二、研究計畫內容：

(一)摘要

由於容器(container)擁有輕量化的虛擬層級和易於移植的特性，大大助長了雲端系統的開發，成為了當紅的虛擬化技術，而在一個容器已經大量取代傳統虛擬機的時代，管理容器所使用之系統更是受到重視。

Kubernetes 是 Google 開發且開源的系統，專門用來自動化部屬、彈性擴充及容器應用管理。在 Kubernetes 上進行 Pod 配置時，使用者可以為 Pod 中的每個容器設定需要使用的計算資源，當一個 Pod 創建成功時，Kubernetes 中的 Scheduler 會參考使用者所設定的資源用量，為該 Pod 選擇一個節點(Node)來執行，這個過程中有兩個問題:

1. 有些資源在異質電腦上的定義其實是不同的，例如以 CPU 來說，假設主機 A 使用較高等級之 CPU，主機 B 使用相對較低等級的 CPU，若同樣要求使用一個核心來處理，兩者的效能表現必然不同。本計畫預計對 Kubernetes 架構中異質電腦上的資源進行計算，讓資源的定義在異質電腦上能夠相等。
2. Scheduler 為 Pod 選擇節點的方式為，先找出所有所持有資源大於 Pod 所要求資源的節點，再使用幾個定義好的規則對節點進行評分並累加得分，最後取得分最高的節點分配。我們思考是否能對 pod 的行為進行需求分析，將分析結果套用在多維背包問題演算策略，利用改善節點配置的方法，提升資源的利用度。

本計畫預計使用 Docker 作為容器，以多個異質電腦作為 Kubernetes 架構的環境，進行對於上述兩點問題改善方法的研究，將改善方法以自己開發的 Scheduler 實現，並將有不同資源需求的應用程式(服務)同時部屬在此架構上進行測試，用以評估能源效率與系統效能之提升程度。

(二)研究動機與研究問題

當 Kubernetes Scheduler 進行 Pod 的調度時，第一要點是確定節點有足夠的資源能讓 Pod 運行，如果一個將會需要大量資源的 Pod 被分配到資源有限的節點上，這個節點有可能會耗盡資源然後停止工作，進而導致程序失去控制。因此，Kubernetes 提供使用者對容器去進行資源要求(Request)與限制(Limit)的設定，Request 指的是容器要求能夠獲取到某資源的最少數量，Limit 指的是容器能使用的資源上限。如此在 Scheduler 調度 pod 時，就會先檢查目前節點上每種類型資源與容器要求資源的和，有沒有超過該節點上全部的資源數量，確保資源足夠可以供給新加入的容器使用。而當容器使用資源超過所設定的限制時，系統也會進行判斷是否要將當前容器終止，以確保節點可以持續正常的運作。

Kubernetes 所提供可以被限制的資源有兩種:

1. CPU—基本單位是 millicores，為絕對數量
2. Memory—基本單位是 Bytes

我們可以發現到，若使用異質電腦作為 Kubernetes 節點，1 個 CPU core 所提供的運算效能很有可能是不同的，若能對異質電腦的資源進行標準化，再以容器本身對資源的需求比重作為分配的依據，將可以提升對資源的利用率，但還必須與 Kubernetes Scheduler 進行配合，目前 Scheduler 用來進行調度的方法為，使用兩個步驟選出滿足資源需求且經過優先權策略評分後獲得最高分的節點來調度。本計畫希望使用多維背包問題策略，搭配上述標準化過的資源與容器需求的分析結果，進行演算，希望藉由優化分配方法，提升應用程式的工作效率、

