# 粒子群最佳化平行演算法的應用及其在 MapReduce 上的實作*

林志穎[1] 邱郁舒[2] 薛宇佑[3] 李錫智[2] 李清然[1]
[1]工業技術研究院綠能與環境研究所
[2]國立中山大學電機工程系
[3]美國紐約倫斯勒理工學院
E-mail:yschiou@water.ee.nsysu.edu.tw, leesj@mail.ee.nsysu.edu.tw

## 摘要

本研究使用粒子群最佳化平行演算法(Particle Swarm Optimization Parallel Algorithm, PSOPA)針對照明控制管理問題求最低能耗解，並且實作在單一機器(S-PSOPA)與多台機器(MR-PSOPA)上，值得注意的是本實作 S-PSOPA 與 MR-PSOPA 於能耗方面的實驗結果完全一致並在此前提下比較執行時間上的優劣，除此之外模擬更複雜的問題接續比較執行時間與所求能耗。

MR-PSOPA 採用雲端運算中屬於開放原始碼的 Hadoop 架構做為建置平台並且利用 HDFS 處理 PSOPA 中必要的資料傳遞行為，最後將 PSOPA 實作於 MapReduce 框架上。實驗結果顯示 MR-PSOPA 在粒子群數量足夠多的情況下其執行時間比 S-PSOPA 來得少並且當問題變得複雜時其執行時間更明顯少於 S-PSOPA。此外，由實驗結果得知 MR-PSOPA 能處理 S-PSOPA 無法處理的更為複雜的照明管理控制問題。

**關鍵詞**：照明控制、Hadoop、MapReduce、雲端運算、粒子群最佳化、平行演算法

## Abstract

In this study, we apply a Particle Swarm Optimization based Parallel Algorithm ( PSOPA ) to deal with the problem of lighting control management for seeking the lowest power consumption. The algorithm is implemented on standalone (S-PSOPA) as well as cluster (MR-PSOPA) platforms. The two versions produce identical results with respect to power consumption. Under the condition of getting identical results, S-PSOPA and MR-PSOPA are compared on efficiency for complex lighting control problems.

MR-PSOPA is implemented in mapreduce and information delivery between subgroups is accomplished by HDFS in Hadoop which is open source in the field of cloud computing. Experimental results show that MR-PSOPA performs better than S-PSOPA in terms of execution time when the number of particles is large enough. Furthermore, MR-PSOPA can robustly solve more complex problems which S-PSOPA may have difficulty with.

**Keywords**：Lighting control, Hadoop, MapReduce, Cloud Computing , Particle Swarm Optimization, Parallel Algorithm

## 1. 前言

照明管理控制問題可以用一個或多個光環境演算法來描述。光環境演算法的目的，就是將影響光環境的各項因素以及光環境的品質與特性，轉化成可量化、可數學運算的參數，使照明管理系統具有全自動化的光環境設計、控制能力，能夠在滿足空間照度的需求下，依據光環境的照明需求，重新評估、調整空間內所有可調控照明設備的運作參數，適度調降過量或是調高過低的燈具亮度。

晝光是最節能、環保，也是最自然、符合人因的照明光源，配合晝光照明的照明管理控制，則是將晝光的照度因素加入系統中，使控制管理系統能依據晝光照明的變化，對應調整照明設備，讓空間中照明的照度、均勻度能夠維持一定的品質，在此前提下，將照明用電量減到最低，以充分利用自然光照明，達成節能的目標。

由於晝光的不穩定，使得照明管理系統必須迅速且即時的反映當前的情況，並正確地調整照明設備，這對於單一電腦來說，運算負擔太過沉重，需要花費大量的時間，因此希望能夠透過雲端運算的優勢，配合更為妥善的光環境演算法，進而開發出一套迅速且成本較為低廉的智慧型照明管理系統，除了可以快速提高市場普及率之外，更能讓此智慧型照明系統更加的實用。

