國立交通大學

網路工程研究所

碩 士 論 文

考量最低能源消耗之雲端資料中心
動態資源分配演算法

A Dynamic Resource Allocation Algorithm with Minimum

Energy Consumption for Cloud Data Centers

研 究 生：連懷恩

指導教授：王國禎　教授

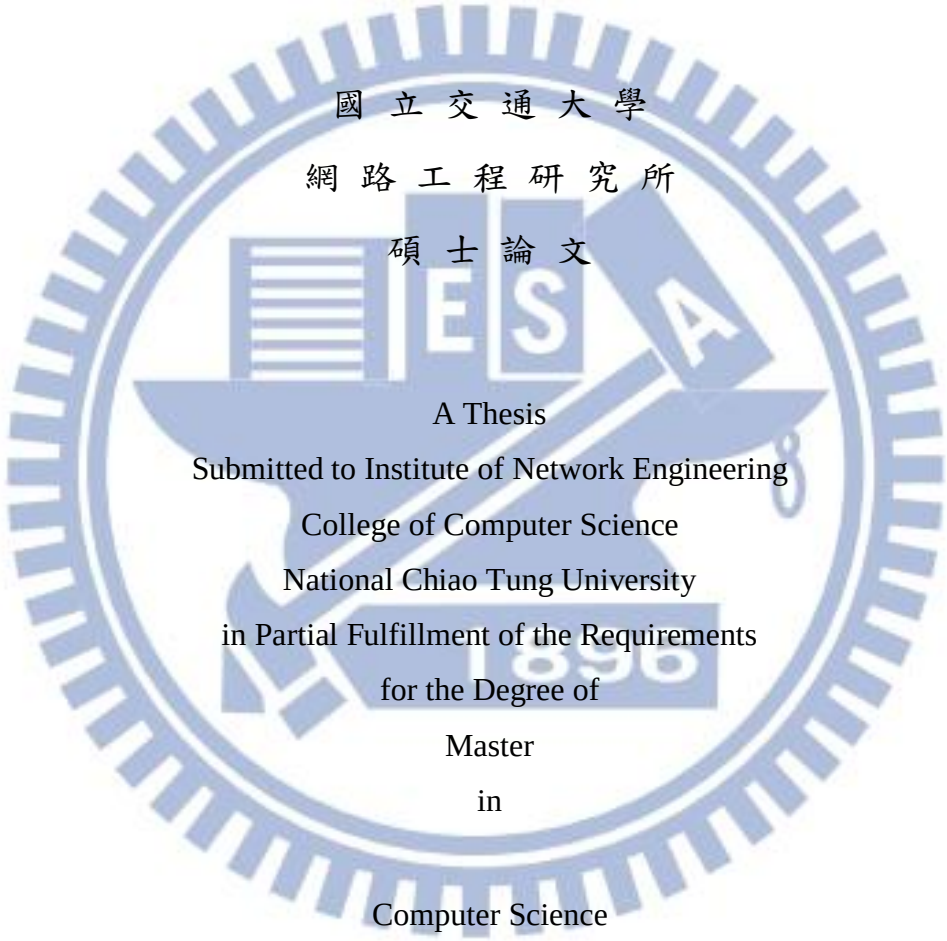
中 華 民 國　１０２　年　７　月

考量最低能源消耗之雲端資料中心動態資源分配演算法

A Dynamic Resource Allocation Algorithm with Minimum Energy Consumption for Cloud Data Centers

研 究 生：連懷恩　　Student：Huai-En Lien

指導教授：王國禎　　Advisor：KuoChen Wang



國 立 交 通 大 學

網 路 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Network Engineering

College of Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

July 2013

Hsinchu, Taiwan, Republic of China

中華民國 102 年 7 月

考量最低能源消耗之雲端資料中心

動態資源分配演算法

學生：連懷恩　　指導教授：王國禎 博士

國立交通大學網路工程研究所

摘 要

隨著越來越多的公司將資訊服務轉移到公有雲資料中心，如何在滿足各個應用程式不斷變動的資源需求下，在雲端資料中心達成省電節能的資源分配，已經成為一個相當吸引人的問題。許多動態資源分配法被提出來解決這些問題，但他們大部分都缺少像同時調整 VM 及 server 分配量、橫跨時間軸上的最佳化、及最佳解演算法這些嚴重影響能源效率的因素。在這篇論文中，我們提出了一個新的應用程式層級的動態資源分配演算法稱作 *Time-Directed Dijkstra* (*TD-D*)，它可以在現有的負載預測機制幫助下，藉由達成運轉耗電(operating cost)及開關耗電(switching cost)間的平衡來達成最低的能源消耗。我們主要的貢獻如下：(1)我們分類並整理了影響資源分配演算法能

源效率的主要因素。(2)我們提出了一個最低能源消耗的資源分配演算法，它可以使用市面上常見的機器，在合理的時間內算出最佳解。(3)我們也驗證了這個新演算法即使在負載預測發生錯誤時，依然可以保持可靠(低資源分配不足率，low resource under-allocation rate)和有效(好的能源效率)。我們的模擬結果顯示，這個新的最佳解演算法可以比其他現有的代表性近似解演算法(local search)耗用更少的運算時間並省下平均 9.5%的能源消耗。除此之外，我們的演算法在使用有預測錯誤的負載預測資料的情況下(7 個單位時間後產生 25%錯誤率)，仍然可以比現有的近似解演算法省下平均 7.1%的能源消耗，並保持很低的資源分配不足(under-allocation)率(在 1 個單位時間內，每個應用程式平均 0.03 台 VM)。

關鍵詞：應用程式層級、雲端資料中心、動態資源分配、能源消耗。

A Dynamic Resource Allocation Algorithm with Minimum Energy Consumption for Cloud Data Centers

Student: Huai-En Lien Advisor: Dr. Kuochen Wang

Institute of Network Engineering

National Chiao Tung University

Abstract

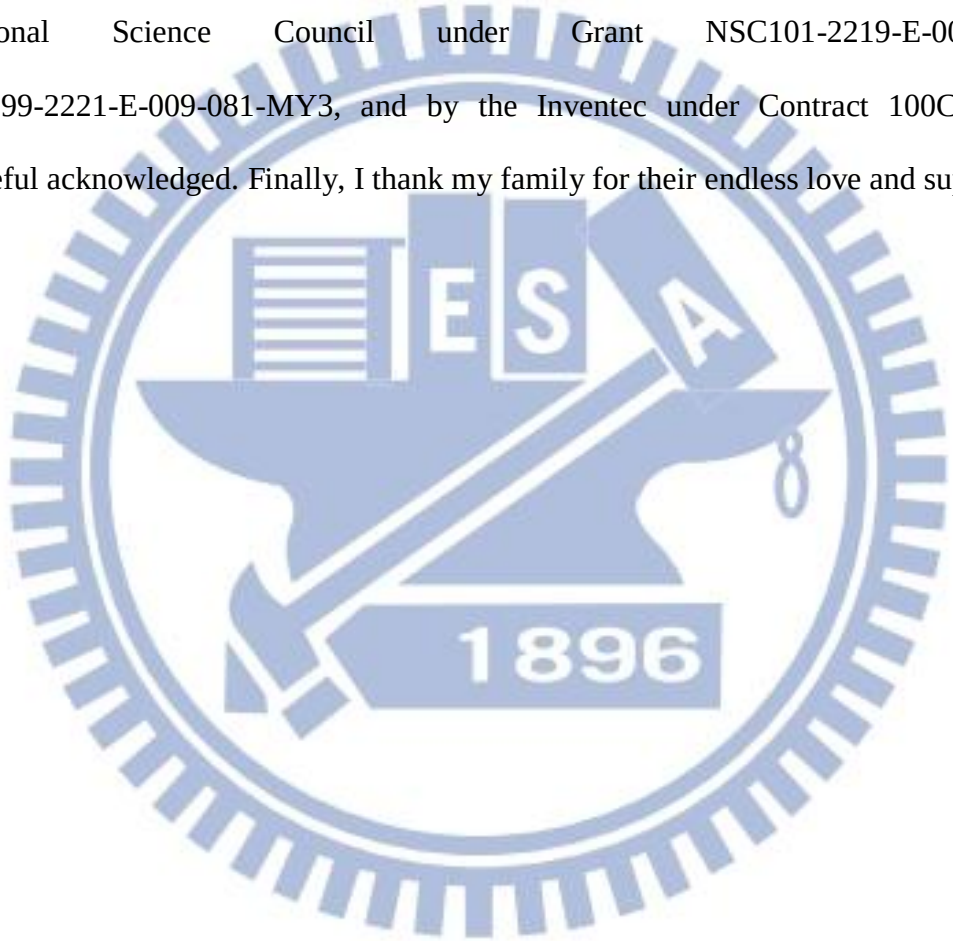
As more and more companies outsource their information services to public cloud data centers, how to perform dynamic resource allocation efficiently to reduce energy consumption while fulfilling each application's fluctuating resource demands has become a very challenging task. Many dynamic resource allocation approaches were proposed to tackle this task, but most of them lack for some influencing factors that may impact the energy efficiency, such as resizing at both VM and server levels, optimization over time horizon, and the optimality of the algorithm. With the help of existing load prediction techniques, in this paper, we design an application-level dynamic resource allocation algorithm, called *Time-Directed Dijkstra* (*TD-D*), which can achieve minimum energy consumption, by seeking the best trade-off between operating cost and switching cost due to switching on/off resources. The main contributions of this paper are as follows. (1) We analyze and categorize the most influencing factors that should be addressed in order to build an application-level