系統的能源效率、減少資源的浪費。

(三) 文獻回顧與探討

虛擬機(virtual machine)技術在過去這十幾年來造就了互聯網上各種服務(網站、手機 APP、線上遊戲...等)蓬勃的發展，這些服務的後端基礎建設則由雲端公司提供，例如: Google、AWS(Amazon Web Service)等，而隨著愈來愈多的服務提供，雲端公司也變得有更多的儲存需求和擴展需求，因為雲端架構是由大量分散式微型服務所組成，所以像 Google 這樣的雲端公司就改為使用容器(Container)這種輕量級虛擬化技術，容器與傳統虛擬機的差別是，容器在主機原有的作業系統上作為獨立的進程運行，只包含了應用程式的程式碼、函式庫和環境配置文件，由作業系統來分配系統資源給容器使用。傳統虛擬機則是實作在虛擬的硬體層之上，依靠額外的虛擬層管理軟體(Hypervisor) 來分配硬體資源及管理，除了應用程式之外，還包含了完整的作業系統。容器省下了運行作業系統及 Hypervisor 的成本，可以有只執行單一程式的大小，讓部屬一個容器的時間非常短，甚至只需幾秒，需要消耗的運算資源更少，這些特點讓硬體主機能承載更多的容器，也因為不用啟動作業系統，而有著更快的運行速度，不管是在開發、部署或管理上，容器都較有效率與彈性。

Kubernetes[6]是一個叢集管理系統，用於自動部屬、擴展與管理容器，支援符合 OCI runtime-spec 規範的容器，像 Docker、rkt、runc 等，本計畫選擇 Docker[7]作為實驗用的容器，Kubernetes 提供的主要功能如下:

1. 使用數個容器組成一個服務(應用程式)
2. 使用叢集架構管理所有機器上的容器。
3. 解決不同機器上容器之間的通訊問題
4. 有自我修復能力，若叢集的某個部分故障，工作將被分配到其他部分繼續運行。

在 Kubernetes 出現之前，是透過容器技術中的私有網路，讓原本獨立互不干擾的容器可以進行溝通，但隨著用戶使用的需求隨著越來越大，同一台主機上提供的服務、要處理資料量等會越來越多，很快的出現不夠用的問題，需要加上新的主機來幫忙，Kubernetes 就是為了多機的容器管理而產生，屬於分布式系統，架構由以下兩種節點組成：

1. Master: 叢集控制節點(主節點)，每個叢集都需要一個 Master 節點負責整個叢集的管理與控制，他會對其他節點發布命令及任務。
2. Node: 工作負載節點，Master 節點會分配工作(運行容器)給 Node，可以同時運行多個容器並動態的增加。

Master 節點與 Node 節點上分別有不同的程序來協助整體架構的運行，在 Master 節點上有以下四項:

1. API Server : Kubernetes 裡所有資源操作(新增、刪除、修改、查看…)的唯一入口，提供 HTTP Rest 接口進行叢集控制。
2. Controller Manager : 控制 Kubernetes 裡所有控制器(endpoints controller、node controller、replication controller、service controller)的自動化控制中心。
3. Scheduler : 負責將 pod 調度(資源調度)的程序。
4. Etcd : 負責存放叢集狀態和配置，可以從 etcd 中獲取叢集中節點的機器狀態，所有容器的狀態也是放在這裡。

而在每個 Node 上則都有以下三個程序:

1. Kubelet : 與 master 節點即時溝通，負責完成 Pod 創建、啟動、監控、重啟、銷毀等任務。
2. Kube-Proxy : 管理網路連線用的通信程式。
3. Docker Engine : 啟動 Docker 容器的必要系統。