## 2. 相關研究

### 2.1. 粒子群最佳化演算法

粒子群最佳化演算法(particle swarm optimization, PSO)[1]是來自於對自然界現象的觀察，科學家試著模擬鳥群與魚群的行為模式，而發展出來的演算法。在搜尋空間中撒下一群粒子，用以模擬生物社會中資訊分享的概念，提供群聚中的粒子相互溝通與資訊交流的機制，可讓這些粒子能夠有效地搜尋近似最佳解。

#### 2.1.1. 粒子群演算法的組成要件

1. 粒子群(swarm)：個體的集合。
2. 粒子(particle)：
   - 位置(position)：各種變數的集合，可視為一個個體。

* 本研究承蒙能源局「固態照明與系統節能技術研發」專案計畫補助，特此致謝。

- 速度(velocity)：計算粒子的速度與方向。

3. PBest：記錄個體粒子的最佳適應值。
4. GBest：記錄粒子群的最佳適應值。
5. 適應函數(fitness function)：計算粒子的適應值。

### 2.1.2. 粒子群演算法流程(如圖 1 所示)

1. 初始粒子群(initialization)：最初以隨機產生粒子數為 N 的群聚。
2. 適應值計算(evaluation)：評估粒子群中每個粒子的適應值。
3. 最佳經驗之更新(update)
   - 更新 PBest(update PBest)：每次更新位置與速度重新計算粒子的適應值，若是此次的適應值比原先的記錄好，則更新。
   - 更新 GBest( update GBest)：找到此次粒子群最佳的適應值，若是比原先的記錄值好，則更新。
4. 移動粒子：更新每個粒子的速度與位置

## 2.2. Hadoop

Hadoop 是針對龐大資料做處理與保存而發展出來的雲端運算平台，是由 Apache 軟體基金會所研發的分散式運算編程工具和分佈式文件系統，並納入 Apache 的頂層專案，其專案使用 Java 為主要開發語言。它實現了 Google 的 MapReduce[2][3]編程模型，如圖 2 所示，提供簡單易用的編程接口，也提供它自己的分散式文件系統(Hadoop Distributed File System, HDFS)與 Google 檔案系統(Google File System, GFS)[4]有異曲同工之妙。

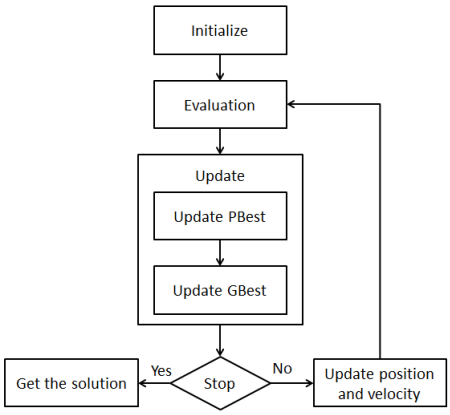


**圖 1 粒子群最佳化演算法流程圖**

Hadoop 是一開放源碼，許多公司利用 Hadoop 做為雲端運算環境中的重要基礎軟體平台，自行開發其應用程式。例如，Amazon 使用 Hadoop 建置產品搜尋引擎的索引；Facebook 使用 Hadoop 儲存內部記錄的副本和原始資料；IBM 的藍雲運算叢集(Blue Cloud Computing Clusters)；Trend Micro 使用 Hadoop 來保存與並分析由產品所傳送回來的大量可疑的病毒行為記錄檔；Yahoo 架設了 4000 個節點的叢集，用於研究廣告系統和網頁搜尋[5]。藉由 Hadoop 的應用程式將以往在超級電腦上強大的運算能力轉移到多台電腦上，利用分散式架構的概念建置另一種超級電腦，一般而言，叢集運算的成本效益高出單一超級電腦甚多。