energy efficient resource allocation algorithm. (2) We develop a minimum energy consumption algorithm that can produce an optimal solution within reasonable computing time using commodity machines. (3) We demonstrate the proposed *TD-D* algorithm is fairly robust in terms of low resource under-allocation rate and energy-efficient to prediction errors. Simulation results show that, our optimal *TD-D* algorithm can save 9.5% of energy consumption in average using error-free workload data compared with a representative best-effort algorithm (local search), and consume much less computing time compared with the representative algorithm. In addition, using workload data with 25% prediction error after a prediction window of 7 time slots, our *TD-D* algorithm can save 7.1% of energy consumption than the representative algorithm and keep a very low resource under-allocation rate (0.03 VM / (time slot × application)).

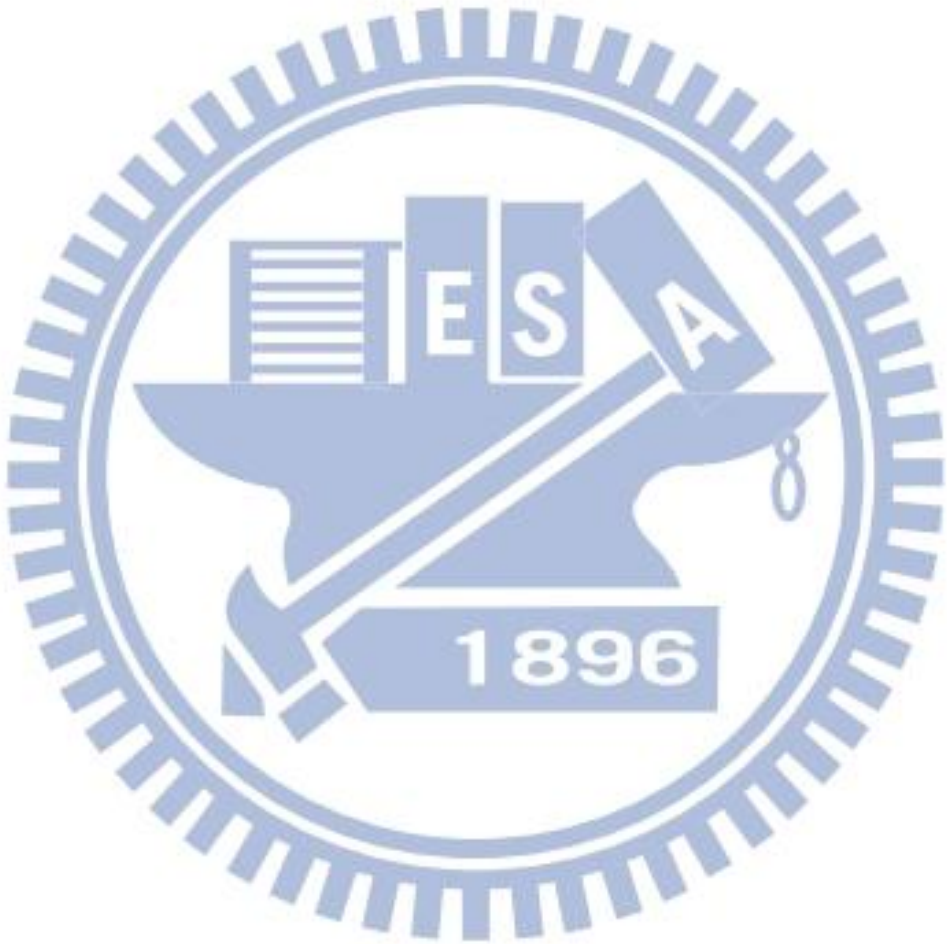
Keywords: application-level; cloud data center; dynamic resource allocation; energy consumption

Acknowledgements

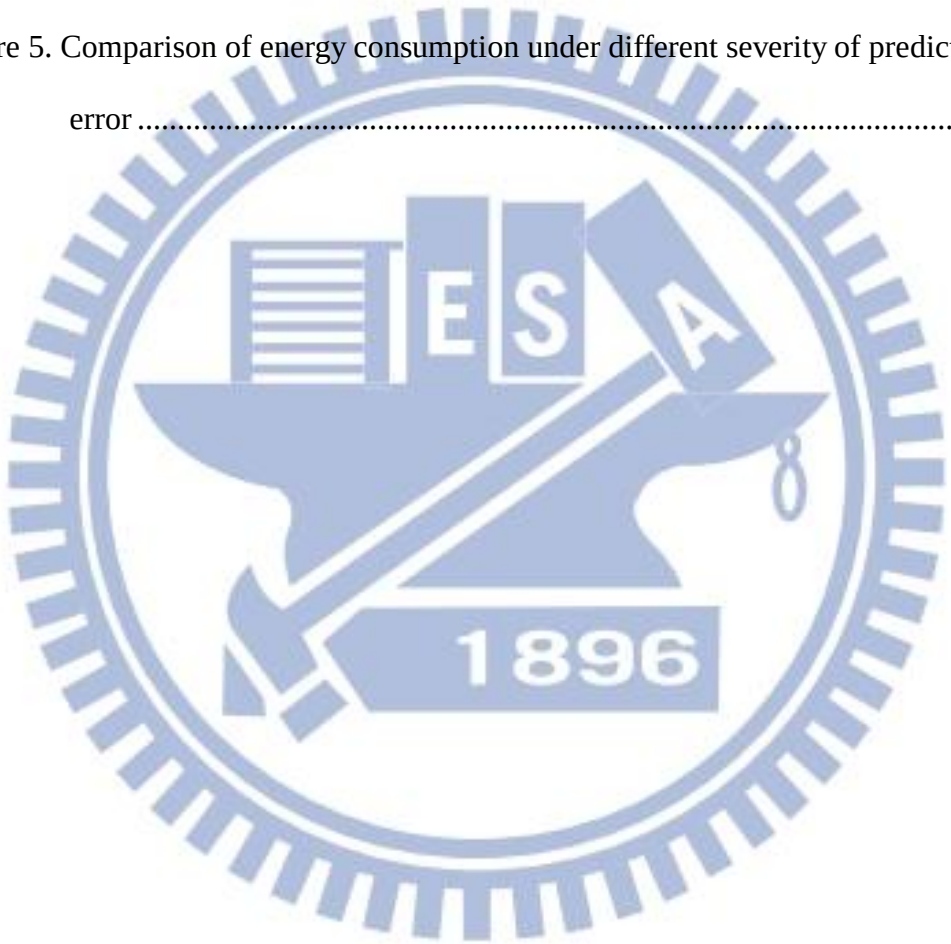
Many people have helped me with this thesis. I deeply appreciate my thesis advisor, Dr. Kuochen Wang, for his intensive advice and guidance. I would like to thank all the members of the *Mobile Computing and Broadband Networking Laboratory* (MBL) for their invaluable assistance and suggestions. The support by the National Science Council under Grant NSC101-2219-E-009-001, NSC99-2221-E-009-081-MY3, and by the Inventec under Contract 100C202 is grateful acknowledged. Finally, I thank my family for their endless love and support.