Kubernetes 叢集架構如下圖 1 所示

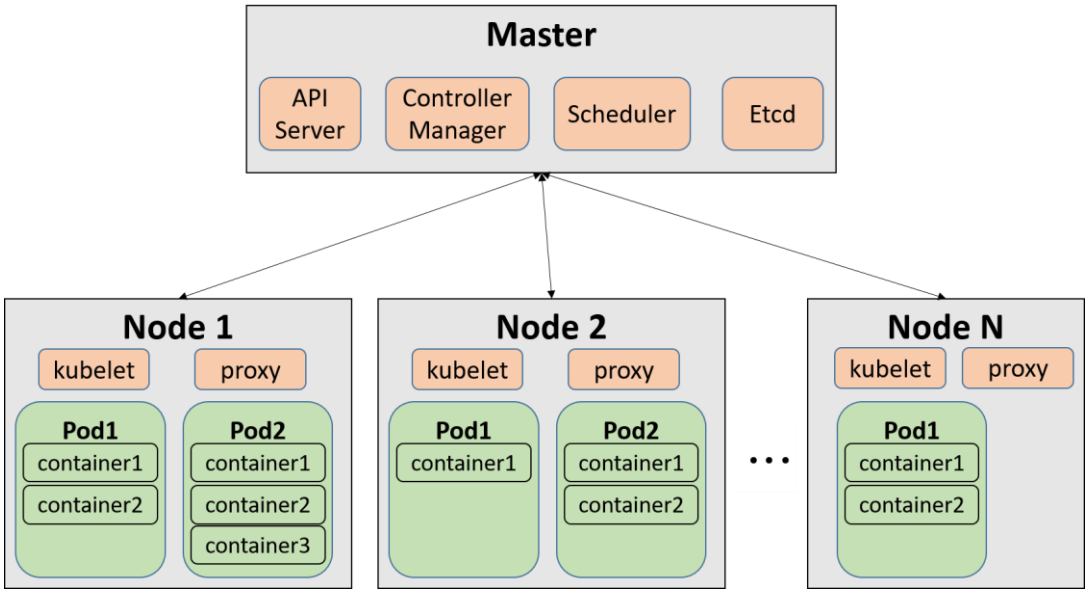

圖 1 - Kubernetes 叢集架構圖

Master 與許多 Node 組成 Kubernetes 的叢集(Cluster)，pod 是 Kubernetes 最小的部屬單位，一個 pod 裡可以有多個容器。如果某個 Node 發生意外，例如硬體故障導致無法繼續提供服務，其上的正在執行的 pod 與新產生的工作會被 Master 節點的 Scheduler 分配到其他 Node 上，而不會造成服務的停止。

Kubernetes Scheduler 預設調度的原則分為兩個步驟。步驟一:檢查所有 Node 篩選出滿足 pod 的資源與其他要求的候選節點。步驟二:使用優先權策略(Prioity)計算出每個候選節點的積分，優先權函數將對節點給出 0-10 的分數，其中 10 代表最適合，0 代表最不適合。將每個優先權函數得出的值相加作為節點的最終得分，Scheduler 會選擇最終得分最高的節點來運行容器。此方式並未考慮異質節點上資源實際的定義，例如 CPU 等級可能有高低之分、耗能幅度不同，若配合主機資源的特性與容器的需求考慮調度策略，能使節點資源有更適當的利用，提升能源效率與服務品質。

本計畫預計使用 Hadoop[8]進行資料運算，作為測試程式，Hadoop 是一個用來儲存與管理大量資料的雲端平台，為 Apache 軟體基金會底下的開源軟體。Hadoop 是一個叢集系統（cluster system），採用 HDFS 分散式檔案系統（Hadoop Distributed File System）將資料存放在叢集中，叢集系統中可能高達幾千個節點，其中會有一個 Master Node，其他的則做為 Slave Node。Hadoop 存放資料的方法是，先將資料分割成數小塊（block），再讓每個小塊擁有三個副本(Data replication)，最後將這些小塊分散給 Slave Node 保管。Slave Node 用 DataNode 程式來存資料，Master Node 則用 NameNode 程式來監視所有 Slave Node 上資料的存放狀態。如果 NameNode 發現有哪個 DataNode 上的資料遺失或遭到損壞，就會尋找其他 DataNode 上的副本（Replica）進行複製，保持每小塊的資料在整個系統都有三份的狀態。透過這種檔案系統，Hadoop 能夠儲存高達 TB（Tera Bytes）甚至 PB（Peta Bytes）大小的巨量資料，並且不用擔

心單一檔案的大小超過一個磁碟區的大小，或是某個機器故障導致資料遺失。Hadoop 使用 MapReduce 技術處理節點上的資料，MapReduce 是以 Map 跟 Reduce 為基礎的應用程式，Map 就是把處理資料的工作分散在各個節點上，Reduce 則是把各節點運算出的結果回傳進行整理。用這樣的方法可以在上千台機器上平行處理巨量資料，節省許多時間。