### 2.2.1. HDFS

Hadoop 分散式文件系統(HDFS, Hadoop Distributed File System)[5]的主要目的是將超大型資料儲存佈署在大量低成本的硬體上，每個硬體都有故障的可能性，因此它必須具備很高的容錯性，可自動偵測錯誤，採用複製多個檔案因應硬體的故障，並且可由複製的檔案執行資料復原。為了確保檔案的一致性，採用 write-once-read-many 的存取模式，檔案一旦建立、寫入，就不允許修改。HDFS 以 block 為單位，每個檔案被分割成許多 block，系統會將每個 block 複製許多複本(預設值為 3)並分散儲存於 HDFS。

HDFS 是由一名稱節點 (namenode) 的主節點和多個資料節點( datanode )組成。名稱節點儲存著檔案系統的中繼資料，這些中繼資料封包包括檔案系統的名字空間等，向使用者映射檔案系統，並負責管理檔案的儲存等服務，但實際的資料放在資料節點上，而使用者在得到名稱節點回傳的中繼資料後便可與資料節點建立資料通信，如圖 3 所示。

### 2.2.2. MapReduce

MapReduce[1][2]是一種分而治之的程式設計模型，可應用於許多大規模運算問題，處理大型資料集，執行分散式運算。使用者只需指定 map 和 reduce 函數然後 MapReduce 框架會協調機器資源自動配置並處理程式的輸入、輸出與執行。

MapReduce 主要分為 map 和 reduce 兩部分，使用者必須自行定義 map 和 reduce 函數。map 和 reduce 的輸入和輸出都是採取(key, value)的組合，待處理的資料經過分割後，每一個分割片段會由一個 map 來執行，經由使用者所定義的 map 函數運算後，可得到一個以上的 (key, value)組合由 map 輸出。在 map 和 reduce 中間會經過 shuffle 和 sort 的處理，其目的是將相同的 key 收集在一起，接著針對 key 的值來進行排序， 最後做為 reduce 的輸入，因此 reduce 的輸入為一個 key 和多個 value 的組合，如圖 2 所示。

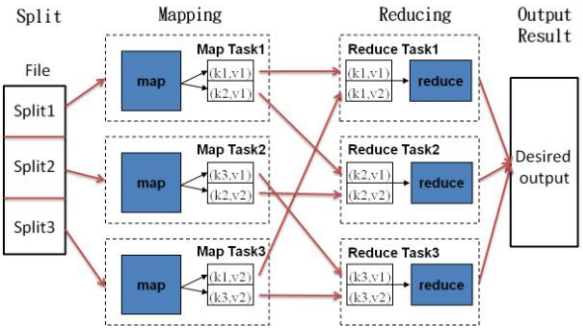


**圖 2 MapReduce 示意圖**

如果 map 任務(map task)在輸出給 reduce 處理之前發生錯誤或是因為其他原因無法完成任務，那

麼 Hadoop 會自行指派此任務給另一個節點，並重新執行失敗的 map 任務之後再輸出到 reduce，同理如果 reduce 任務發生錯誤或是無法完成，也會比照此方式處理。map 輸出的結果經由網路傳輸到處理 reduce 任務的節點上進行合併，reduce 輸出的結果存放在 HDFS 上。

