>
9. Google's Bigtable Distributed Storage System, <http://storagemojo.com/?p=239>
10. Google 架構, <http://www.yaosansi.com/blog/article.asp?id=1145>