在容器調度策略文獻方面，文獻[1]解決了在虛擬網絡功能（VNF）中使用容器的問題:高性能 VNF 通常會在容器之間產生頻繁的通信工作負載,但是因為容器通信效率不高，導致 VNF 的性能大幅降低。在主機之間正確分配容器能有效且低成本降低通信工作負載，通常的做法是，將提供相同服務的容器部屬在鄰近或相同的主機上，但這種方法會讓吞吐量下降，原因是提供相同服務的主機通常都使用同一種的系統資源，導致主機上的資源利用率嚴重的不平衡。為此該團隊提出了一個名為 FreeContainer 的解決方案，利用一種新的兩階段算法在主機之間重新分配容器，第一階段是對具有高資源利用率的主機進行處理,因為此種主機通常是影響整體吞吐量的瓶頸,第二階段是使用 local search 再優化。此團隊在百度叢集中部署了 6000 台服務器和 35 項服務進行了大量實驗以評估所提出方法的性能，實驗的結果為 FreeContainer 可以將總體吞吐量提高 90%，同時顯著降低通信開銷。

文獻[2]對 Docker Swarm 預設調度方法進行改善，目的為:1.計算資源的消耗量，以進行收費(使用者付費策略)，提升公司利益。2.只在必要且合乎經濟效益的應用程式上消耗資源，避免資源的浪費。3. 讓用戶選擇採用容器技術以建立市場。此團隊開發以 Service Level Agreement (SLA)為基礎的調度策略，根據用戶的 SLA 來提供執行服務的容器，有三種 SLA 類別:

1. 長期服務(SLA 1 級): 持續性的服務，負載(CPU, ram, disk, network)的峰值低。
2. 短期服務(SLA 2 級):這種服務會導致過載(overload)，或是必須要等待遠程的服務。例如轉換 PDF
3. 微服務(SLA 3 級):服務生命週期極短，只有幾毫秒。

此種調度策略特點為根據服務經濟模型(SLA 等級)和目前平行機器上私有的基礎設施部屬程度，動態的為應用程式計算所需的 CPU 核心數，使資源能最有效的被利用。

文獻[3]團隊發現 Swarmkit 容器管理工具系統預設的節點分配策略 Spread 做法是選擇容器最少的節點來運行新的容器，這種方法的目的是讓節點之間的任務分配很平均，但是忽略了異質節點上主機的資源不相同，以及因為有各式各樣的服務，所以容器對資源種類的需求也不相同。因此，他們提出了一種新的分配策略 DRAPS(Dynamic and Resource-Aware Placement Scheme)，這種策略會根據異質節點架構叢集中的可用資源與服務的需求來動態的分配容器，對不同需求的服務，例如 CPU 密集型服務、Memory 密集型服務...等，提供更適合的資源分配方式，提高異質節點叢集中的系統性能。做法是透過監控容器來取得容器需要的主要資源類型，然後把此容器和與其具有互補需求的容器放在同一機器上，平衡資源的使用。該團隊將 DRAPS 實施到 Swarmkit 中，並使用 4 種類型的 18 個服務進行實驗。實驗結果表示，DRAPS 優於 Spread，並在一個特定節點上降低了 42.6%的使用率。

上述文獻皆對調度策略進行了改良，並且有顯著的效果，說明了在多異質節點叢集架構上，資源的分布情況對整體性能有很大的影響，若能將節點硬體資源與服務需求因素考慮進調度策略，對資源做適性的分配，對系統會有節省能源、提升性能的益助。