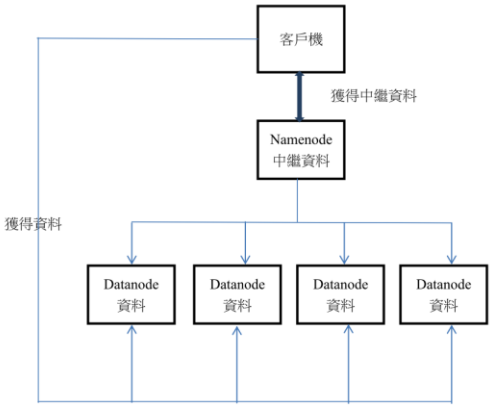


**圖 3 HDFS 架構圖**

## 3. 研究方法

### 3.1. 光環境演算法

假定目前空間中共有 n 盞燈與 m 個受照的目標，其中各個燈具的亮度為 S，受照目標所獲得的照度值為 T，示意圖如圖 4 所示。

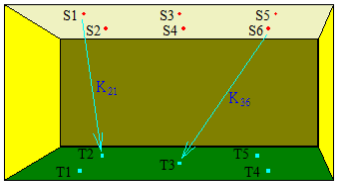


**圖 4 環境示意圖**

因此我們可以獲得光環境演算法-1 並從各燈具亮度 S 之設定值計算各目標點照度值 T：

$$\begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_m \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1n} \\ K_{21} & K_{22} & & K_{2n} \\ \vdots & & & \\ K_{m1} & K_{m2} & & K_{mn} \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_n \end{bmatrix}$$

**圖 5 光環境演算法-1**

其中 $S_1$ 指的是第一號燈具的亮度，$T_1$ 代表的是第一個目標點的所獲得的照度，$K_{21}$ 表示的是第一號燈具的亮度對於第二個目標點的照度影響係數，而我們可以透過量測各盞燈具對於各個目標點的照度影響取得係數。

實際上，我們可以設定各個目標點的期望照度 T，並透過運算取得各盞燈具 S 的理想值，如光環境演算法-2 所示：

$$\begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_n \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1n} \\ K_{21} & K_{22} & & K_{2n} \\ \vdots & & & \\ K_{m1} & K_{m2} & & K_{mn} \end{bmatrix}^{-1} \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_m \end{bmatrix}$$

**圖 6 光環境演算法-2**

我們必須同時考量各盞燈具的能耗特性，如圖 7 所示，來進行總耗能的最佳化。在滿足各個目標點所需的最小照度前提下來最佳化所有燈具的總能耗。

| 燈具#1 | | |
|---|---|---|
| 調光指令 | 燈具能耗(W) | 燈具輸出光量(lm) |
| 0 | 4 | 5 |
| 1 | 4.5 | 7 |
| 2 | 5 | 10 |
| | | |
| 255 | 60 | 4200 |

**圖 7 燈具能耗特性**

於是我們的演算法流程圖如圖 8 所示，若加入畫光的因素，其流程圖如圖 9 所示：

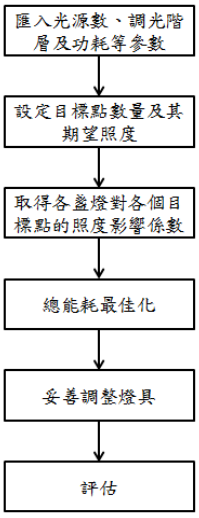


**圖 8 光環境演算法流程圖**

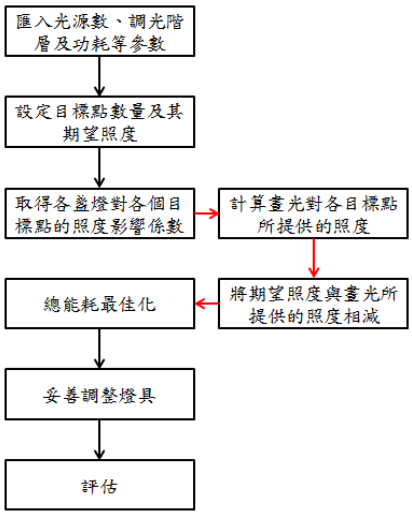


**圖 9 考量畫光影響之光環境演算法流程圖**

### 3.2. 粒子群最佳化平行演算法

PSO 與 PSOPA 之間最大的不同在於 PSO 在隨機初始化粒子階段將所有粒子視為一個群體而 PSOPA 則是將所有粒子視為 N 個群體，因此 PSO 每次迭代求出一個值 GBest 且當達到停止條件時此 GBest 即為所求最佳化解，然而 PSOPA 每次迭代求出一個集合 ***GBest*** = {GBest1, … , GBestN}，再從中挑選出 GBest 而當達到停止條件時此 GBest 即為所求最佳化解。