Contents



List of Figures



List of Tables

Chapter 1

Introduction

Energy cost takes a significant fraction of budget in public cloud data centers and this cost is expected to grow as the scale of cloud data centers and the price of energy are increasing in coming years. If a data center can adjust its resource provisioning in a more precise way, it can not only reduce unnecessary energy consumption, but also accommodate more applications and more profit can be made. Hence, there is a growing demand to improve the energy efficiency of cloud data centers. Many studies of auto-scaling and resource resizing for data centers have been published to respond to such a challenge. By the help of existing load prediction techniques [1, 2, 3, 4], such precise auto-scaling is possible.

However, even adopt the same state-of-the-art load prediction technique, different resource allocation approaches may have different ways to use these predicted workload data and concern different factors, thus having diverse scenarios and influencing results of energy saving. For example, M. Lin et al. [5] used a load prediction technique to predict the future workload of the whole data center, and resizes the number of active servers accordingly. Such a scenario is suitable for a private data center, but is not suitable for a public cloud data center since no application-level workload prediction and no virtualized resources allocation were concerned. In contrast, C. Tang et al. [6] allocates virtualized resources to each application, but it does not perform well in energy saving due to the lack of server level resizing. D. Ardagna et al. [9] and V. Petrucci et al. [10] concerned the switching cost due to switching on/off resources to further improve energy efficiency. However,

many of these approaches only concern the resource allocation for the next time slot, not the optimization over time horizon. A switched off resource may need to be switched on again quickly, and therefore not much operating cost is saved. Finally, since the complexities of such data center optimization problems are usually very high, most of the existing approaches [6, 7, 8, 9] only provide best-effort algorithms, which usually cannot promise the quality of the solutions and their performance are highly correlated to the implementations. To cope with these problems, in this paper, we propose a minimum energy consumption resource allocation algorithm called *Time-Directed Dijkstra* (*TD-D*). In this algorithm, we use predicted application-level workload data from a load prediction module to construct the solution space throughout a prediction window of several time slots. Then we apply a level-by-level, time-directed minimum cost path search, which is actually a Dijkstra shortest path search on a time-directed graph, to find the optimal solution. The optimal solution itself involves a best trade-off between operating cost and switching cost due to switching on/off servers and VMs so that it is the resource allocation with minimum energy consumption. Simulation results show that our algorithm can save 9.5% of energy consumption in average using error-free workload data compared with a representative best-effort dynamic resource allocation algorithm, and consume much less computing time compared with the representative approach. We also evaluate scenarios with prediction error, and the results showed that our algorithm is fairly robust in terms of low resource under-allocation rate and energy-efficient to prediction error. Last but not least, while achieving the minimum energy consumption, our algorithm can keep the resource under–allocation rate in a very low level, which is usually the weakness of many existing energy efficient approaches since energy efficiency and resource under-allocation rate are two trade-off parameters.

The rest of the paper is organized as follows. In Chapter 2, we discuss the background and categorize related work on dynamic resource allocation for cloud data centers. In Chapter 3, we give our problem formulation and discuss the difficulties of resizing VMs and servers at the same time. In Chapter 4, we describe our *Time-Directed Dijkstra* algorithm and system architecture. In Chapter 5, we list experiment settings and evaluate experiment results. Finally, in Chapter 6, we conclude this paper and outline future work.

Chapter 2

Background and Related Work

To design an energy efficient resource allocation algorithm for cloud data centers, one should consider the following factors: application-level resource allocation, resizing at both VM and server levels, switching cost, optimization over time horizon, and optimality of the algorithm.

2.1 Application-level Resource Allocation

Many researches on the resource resizing problem in data centers have an assumption that there is a load prediction module we can exploit. A recently published approach [5] uses predicted total workload of the whole data center as an input and then derives the number of active servers accordingly. This is fine for a private data center, but is not suitable for a data center which intends to provide public cloud services. It assumes that the workload from different applications can be aggregated to one machine under one OS. This could lead to the following three consequences. First, it may be less accurate in predicting the total workload for the entire data center, since the total workload comprises the workload from different types of applications. It is conceivable that different types of applications may have very different workload patterns. Therefore considering the total workload of the entire data center is not a good idea. Second, for a cloud data center, different applications may come from different subscribers or tenants, and from subscribers' point of view, it is unacceptable that one subscriber/tenant has to share the same VM with another subscriber/tenant

due to security concern and accounting. It is better to allocate dedicated VMs to each subscriber/tenant. Third, it is desirable to provide customized or graded accounting for different QoS levels and prices based on each subscriber/tenant's needs and budget. To do so, we need to distinguish the workload from different applications and then allocate dedicated VMs accordingly.

2.2 **Resizing at both VM and Server Levels**

Some of the existing researches like [5], [6] only consider the resizing problem either at server level or at VM level. However, to reach the minimum energy consumption, resizing at both levels are necessary. It is easy to understand that if we only resize one kind of resource, the other one may incur unnecessary energy consumption. Since the number of active servers is closely related to the number of running VMs, we have to resize at both levels at the same time. In the next chapter we will discuss their relationship and show the difficulties if we want to resize at both levels at the same time to attain minimum energy consumption.

2.3 **Switching Cost**

Any manipulation of adding/removing VMs and switching on/off servers will incur switching cost. The switching cost mainly refers to the power consumed due to switching, but it can also include wear-and-tear overhead and performance delay due to switching. A good resource allocation algorithm should consider the trade-off between operating cost (the energy cost to keep resources operating) and switching cost.

2.4 **Optimization over Time Horizon**

Many approaches like [6, 8, 9, 10] use a load prediction module to predict workload in the next time slot and allocate resources accordingly. Some of them like [9], [10] even consider switching cost to further improve energy efficiency. Although these approaches are effective in their own scenarios, when we look for a resource allocation with minimum energy consumption that crosses several time slots like [5] and [7], these approaches may not provide a good solution. They did not utilize predicted workload that are across several time slots later, which many existing load prediction techniques like [1], [2] can offer. Besides, if we only conduct resource allocation in the next time slot, some resources being switched off may need to be switched on again due to fluctuating resource demand, thus causing a lot of unnecessary switching cost. As a result, a resource allocation approach that provides optimization over time horizon will usually outperform the one that only deals with resource allocation for the next time slot.

2.5 **Optimality of the Algorithm**

Most of the optimization problems in cloud data centers are formulated as integer programming or mixed integer programming problems, which are mostly NP-hard problems. The complexity may be even higher if we want to include all above-mentioned factors. As a result, many approaches use approximation algorithms like neural network [7] or local search [7], [9] to obtain best-effort solutions. In addition to the worse energy saving in these best-effort solutions, the major problem is that we usually cannot guarantee the quality of these solutions, or so-called approximation ratio. Another problem of these approximation algorithms is that their performance is usually highly correlated to the implementation, like the hidden layer

used in a neural network or the initial value of a local search. All of these make these approximation algorithms unstable.