本計畫旨在對每部 Kubernetes 架構上的主機資源與應用程式的需求進行分析，因為每部主機擁有的硬體資源不盡相同，例如:CPU 核心數相同，但運算速度不同、網路速度不同等，藉由分析後的資訊來改善調度策略，讓所提供的硬體資源更貼近應用程式的需求，我們預期若能在主機之間進行適當的容器分布，將能提升資源的利用效果，例如:為運算需求較大的 pod 選擇有較好效能 CPU 主機的節點，以提升速度，降低 pod 執行時間，為需要等待(沒有一直在執行運算)的 pod 選擇較省電的 CPU，以節省耗電量。因要考慮 CPU 核心數、記憶體以及主機耗電等多個因素，預計使用多維背包問題算法來求出最佳的容器分布方法，並在經由我們改善過調度方法 Kubernetes 架構運行 Hadoop 上的服務，以驗證修改過的系統效能。

(四)研究方法及步驟

問題定義

1. 找出 pod 的主要資源需求

   同一個 pod 裡的容器通常是用來提供特定的某個服務，例如:MySQL 資料庫服務、Http Web 服務、數學計算…等，因此我們預計透過監控 pod 的行為，使用 profile 技術求出對此 pod 來說資源的需求比重，像是 CPU、網路速度、I/O 需求。

2. 對節點主機資源進行分析

   取出每個節點主機的硬體資料，例如 CPU 的頻率、對於程式的 CPI 等，進行主機各資源提供能力的分析與評比。

3. 使用多維背包問題優化資源配置

   基本多維 0-1 背包問題:

   給定 n 個物品以及他們個別的 size(體積)、weight(重量)與 profit(獲益)，在總體積不超過 S 且總重量不超過 W 的條件下，找出一些物品使得總獲益最大。

   將多維 0-1 背包變形成多維多背包問題，套用在資源配置上:將 pod 視為物品，Node 視為背包，背包有記憶體大小及 CPU 核心數量等資源的限制，若 pod 對資源的最低要求(request)，有其中一項超過了背包目前剩餘的容量，則視為裝不下，無法將此 pod 分配到節點上。而每個物品都有價值，用來判斷裝哪個物品的利益較大，在這點上我們改為不同背包視同個物品的價值將有不同，舉例來說就是，將物品 A 裝進背包 1 的價值為 50，裝進背包 2 的價值為 100，那我們可能就會選擇將物品 A 裝進背包 2，價值的設定則取決於 pod 的需求特性與節點資源特性的相符程度，例如某 pod 提供運算的服務，那麼他在運算速度較快的電腦上就會有較高的價值。最終的目標是在物品裝完之後，讓所有背包價值的合能為最大的裝法。

   虛擬碼如下:

   Input:

   一個欲分配的 pod、叢集中所有的 Node

   Output:

   最適合分配的 Node

```
//pod 的結構
Typedef struct{
   int request_cpuCores; //pod 所要求的 CPU 核心數
```

```
    int request_memory;  //pod 所要求的記憶體大小
    int[] profile; //pod 使用資源的特性
}pod;

//Node 的結構
Typedef struct{
    int cpuCores; //節點目前可用的 CPU 核心數量
    int memory; //節點目前可用的記憶體大小
    int[] profile; //節點資源特性(CPU 的 CPI 等)
}node;

while (the pod is not packed)
{
        嘗試使用以下作法分配:
          1. 從所有 Node 中選出符合 Pod 資源要求的節點。
          2. 將 pod.profile 與 node.profile 提供的資料利用線性回歸等
             方法運算，作為對每個 node 來說裝入 pod 的價值，以解決此
             多維多背包問題。
}
Output node; //將最後選中的 node 輸出
```

預計採用方法

- 修改 Scheduler

  Kubernetes 允許開發者可以使用自己開發的 Scheduler 來進行 pod 的調度，在沒有設定的情況下，pod 會使用系統預設的 Scheduler，若在 pod 設定文件中指定自訂的 Scheduler name，預設的 Scheduler 將不進行該 pod 調度，轉由指定 Scheduler 來進行。本計畫將進行自訂 Scheduler 的開發，並將其部屬在 Kubernetes 系統中，指定所有 pod 使用自訂的 Scheduler 進行調度。