S-PSOPA 的架構及流程如下圖 10：一開始的虛擬隨機初始化粒子是為了確保 S-PSOPA 及 MR-PSOPA 的實驗結果兩者一致，不僅在內迭代次數、外迭代次數甚至於粒子訊息更動的軌跡均為一致，以客觀比較兩者的執行時間。

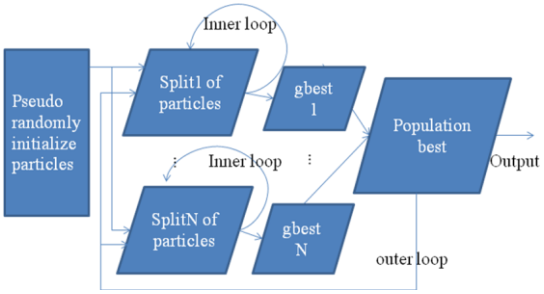


圖 10 S-PSOPA 架構流程

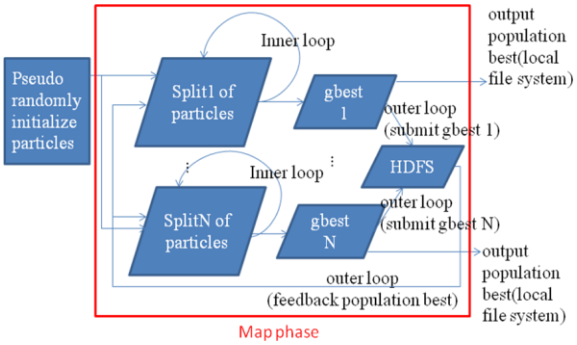


圖 11 MR-PSOPA 架構流程

MR-PSOPA 與 S-PSOPA 的不同之處在於處理 N 群中的粒子時 MR-PSOPA 是 N 台機器同時執行處理而 S-PSOPA 則由一台機器從第 1 群處理到第 N 群。在 MR-PSOPA 的實作上需要解決的難處在於它不像 S-PSOPA 在處理的過程中即已掌握住各個群的資訊而需要在機器之間做粒子資訊傳遞的動作以求得 GBest 進而執行外迭代處理。

迭代式演算法結合 Mapreduce 的相關實驗結果比較顯示以 Haloop[6][7] 及 Twister[8]作為實作平台皆表現得比 Hadoop 好。Haloop 是針對 Hadoop 的部份內容做修改而來，其效能改善是致力於有效重複使用大量靜態資料而 Twister 則是以共用記憶體的概念開發出讓各台機器可以在記憶體中直接交換訊息的功能，由於我們處理的問題沒有需要處理的大量靜態資料及大量的訊息交流，故我們依舊採用 Hadoop 作為本次實驗的平台。

由於粒子資訊互相傳遞的控制是在 Map 階段內處理，故從圖 11 架構流程中可以看到我們的 Mapreduce 實作架構是沒有 Reduce 階段的。

下列為 map()的虛擬碼：

```
function map(){
    虛擬亂數產生 pps[1] 個粒子;
    do
        對於此分割中的粒子執行 PSO 演算法;
        利用 HDFS 更新 population best;
        更新此分割中的 GBest;
    until 達到終止條件;
    當前的 GBest 即為所求最佳解;
}
```

## 4. 實驗結果

### 4.1. 實驗資料

本實驗採用的實驗資料包含光照係數表、燈階耗能表、燈階光量表及受照目標照度表均是由工研院提供的真實資料。而真實資料欲處理的問題是在可調光 16 階的燈 12 盞在受照目標數為 5 的情況下(12×5)必須快速求得最低能耗解。除此之外，本實驗進一步以真實資料為基礎模擬出同樣可調光 16 階的燈在更多燈數及受照目標數的環境下使用 S-PSOPA 及 MR-PSOPA 求最低能耗解。