2.6 **Other Factors and Comparison Table**

Some paper may concern other factors such as dynamic voltage frequency scaling (DVFS). However, some papers like [7] have pointed out that such technique has little effect on energy saving compared with resizing the number of active servers. Another reason that we do not use such a technique is that as number of cores increases within a CPU, such a scaling may incur more VM migrations due to consolidating VMs that are working in the same frequency to the same destination machine. A qualitative comparison between the above-mentioned related work including the proposed TD-D is shown in Table 1.

Table 1. Comparison of different resource allocation algorithms for cloud data centers

Approach	**Resizing VMs**	**Resizing servers**	**Application-level resource allocation**	**Switching cost**	**Optimization over time horizon**	**Optimality**
Minghong Lin et al. [5]	No	Yes	No	Yes	Yes	Optimal
Chunqiang Tang et al. [6]	Yes	No	Yes	No	No	Best-effort
Dara Kusic et al. [7]	Yes	Yes	Yes	Yes	Yes	Best-effort
Anton Beloglazov et al. [8]	Yes	Yes	Yes	No	No	Best-effort
Danilo Ardagna et al. [9]	Yes	Yes	Yes	Yes	No	Best-effort
Norman Bobroff et al. [10]	Yes	Yes	Yes	No	Yes	Best-effort
Vinicius Petrucci et al. [11]	Yes	Yes	Yes	Yes	No	Optimal
Time-Directed Dijkstra (TD-D, proposed)	Yes	Yes	Yes	Yes	Yes	Optimal

Chapter 3

Problem Formulation

In this chapter we describe the energy consumption model, the cost function and the problem formulation of our minimum energy consumption resource allocation algorithm.

3.1 Energy Consumption Model and Cost Function

To solve the minimum energy consumption problem, we transform it into an optimization problem and build a cost function to minimize the cost of cloud data centers. To simplify our work when doing the auto-scaling on both server and VM level, we use a homogeneous model that all servers and VMs are of same capacities and energy consumption. This assumption makes our work as a simple discrete packing problem that all servers have equal number of slots to host VMs and each VM has equal capacity to handle requests. Now, we need an energy consumption model for a running server. We refer to the energy consumption model introduced in [11], which is

$$P(u) = P_{idle} + (P_{busy} - P_{idle}) \times u \quad (1)$$

Here, the energy consumption of a running server is shown as a linear equation, where P_{idle} is the basic energy consumption when the server is idle, P_{busy} is the augmented energy consumption when the server is at 100% utilization, and u is the CPU utilization which falls into the interval of [0, 1]. We modify the linear model of (1) and give a discrete step function version:

$$P(n) = P_{server} + P_{VM} \times n \quad (2)$$

where n is the number of VMs that currently run on that server, P_{server} is the basic energy consumption of an idle server like P_{idle}, and P_{VM} is the augmented energy consumption for each VM. The transformation from (1) to (2) implies every VM is fully loaded and such an implication is feasible since an auto-scaling algorithm always resize its resource provisioning closely to its actual demand.

Now we can build up our cost function. Table 2 lists the symbols used in our cost function. There are two kinds of cost, the *operating cost* and the *switching cost*. *Operating cost* is the energy consumption of running servers and VMs, while *switching cost* includes the cost to switch on/off servers and startup/shutdown VMs. For simplicity, we let the switching cost of switching on and switching off a particular resource be the same value. We can put *operating cost* and *switching cost* together in one cost function by introducing the break-even time parameter. The *break-even time* is the time period that the *operating cost* of a particular resource equals to the *switching cost* of that resource multiplied by two. For example, for a server, it will be

$$P_{server} \times \Delta_{server} = \delta_{server} \times 2 \tag{3}$$

where Δ_{server} is the break-even time of a server and δ_{server} is the *switching cost* of a server. For a VM, we can also use the same idea of break-even time and have a similar relation:

$$P_{VM} \times \Delta_{VM} = \delta_{VM} \times 2 \tag{4}$$

Assume that there are N apps, and the prediction window size is W time slots, then we can build our cost function as follows:

Minimize

$$P_{VM} \times \sum_{t=0}^{W} \sum_{i=1}^{N} v_{i,t} + P_{server} \times \sum_{t=0}^{W} s_t + \delta_{VM} \times \sum_{t=0}^{W-1} \sum_{i=1}^{N} \left| x_{i,t+1} - x_{i,t} \right| + \delta_{server} \times \sum_{t=0}^{W-1} \left| s_{t+1} - s_t \right| \tag{5}$$

Subject to

$$v_{i,t} \geq d_{i,t}, \forall i \in \{1,2,...,N\}, \forall t \in \{0,1,2...,W\} \quad (6)$$

$$\sum_{i=1}^{N} v_{i,t} \leq capacity \times s_t, \forall t \in \{0,1,2,...,W\} \quad (7)$$

The cost function (5) is simply the summation of all *operating cost* and *switching cost* over W+1 time slots. Restrictions (6) and (7) restrict that the allocated VMs must be sufficient for minimum resource demand of each application and the number of active servers must be able to host all VMs at any moment. We require that a time slot in our system must be long enough to perform any desired resource rearrangement manipulations (server switch on/off, VM startup/shutdown). We also assume we have a workload predictor deployed in our system so that we can get the predicted workload of each application up to several time slots later.

Table 2. Symbols used in the cost function

Variable	**Definition**
$v_{i,t}$	The number of VMs allocated to the *i*th app in time slot *t*
s_t	The number of active servers in time slot *t*
$d_{i,t}$	The predicted resource demand (in terms of number of required VMs) of *i*th app in time slot *t*
Constant	**Definition**
P_{server}	Operating cost of a single idle server in one time slot
P_{VM}	Operating cost of a single VM in one time slot
δ_{server}	Switching cost to switch on/off a single server
δ_{VM}	Switching cost to switch on/off a single VM
C	Each server can host at most C VMs

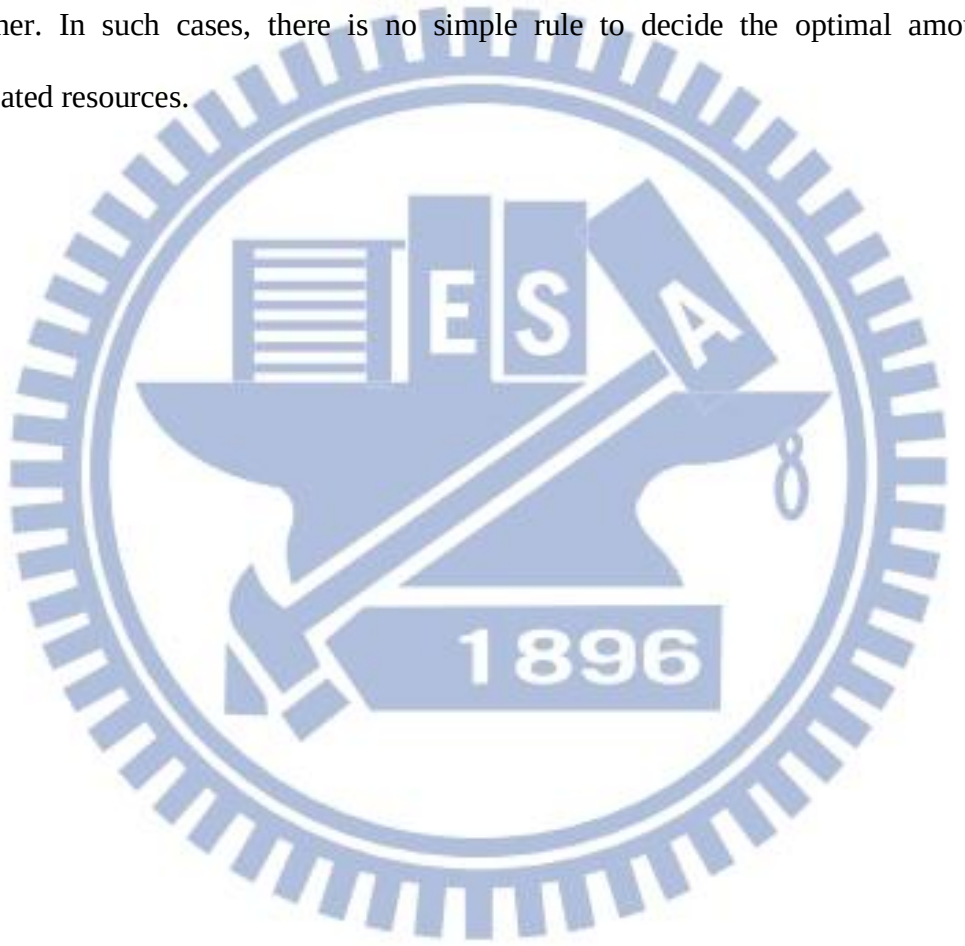
3.2 **Difficulties of Resizing at Both VM and Server Levels**