- 資源監控與分析

  資源的監控則是透過 cAdvisor(Container Advisor)[9]是 Kubernetes 預設監控容器狀態的工具，當節點上的 kubelet 組件啟動同時會自動啟動 cAdvisor，cAdvisor 便會開始收集此節點上主機和所有容器的運行資訊，包括 CPU、Memory、網路、硬碟等資源的及時使用情況詳情，但僅限於單個節點，為了達到同時監控多節點上的主機及其上的所有容器，我們使用以下三個工具來達成叢集系統的監控:

1. Heapster[10]:將每個節點上 cAdvisor 收集的數據進行彙整的系統，他經由使用每個節點上 kubelet 的 API，再透過 kubelet 調用 cAdvisor 的 API 來取得資源使用數據，對數據進行整合再存入後端儲存系統(預計採用 Influxdb)。
2. Influxdb(influxdata)[11]: InfluxData 開發的開源時序型資料庫，適合用來處理和分析像資源監控數據這種時序相關數據。
3. Grafana[12]:將時序資料庫中的數據以圖表等方式呈現的工具，將使用此工具圖形化 Influxdb 中儲存的資料，方便進行理解與分析。

Heapster、Influxdb、Grafana 均以 pod 的形式啟動和執行，架構如圖 2 所示

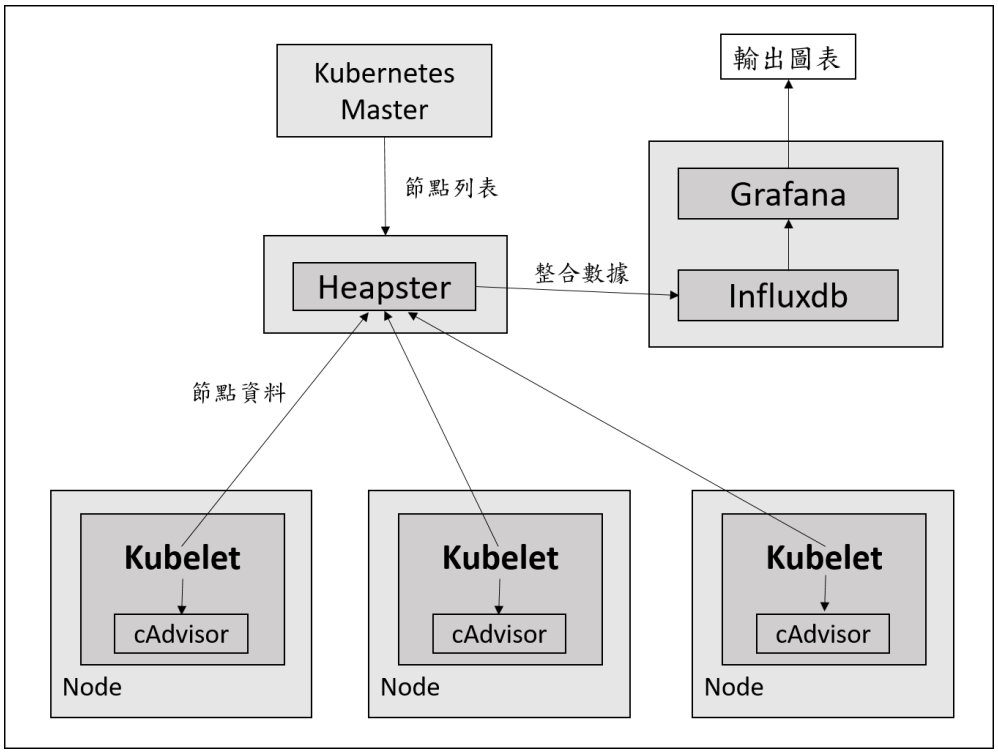

圖 2 – 叢集監控系統架構

本計畫將對叢集資源分布情形進行分析，配合每種不同測試程式所需要的資源需求，規劃出對節點來說 pod 的價值(對背包來說物品的價值)，以多維背包策略修改 Scheduler 的節點分配方法，並透過叢集監控系統顯示的資源運用訊息評估效能的改善程度。