### 4.2. 實驗環境架設

本實驗使用 4 台個人電腦來實作。配備皆為 Intel CPU i5-2500 3.30GHz 處理器、500GB 的硬碟、8GB 的 RAM。各台的作業系統為 Ubuntu 12.04 LTS，並安裝 Hadoop 1.2.1 版本，以下實驗皆在上述環境下執行。

### 4.3. 實驗結果及分析

本實驗的 PSOPA 設有停止條件且不指定內迭代及外迭代必須執行特定次數以上以期能達到較客觀準確的執行時間，因為假如實際上達到停止條件時的迭代次數和指定的次數有差距，則剩下的執行時間均是沒有必要的浪費，例如四塊分割中的粒子其內迭代次數分別為 3,30,3,3，假如指定必須迭代至少 50 次，則分割中的粒子其迭代次數分別為 50,50,50,50，原本只需要 3 + 30 + 3 + 3 = 39 的迭代次數變成 50 + 50 + 50 + 50 = 200 的迭代次數，這使得在 S-PSOPA 及 MR-PSOPA 的執行時間比較上也有很大的差異：如果迭代 1 次需要 1 秒，39 次迭代在 S-PSOPA 需要 39 秒，MR-PSOPA 需要 max(3,30,3,3) = 30 秒；迭代 200 次在 S-PSOPA 需要 200 秒，MR-PSOPA 需要 max(50,50,50,50) = 50 秒，是故本實驗僅設停止條件且不指定內迭代及外迭代的執行次數，並且是在分割數與電腦數均為 4 的

[1] pps ( particles per split )，即每個分割中的粒子數，本實驗中使用的單位為 $10^5$。

條件下完成。

### 4.3.1. 真實資料 12×5

由圖 12 可以看到用 S-PSOPA 及 MR-PSOPA 在 12 盞燈和 5 個受照目標的環境下的時耗比較：
我們可以看到 S-PSOPA 在 pps 為 35 以上時，其執行時間已大於 MR-PSOPA，在 pps 為 46.875 時期執行時間為 39.825 秒，而在 pps 為 47.5 時已經超出記憶體的負荷。另一方面，MR-PSOPA 在 pps 為 46.875 時其執行時間為 31.093 秒，pps 最多能達到 181.25，pps 達 182.5 時也超出記憶體的負荷，這意味著它在遭遇更複雜的問題時有更大的機會求出更好的能耗解，可惜在執行時間方面 MR-PSOPA 也沒有明顯的優於 S-PSOPA，再者，從圖 13 來看所求平均能耗可以發現在 12×5 這個環境下 S-PSOPA 及 MR-PSOPA 在各種 pps 下求得的最低能耗解完全一樣為 246.34，於是我們模擬更複雜的環境來觀察執行時間及平均能耗。