In most cases, such a discrete resizing problem is relatively easy if we have predicted workload data. Intuitively, to achieve the minimum energy consumption between *operating cost* and *switching cost*, we can allocate resources to exactly match the predicted resource demand if the demand is rising or no change, compared with the demand in the previous time slot. If the demand is falling, we check if the demand will return to the previous level within a break-even time or not to decide to keep or switch off the resource. But the problem becomes complicated if we want to resize at both VM and server levels. The problem is that the amount of allocated VMs will affect the amount of servers that need to be allocated. Keeping an unnecessary VM who will be needed within a break-even time will not always bring the minimum energy consumption. We give two examples here. In the first example, in most cases, when there is a VM break-even time event, or we say the demand of that VM goes down then back within a break-even time, we keep that VM. However, if closing that VM may help to switch off the host server, and if the total resource demand of the data center goes down in the time slot that the VM is again to be needed, closing that VM may be a better choice since we can start that VM in other server and thus saving the operating cost of the original host server. We can use (8) to express such relation:

$$\delta_{VM} \times 2 < (P_{VM} + P_{server}) \times \Delta_{VM} \tag{8}$$

In (8), the left-hand side is the cost to release a VM, and the right-hand side is the cost to keep that VM. Note that in the right-hand side, we don't have the *switching cost* of the host server since no matter we decide to keep that VM or not, the host server is doomed to be shut down in both sides. In the second example, when we keep a

temporarily unnecessary VM and meanwhile, some new VMs from other applications are about to be activated, we may need to switch on a new server to support the increasing capacity demand. In fact, things are usually more complicated than the above examples, if the break-even time period is long, and several overlapped but not simultaneous VM break-even time events occur. In this case, it forms a chain of break-event time events among several applications, and these events may affect one another. In such cases, there is no simple rule to decide the optimal amount of allocated resources.



Chapter 4

Proposed Time-Directed Dijkstra Algorithm

In this chapter, we introduce the proposed resizing algorithm called *Time-Directed Dijkstra* (*TD-D*), which can help us reach the minimum energy consumption by considering both *operating cost* and *switching cost*. That is, find a minimum cost solution of equation (5) in Chapter 3.

4.1 Preliminaries of the Algorithm

Since in a resizing problem we don't consider the placement of resource, we simplify (5) to (N+1) kinds of resources in all W time slots. The (N+1) kinds of resources include the number of allocated VMs for N apps, plus the number of active servers.

First we define a new term called *number-of-combinations*. It gives a best known tight upper-bound of the optimal resource provision, which means the optimal number of allocated VMs for a particular application, or the optimal number of active servers, in a particular time slot. It can be defined as follows:

$$\text{minimum resource demand} \leq$$

$$\text{optimal resource provision} \leq$$

$$\text{minimum resource demand} + \textit{number-of-combinations} - 1 \qquad (9)$$

That is, assuming the minimum resource demand of a particular resource in a particular time slot is *m*, then the optimal resource provision must be an element of a *resource sequence* {*m*, *m* + 1, *m* + 2, … , *m* + *number-of-combinations* - 1}. From the description in Chapter 3, we know the *number-of-combinations* will not be 1 only when there is a break-even time event of that kind of resource. To find the *number-of-combinations* for each kind of resource in each time slot, we examine if it happens a break-even time event. If it does, then we check every descending count of predicted resource demand to see if that resource demand will climb back to the previous level within the following break-even time. For example, we assume the resource demand for a particular application in a consecutive four time slots are 3, 1, 2, 3, and assuming Δ_{VM} is 1. From our definition, we can find the *number-of-combinations* of these consecutive four time slots are 1, 2, 1, 1. Note that the *number-of-combinations* in the second time slot is not 3 because the resource demand climbs back to 3 after the break-even time period. Since the resource demand of servers is affected by the number of allocated VMs, we need a two-pass scan to determine the *number-of-combinations* of each kind of resource. In the first pass, we determine the *number-of-combinations* of *N* apps in all *W* time slots. Since the maximum number of active servers must be enough to accommodate the maximum number of allocated VMs, in the second pass, we determine the *number-of-combinations* of server in all *W* time slots by the information of maximum possible VMs from the first pass. Once we complete the two-pass scan, we have (*N*+1) *number-of-combinations* in each time slot, and we can find the *resource combination set* in all *W* time slots. In each time slot, the *resource combination set* is a (*N*+1)-ary Cartesian product over (*N*+1) *resource sequences* in that time slot.

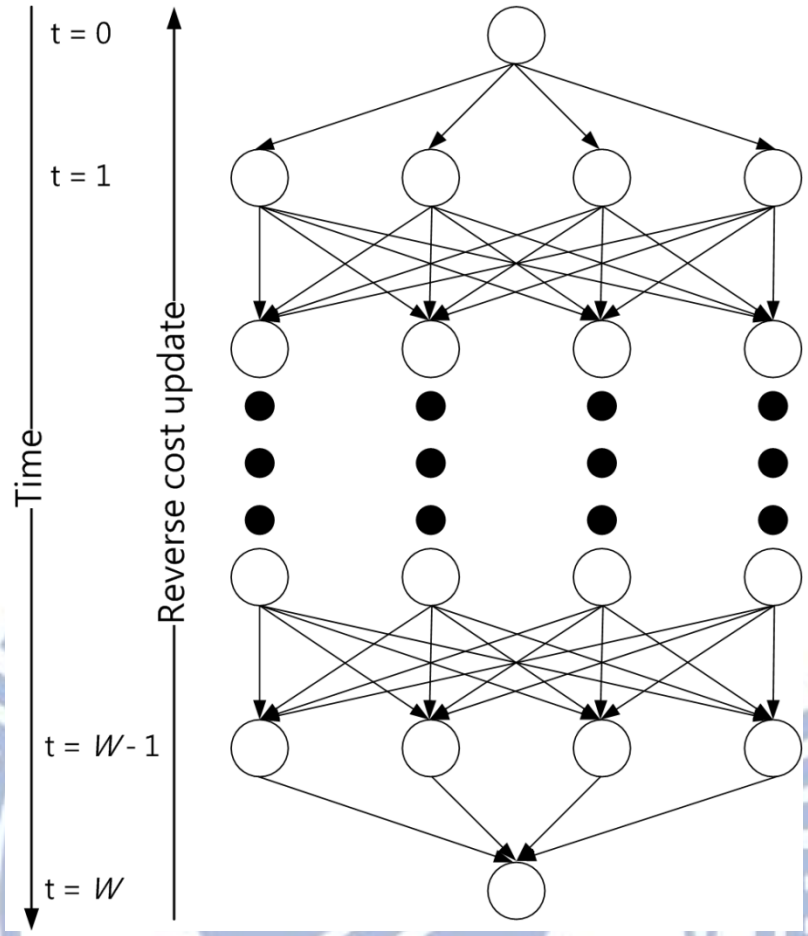


Figure 1. Illustration of the Time-Directed Dijkstra algorithm

Here we introduce the terminologies used in our *TD-D* Algorithm. As illustrated in Fig. 1, it can be understood as finding the minimum cost path, or the shortest path from the source node (the root node in the figure) to the destination node (the leaf node in the figure). The graph is composed of W+1 levels and there are edges connecting any two adjacent levels. The term "level" can be considered as "a group of nodes that resides at same time slot". Here each node stands for a resource allocation state in that time slot; that is, an element of a *resource combination set* in that time slot. An edge represents a switching process from one resource allocation state to another. Since a path can travel from any node in the upper level to any node in the lower level, any two adjacent levels form a complete bipartite graph. Clearly, the number of nodes in a particular level equals the cardinality of the *resource combination set* of that level. We