研究步驟

1. 閱讀相關文獻與資料

   閱讀改善容器叢集資源配置相關文獻，如(三)文獻回顧與探討所提到之三篇文獻。查詢容器叢集概念、操作與應用方面文章，以掌握相關知識。

2. 安裝環境，熟悉軟體功能及操作

   以多台異質主機架起 Kubernetes 環境，再安裝資源監控相關軟體。熟悉如何部屬提供 Hadoop 服務的容器。目前已架構好 Kubernetes，並運行簡單的測試容器，如下圖 3

```
root@blade03:/home/slytherin# kubectl get pods --all-namespaces
NAMESPACE     NAME                              READY   STATUS      RESTARTS   AGE
default       busy-sleep-less                   1/1     Running     9          155m
default       busybox                           0/1     Completed   0          167m
default       busybox-sleep                     1/1     Running     0          155m
default       myweb                             2/2     Running     0          165m
default       nginx                             1/1     Running     0          37m
default       nginx-54458cd494-cw7rg            1/1     Running     0          3h4m
default       nginx-54458cd494-vnl2t            1/1     Running     0          3h4m
default       nginx-app-6674df7c66-fv6rf        1/1     Running     0          3h2m
```

圖 3 – 在 Kubernetes 架構上運行容器

3. 以多維背包問題解決策略開發自訂義 Kubernetes Scheduler

   學習如何撰寫 Scheduler 程式，完成解決配置策略之演算法，將演算法實現在 Scheduler 程式上。

4. 建置實驗環境

確定實驗主機架設完成，將自訂 Scheduler 部屬到 Kubernetes 上，部屬實驗用的測試程式，並設定使用自訂 Scheduler。

5. 進行系統效能評估實驗

以資源監控程式輸出的結果分析與預設 Scheduler 的差異，調整 Scheduler 程式的參數，找出最適合的配置方法。

6. 實驗數據分析並結論

將最終的實驗數據做詳細的分析比對，總結實驗的成果與心得。

7. 撰寫實驗報告

將完成所有研究步驟所得到的收穫與結果，轉化為詳細的報告。

實驗環境

本計畫實驗環境所使用之各軟體套件版本如下表所示。

表 1 軟體環境

軟體名稱	***版本號***
Ubuntu	16.04
Docker CE	18.06.1
Kubernetes	1.13.3
Hadoop	2.7.3
Hbase	2.1.2
Java	8
Heapster	1.5.4
InfluxDB	1.7.4
grafana	6.0.0

實驗環境預計採用 7 台不同的實體機器以呈現異質性環境，如下表 2 所示。

表 2 硬體環境

PC Name	***Processor***	***CPU Cores***	***Memory***	***Storage***
HP01	Intel(R) Xeon(R) CPU E5520 @2.27GHz	16	36 GB	146+146 GB
HP02	Intel(R) Xeon(R) CPU E5520 @2.27GHz	16	30 GB	146+146 GB
HP03	Intel(R) Xeon(R) CPU E5520 @2.27GHz	16	30 GB	146+146 GB
HP04	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz	24	32 GB	300 GB
PC01	Intel(R) Core(TM) i7-6700HQ CPU @2.60GHz	4	16 GB	128 GB
PC02	Intel(R) Core(TM) i7-8750H CPU @2.20GHz	6	8 GB	128 GB
PC03	Intel(R) Core(TM) i5-6200U CPU @2.30GHz	4	8 GB	256 GB

實驗節點整體系統架構如下圖 3 所示。

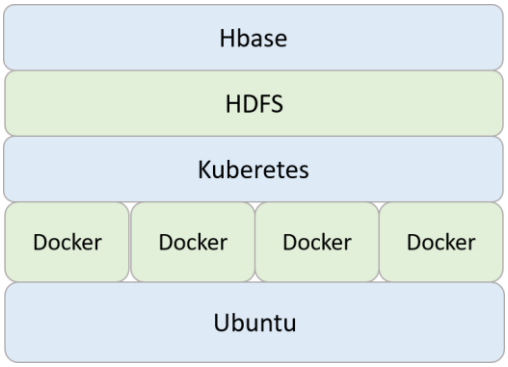

圖 3 整體系統架構

執行時間甘特圖

	七月	八月	九月	十月	十一月	十二月	一月	二月
閱讀相關文獻與資料								
安裝環境，熟悉軟體功能及操作								
以多維背包問題解決策略開發自訂義 Scheduler								
建置實驗環境								
進行系統效能評估實驗								
實驗數據分析並結論								
撰寫實驗報告								