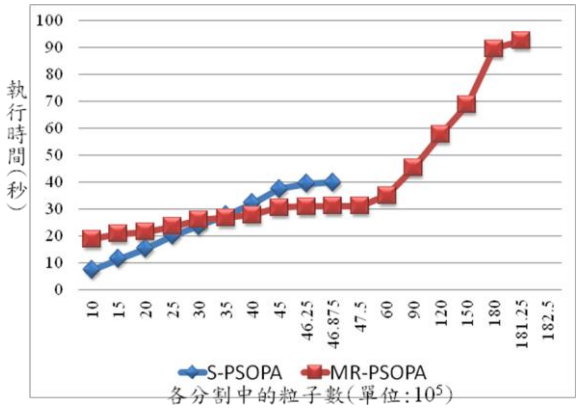


**圖 12 S-PSOPA & MR-PSOPA 時耗比較(12×5)**

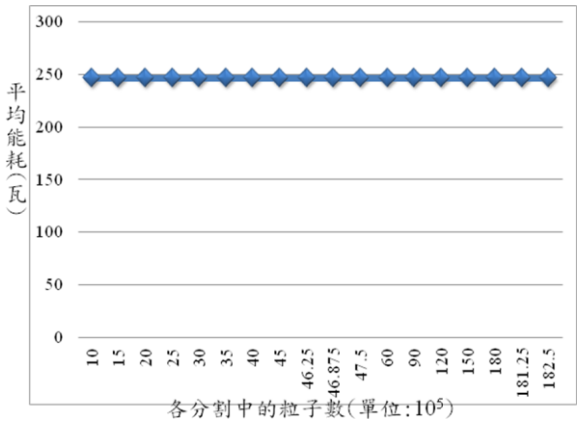


**圖 13 於不同粒子數所求平均能耗分布(12×5)**

### 4.3.2. 模擬資料 36×5

圖 14 及圖 15 是 36 盞燈與 5 受照目標的模擬環境下的實驗結果，由於燈數變多使得每個指令長度變長，連帶地也使 pps 的範圍也變窄，不過這並不影響 S-PSOPA 與 MR-PSOPA 之間的比較：S-PSOPA 在 pps 大於 8 時其執行時間已高於 MR-PSOPA，在 pps 為 20.625 時其執行時間為 75.003 秒，在 pps 為 20.78125 時會超出記憶體使用量；MR-PSOPA 在 pps 為 20.625 時其執行時間為 43.556 秒且 pps 最多可以達到 81.25，pps 達 82.5 時也超出記憶體使用量。

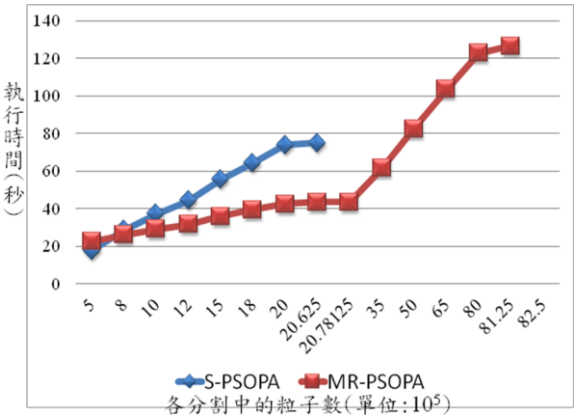


**圖 14 S-PSOPA & MR-PSOPA 時耗比較(36×5)**

由圖 15 可以看出較大的 pps 能有較高的機率找到最低能耗解，而圖中在 pps 為 65 找到的最低能耗解比 80 和 81.25 找到的最低能耗解都來的低，這說明了當粒子幸運地找到了很棒的最低能耗解時，此平均結果會表現得比平常來的優異，雖然實驗結果是執行 100 次取平均，但在這樣的解空間下(共有 $16^{36}$ 組解)，仍然無法有效平滑實驗結果。

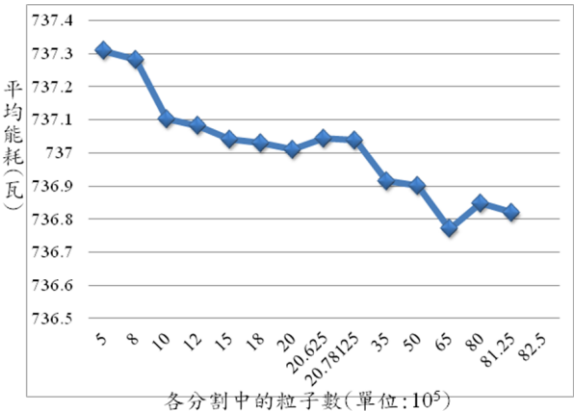


**圖 15 於不同粒子數所求平均能耗分布(36×5)**

### 4.3.3. 模擬資料 36×15

圖 16 是 36 盞燈與 15 受照目標的模擬環境下的實驗結果，由於受照目標數變多使得執行時間變長，不過由於我們模擬資料的方式著重於提高計算複雜度，其平均耗能的分布和圖 15 是一樣的，如果是執行在真實資料上，則平均能耗分布也會隨之而有所不同。