call it "*Time-Directed Dijkstra*" since it uses the same idea of Dijkstra's algorithm to find the minimum cost path (shortest path), however, with a few differences. First, it is a directed graph and each path can only be traversed along the time-axis. Second, unlike conventional weighted graphs, there are three types of weights (costs) used in this algorithm, and both nodes and edges are weighted. It is illustrated in Fig. 2. $Oper_x$ is the *operating cost* of a node *x*; $Switch_{xy}$ is the *switching cost* from node *x* to node *y*; and Low_x is the lower bound cost (*operating cost* plus *switching cost*) from node *x* to the leaf node. The *operating cost* of each node and the *switching cost* of each edge remain unchanged, while the lower bound cost will be updated throughout the algorithm.

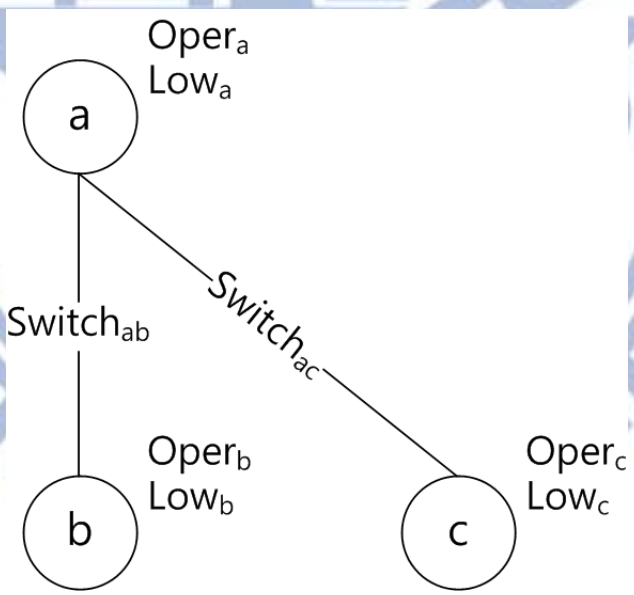


Figure 2. An example of deriving a minimum cost path

4.2 **Time-Directed Dijkstra Algorithm**

Now we introduce the whole process of our *Time-Directed Dijkstra* Algorithm (*TD-D*). First, we use the two-pass scan to obtain all *number-of-combinations* and the *resource combination set* in every time slot, thus producing all nodes in every level. Then we use a reverse update manner, that is, from the leaf level to the root level, we update the lower bound cost of every node. Again we use Fig. 2 for example, Low_a will be the min($Low_b + Switch_{ab}$, $Low_c + Switch_{ac}$), plus $Oper_a$. In this way, $node_a$ iteratively check all nodes in the next level, update the lower bound cost, and finally Low_a becomes the minimum cost from $node_a$ to the leaf node. The whole process is shown in Algorithm 1.

Algorithm 1 Time-Directed Dijkstra

1: **Two-pass scan** to generate all nodes in each level
2: **Set** $Low_{leaf} = Oper_{leaf}$ and $Low = \infty$ for other nodes
3: **for** $t = W - 1$ to 0 **do**
4: **for** i = 1 to cardinality of *resource combination set* at level = t **do**
5: **for** j = 1 to cardinality of *resource combination set* at level = $t + 1$ **do**
6: $Low_i = \min(Oper_i + Low_j + Switch_{ij}, Low_i)$
7: **Output** *Low* of root node and the corresponding minimum cost path

4.3 **Correctness and Complexity Issues**

For the correctness of the algorithm, we use Fig. 2 again as an example. In Fig. 2, it is easy to see that, if $node_a$ chooses $node_b$ as its best descendent toward the leaf node, then for one of the $node_a$'s ancestor node, named $node_d$, its best path through $node_a$ must be *d-a-b*, not *d-a-c*. The suggested best path obtained from the lower level

still works when we move on to the upper level. That is to say, when we iteratively update the lower bound cost of all nodes level by level, in a reverse manner, we always produce the minimum cost path to the leaf node. In fact, we can actually run this algorithm in the opposite direction, that is, update from the root node to the leaf node, and it will output the same result. The structure shown in Fig. 2 is a reversible structure.

The other thing is the complexity of this algorithm. We analyze it from two parts. First part is the size of the outer layer for loop, which is proportional to the prediction window size *W*. We can consider this part as a polynomial time complexity part. The other part is the two inner layer for loop. These two for loop should have the same complexity order, since they represent the average cardinality of *resource combination set* of all levels. The average cardinality of *resource combination set* is strongly dominated by the severity of workload fluctuation, the trend of workload, and the number of applications in data center. If many applications have smooth workload, or monotonic increasing or monotonic decreasing workload, the cardinality of *resource combination set* will be small since there are not too much break-even time events we should concern. But in the worst case, though it is almost impossible in the real world, that all applications experience break-even time events simultaneously and severely, then the *resource combination set* will be a very large set. The order of the average cardinality of *resource combination set* is *O*((average *number-of-combination* in a break-even time event) ^ (average number of break-even time events in a level)). It is exponential time complexity, but in our algorithm, we only consider those promising resource combinations, like break-even time events, making it still a practical approach. In our simulation, in almost all cases, we can finish our *Time-Directed Dijkstr*a Algorithm in three minutes, under the common scenario used in public cloud data centers.

4.4 System Architecture

The proposed *TD-D* algorithm is not a standalone component. It needs to work with other components to perform its function. Fig. 3 illustrates the system architecture for our *TD-D* algorithm and its workflow when time slot = *t*. First, as illustrated in (a), the *workload monitor* will monitor and record the workload from each application that running on computing cloud in real-time. Since the real-time monitored data is numerous and jumbled, the *workload monitor* will process them into ordered and usable statistics data, usually the peak workload values for each application in that time slot, and then sends the data to *reactive controller* and *workload predictor*, as illustrated in (b). In (c), once the *reactive controller* receives the monitored workload data from *workload monitor*, it will perform its only but essential function, that is, dynamically switched on VMs/servers if any resource under-allocation is detected. This is a critical function that a proactive, long-term controller like our *TD-D* algorithm will need, since there are always prediction error and resource under-allocation is inevitable. We put this reactive, or short-term controller into our system architecture as the auxiliary of our long-term, proactive controller, and the countermeasure to prediction error. Another component that uses the workload statistics data is the *workload predictor*. As mentioned in the previous chapter, we make use of existing workload prediction technique to deploy a *workload predictor* in our system, to provide workload prediction for the following *W* time slots for each application. Finally, as illustrated in (d), the *proactive controller* receives the updated prediction data from *workload predictor*, performs our *TD-D* algorithm, and then sends control messages to computing cloud to perform any desired resource reallocation, as illustrated in (e).