(五) 預期結果

1. 完成容器叢集系統架設
   - 學會架設 Kubernetes 叢集系統，並熟悉操作 Master Node 進行容器的管理與節點的監控，能理容器運行狀態資訊如何表示，學會查詢節點提供的硬體資源種類及數目。
   - 學會建置用來提供 Hadoop Master 與 Slave 服務的 Docker Image，並將其部屬到 Kubernetes 系統上運行。
   - 使用開源軟體建構 Kubernetes 叢集用的監控系統，並能生成圖表表示資源的使用情況。
2. Scheduler 開發並用實驗結果調整參數
   - 設計能考量節點資源與節點需求配合度的多維背包策略。
   - 了解 Kubernetes Scheduler 內部程式碼寫法，以新策略開發自訂 Scheduler 進行替換。
   - 設計擁有不同資源需求的 Hadoop 服務，並運行在以多個異質電腦作為節點的 Kubernetes 叢集系統上。
   - 學會在容器運行時透過監控系統分析資源使用的狀況，能了解修改配置策略的方向，對 Scheduler 進行調整直到出現最佳配置結果。
3. 實驗結果分析與探討
   - 與系統預設調度程式所耗費時間、效能、所耗費能源等進行比較，歸納出詳細的結論，撰寫實驗報告。
   - 了解此策略的優勢與適合使用此策略的架構環境與服務特性，探討實務上的應用。

(六) 參考文獻

[1] Yuchao Zhang, Yusen Li, Ke Xu1, Dan Wang, Minghui Li, Xuan Cao and Qingqing Liang, “A Communication-aware Container Re-distribution Approach for High Performance VNFs,” 2017 IEEE 37th International Conference on Distributed Computing Systems

[2] Christophe Cerin, Tarek Menouer, Walid Saad and Wiem Ben Abdallah, “A New Docker Swarm Scheduling Strategy,” 2017 IEEE 7th International Symposium on Cloud and Service Computing

[3] Ying Mao, Jenna Oak, Anthony Pompili, Daniel Beer and Tao Han Peizhao Hu , “DRAPS: Dynamic and Resource-Aware Placement Scheme for Docker Containers in a Heterogeneous Cluster,” 2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)

[4] Ye Tao, Xiaodong Wang, Xiaowei Xu and Yinong Chen, “Dynamic Resource Allocation Algorithm for Container-based Service Computing,” 2017 IEEE 13th International Symposium on Autonomous Decentralized Systems

[5] Uchechukwu Awada and Adam Barker, “Improving Resource Efficiency of Container-instance Clusters on Clouds,” 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

[6] https://kubernetes.io/

[7] https://www.docker.com/

[8] https://hadoop.apache.org/

[9] https://github.com/google/cadvisor

[10] https://github.com/kubernetes-retired/heapster

[11] https://www.influxdata.com/

[12] https://grafana.com/

（七）需要指導教授指導內容

- 實驗相關硬體設備提供

  實驗需要多台異質主機來做為叢集架構的節點，因此需要指導教授提供實驗室中多台不同的主機做為硬體環境。

- 相關演算法的討論與設計

  修改調度方法過程中可能使用多種策略來解決多維多背包問題，例如線性回歸、機器學習等，此方面須指導教授提供協助與建議。

- 資源監控結果的判斷

  資源監控結果由巨量的數據組成，圖表輸出之後還需要利用相關知識進行分析，需要指導教授協助討論分析的正確與否，以及參數修改的方向。

- 實驗步驟與實驗結果的分析與討論

  驗證實驗步驟的正確性，是否實驗中有不合理步驟將影響實驗結果。學生對實驗結果的分析是否正確，結論是否合理等。