圖 16 顯示 S-PSOPA 在 pps 為 20.625 時其執行時間為 103.608 秒而 MR-PSOPA 在同樣的 pps 時其執行時間為 51.069 秒，由實驗結果得知 MR-PSOPA 在越複雜的問題表現的比 S-PSOPA 來得越好。

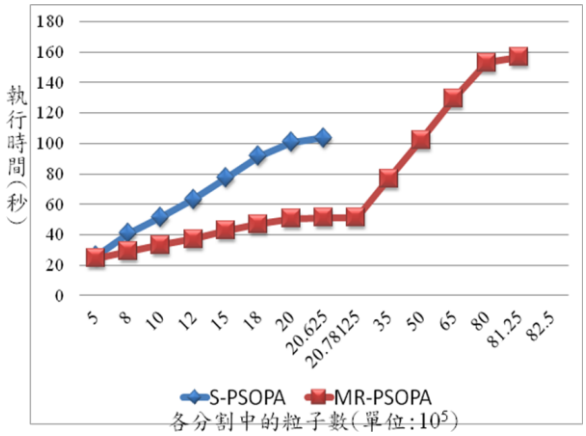


**圖 16 S-PSOPA & MR-PSOPA 時耗比較(36×15)**

## 5. 結果與未來展望

### 5.1. 結論

實驗結果顯示對於真實資料 12×5 而言 S-PSOPA 表現得比 MR-PSOPA 來得好，MR-PSOPA 只在粒子群數量足夠多的情況下其執行時間比 S-PSOPA 來得少而且執行時間的改善微乎其微，探究原因：問題本身不夠複雜以致於 MR-PSOPA 無法展現其優勢。在模擬資料 36×5 及 36×15 的表現上 MR-PSOPA 的執行時間明顯少於 S-PSOPA 而且更能展現出 MR-PSOPA 在處理越複雜的問題越能大顯身手。此外，由實驗結果得知 MR-PSOPA 不僅在架構的差異上(記憶體限制)及效能的展現上均能處理 S-PSOPA 無法處理的更為複雜的照明管理控制問題。

### 5.2. 未來研究方向

本實驗結果顯示出 MR-PSOPA 能夠處理更複雜的問題，我們希望能在未來針對更多更複雜的問題繼續完成實驗結果並使研究更趨完整。再者我們想要進一步的完成結合粒子交換或者重新分布的 PSO 演算法並實作在 Mapreduce 架構上以期獲得更好的實驗結果並促進社會進步改善人們生活品質。

**參考文獻**


[1] J. Kennedy and R.C. Eberhart, “Particle swarm optimization,” in Proceedings of IEEE International Conference on Neural Networks, Perth, Australia, vol. 4, pp. 1942-1948, 1995.

[2] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” in Communications of the ACM, vol. 61, no. 1, pp. 107–113, Jan. 2008.

[3] T. White, Hadoop: The Definitive Guide, America: O’Reilly Media, 2009.

[4] I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[5] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The Google File System,” in Proceedings of the 19th ACM Symposium on Operating Systems Principles, 2003.

[6] http://hadoop.apache.org/

[7] Y. Bu, B. Howe, M. Balazinska and M. D. Ernst, “HaLoop: Efficient Iterative Data Processing on Large Clusters,” in Proceedings of the Very Large Database Endowment, Volume *3* ,Singapore, 11-17 September, 2010.

[8] Y. Bu, B. Howe, M. Balazinska and M. D. Ernst, “The HaLoop Approach to Large-Scale Iterative Data Analysis,” *in* The VLDB Journal (VLDBJ), Volume 21, Number 2, April, 2012

[9] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S. H. Bae, J. Qiu and G. Fox,” Twister: A Runtime for Iterative MapReduce," in The First International Workshop on MapReduce and its Applications (MAPREDUCE'10) - HPDC2010