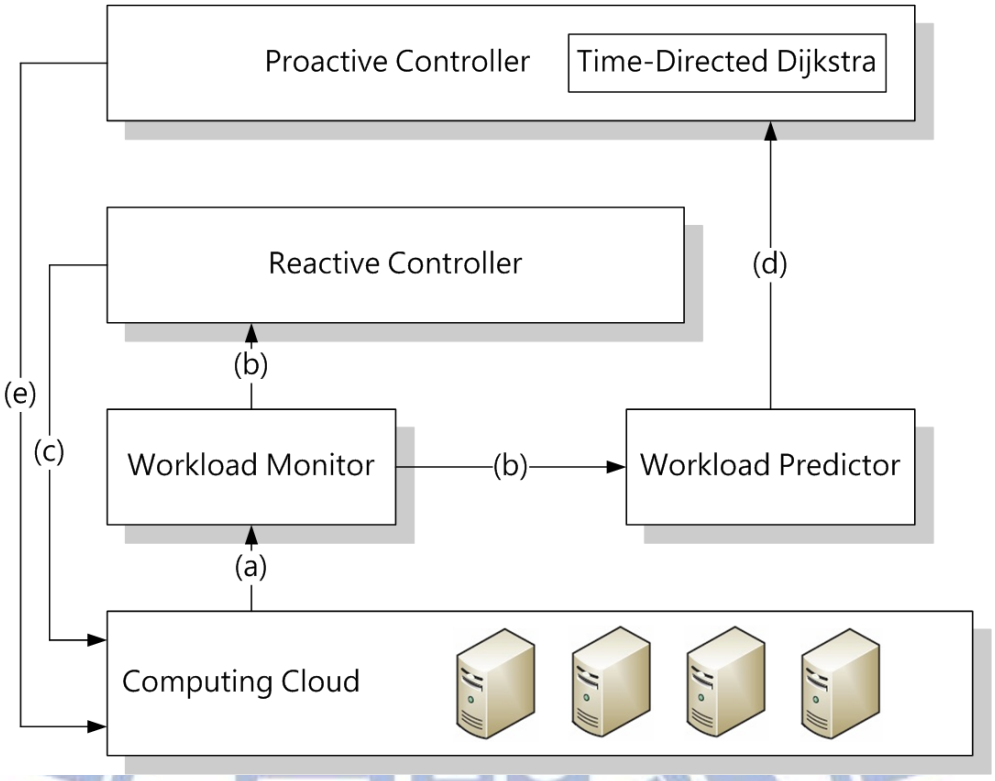


Figure 3. Resource management system architecture and its workflow, assuming time slot = *t*. (a) Real-time workload data gathered from the Computing Cloud. (b) Real-time workload statistics, usually using the peak workload within a time slot. (c) Dynamically switch on new VMs/servers if under-allocation is detected. (d) Provide updated workload prediction from $t + 1$ to $t + W$. (e) Perform dynamic resource allocation at the beginning of each time slot, according to the direction of the proposed Time-Directed Dijkstra algorithm.

Chapter 5

Evaluation

In this chapter, we introduce our experiment settings, the comparison approaches we used, the experimental results and discussion.

5.1 Experiment Settings

We build up a simulation environment to do our evaluation. First, we defined the parameters used in our evaluation, which are listed in Table 3. Note that the energy consumption unit we use here is a relative unit so there is no energy unit like joule or KW/hr. That is, we set P_{server}or P_{VM} = 1, then we get δ_{server} and δ_{VM} from the value of Δ_{server} and Δ_{VM} , by applying (3) and (4). The values of break-even time Δ_{server} and Δ_{VM} can be determined by measuring the energy consumption on real servers and VMs or determined by operator policy. We also use the energy consumption measurement data from [12] as our operating cost parameter. The peak energy consumption for a 2x Intel Xeon X5550 Quad core server is 248W, and the energy consumption is 149W in its idle state, by this we determine the ratio of P_{server} and P_{VM} . Another thing we should notice is that although (3) and (4) are in equation form, when in a practical use, we should add an extra tiny cost to both δ_{server} and δ_{VM} so that we will always try to keep the resource when the left-hand side are very close to the right-hand side in (3) and (4). The parameters used in the evaluation are listed in Table 3.

Then we implement a synthetic workload generator, which can provide the predicted workload of every app for the following *W* time slots. By intuition we may

Table 3. Parameter settings used in the evaluation

Parameter	Definition	Value
N	The number of applications	30
MAX_VM_APP	The maximum number of VMs which can be allocated to each application	10
NUM_SERVER	The number of available servers in the data center	ceiling(NUM_APP ×MAX_VM_APP / C)
W	Prediction window size	7 time slots
P_{server}	Same as defined in Table 2	9
P_{VM}	Same as defined in Table 2	1
Δ_{server}	Same as defined in (3)	3
Δ_{VM}	Same as defined in (4)	2
C	Same as defined in Table 2	6

think a workload generator that generates workload that follow a Gaussian distribution. But such workload generator mostly generates workload that vibrate along the mean value, but rarely the increasing or decreasing workload, which do happen in the real world data center at the transition time between rush-hour and off-hour. Here we implement a synthetic workload generator that using the discrete version of Gaussian Random Walk model. For each app, we see the predicted resource demand as a series of W non-negative integer random variables like $\{d_1, d_2, ..., d_W\}$, and these random variables form a time-homogeneous Markov chain. The term "Gaussian Random Walk" means for every step from d_t to d_{t+1}, $0 \le t < W$, $d_{t+1} = d_t + diff$, where $diff$ is an integer random variable that follow the same Gaussian distribution $N(0, \sigma^2)$. Note that since the number of VMs that allocated to an app should be ranged from 0 to MAX_VM_APP, we force any negative d to be 0 and any

d that lager than MAX_VM_APP to be MAX_VM_APP, making the state space of the Markov chain a closed communicating class. Besides, there are always prediction error in real world, so we implement Additive White Gaussian Noise (AWGN) to add prediction error into our predicted workload. The AWGN works similarly as Gaussian Random Walk, that they all apply a Gaussian distribution of the form $N(0,\sigma^2)$, so we can adjust the degree of workload fluctuation and the severity of prediction error by applying different variance values.

We categorize the approaches used in the evaluation into five classes, listed as follows:

(1) Resizing VM only, without break-even time: This class of approaches only do the VM resizing and not concern the break-even time, or we can say the switching cost. [6] can be categorized into this class.

(2) Resizing VM only, with break-even time: This class of approaches only do the VM resizing and using the rule of break-even time to balance the operating cost and switching cost.

(3) Resizing VM and server, without break-even time (on demand): This class of approaches do both the VM and server resizing, but not concern the break-even time, or we can say the switching cost. [8] and [13] can be categorized into this class.

(4) Resizing VM and server, with break-even time: This class of approaches do both the VM and server resizing, and using the rule of break-even time to balance the operating cost and switching cost. [7], [9], [10] can be categorized

into this class.

(5) The proposed *Time-Directed Dijkstra* algorithm.

Note that since every approach has their own scenario, here we assume that all approaches can do the time horizon optimization over W time slots, thus relax and improve some approaches. Another thing is that since the performance of approximation algorithm is highly correlated to the implementation, our simulation results may not reveal the true performance of those approaches that using the approximation algorithm. An approximation algorithm with good implementation can often get the solution very close to the optimal.

Finally, to do an objective and credible measurement on the computing time, in our evaluation, we implement our algorithm to a single-thread, single-process program, running on a Intel Core i5-2500 3.3GHz machine with 8GB RAM.

5.2 **Experiment Results and Discussion**

First we evaluate the performance of energy saving using error-free workload information. In Table 4, we compare our algorithm with other two approaches that only resizing VM. It can be easily understood that the first two approaches consume much more energy since they don't provide the server level resizing, and the basic energy consumption P_{server} takes a significant fraction of the overall energy consumption on a working server [7]. Next we compare our algorithm with other two that resizing at both VM and server levels. The result is illustrated in Fig. 4. We can easily observe that, as the degree of workload fluctuation increases, the energy consumption also increases due to more and more *switching cost*. We notice that the

Table 4. Energy consumption comparison of the proposed TD-D with approaches only resizing VMs

Variance of workload fluctuation	Resizing VMs only, without break-even time [6]	Resizing VMs only, with break-even time	TD-D (proposed)
2	2253	2234	248
2.38	2314	2288	292
2.72	2328	2297	311
3	2328	2292	350

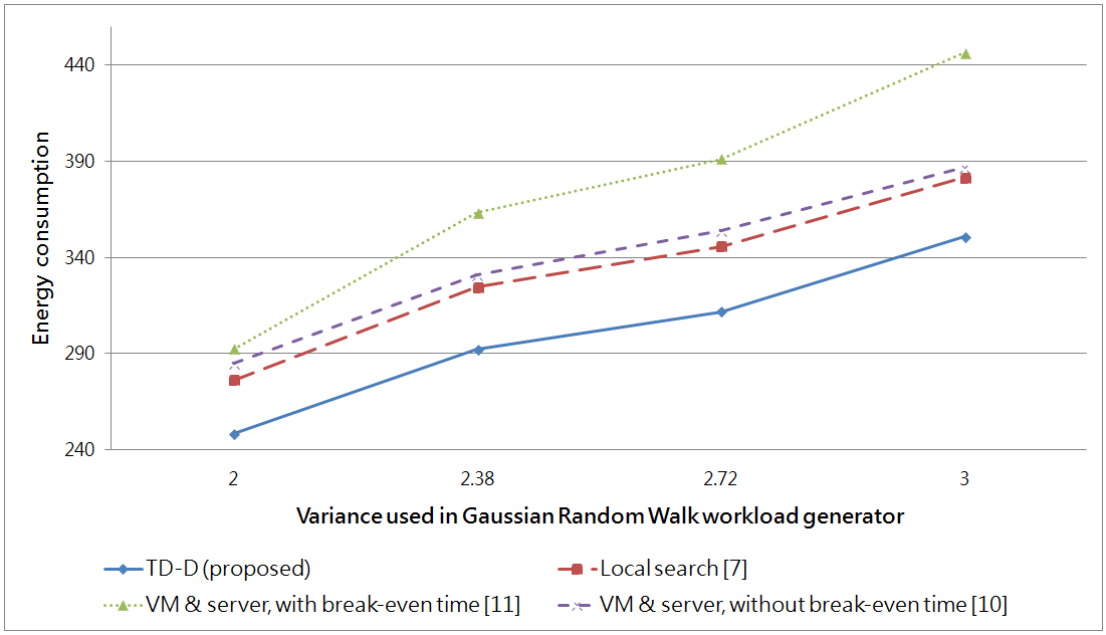


Figure 4. Energy consumption comparison of the proposed TD-D with approaches that resize both VMs and servers

approach concerning break-even time consume more energy than the one that uses on-demand resizing. This phenomenon can be understood as we described in Chapter 3, that when applying a simple break-even time rule, we may need to allocate more servers to accommodate the VMs that we kept in VM break-even time events. As the workload fluctuation become severe, more VM break-even time events happen and

Table 5. Comparison of average computing time and percentage of times an algorithm completed within 3 minutes

Algorithm	**Variance used in workload generator**			
	2	**2.38**	**2.72**	**3**
TD-D (proposed)	1.1 sec (100%)	8.05 sec (100%)	9.1 sec (98%)	34.78 sec (95%)
Local search [7]	180 sec (0%)	180 sec (0%)	180 sec (0%)	180 sec (0%)

more wasted servers are allocated. This is a good example why we need an optimal algorithm rather than a best-effort algorithm that using simple rules or heuristics.

In the second part, we measure the computing time of our algorithm. Since the complexity of our algorithm is mainly dominated by the level of workload fluctuation, we record the computing time under different workload fluctuation level. We also show the computing time of local search approach for comparison, which is the one that closest to *TD-D* in energy saving. The results are shown in Table 5. As we can see, the average computing time is acceptable for a long term resource allocation algorithm, and there is a high percentage of times that the algorithm can be completed within a time slot (3 minutes) and give us the optimal solution. In contrast, the local search approach never completes its search within three minutes due to its unbounded search space and the lack of stopping criterion. Since we use the relative energy unit, that is, we set the $P_{VM} = 1$ and the proposed *TD-D* can be completed within one time slot, we can conclude that the energy consumption of performing our algorithm is no more than 1. Compare the energy our algorithm can save with the energy and time our algorithm costs, we show the effectiveness and efficiency of our algorithm in energy saving.

Finally, we evaluate the reliability and effectiveness of our algorithm to

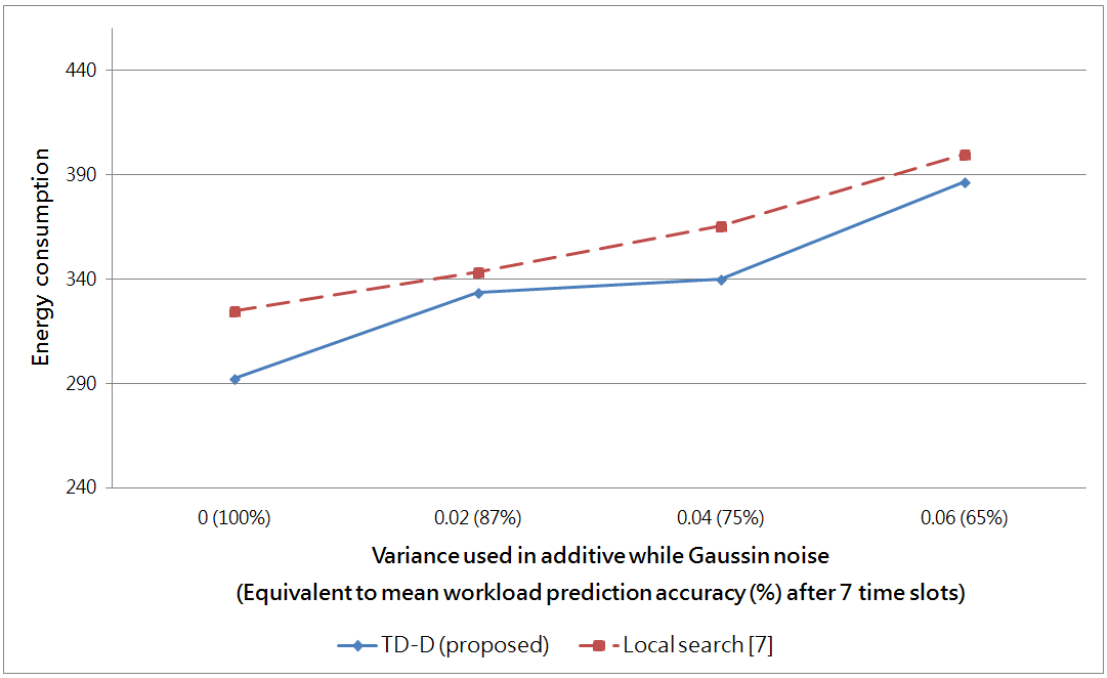


Figure 5. Comparison of energy consumption under different severity of prediction error

prediction error. Besides the reactive controller showed in Chapter 4, we re-perform our *TD-D* algorithm every 2 time slots, instead of *W*, to resist prediction error. When a resource over-allocation occurs, it brings extra energy consumption of operating cost. When a resource under-allocation occurs, it brings extra energy consumption of switching cost since the *reactive controller* has to switch on new VM/server to fulfill the demand. Again we use local search approach for comparison. The *reactive controller* and algorithm re-performing are also implemented in the local search approach. We set the variance used in workload generator = 2.38 and the results are shown in Fig. 5. As we can see, as the prediction error become severer, the energy consumption also become larger, due to the extra operating cost caused by over-allocation and the extra switching cost caused by under-allocation. We find our algorithm can still save more energy than the comparative approach under prediction error. Another way to evaluate the reliability to prediction error is under-allocation ratio. This is an important evaluation since some energy efficient algorithm may take

the risk of under-allocation to achieve less energy consumption. The results are shown in Table 6. The difference between the second and third column of Table 6 is, the second column show the average VM under-allocation counts of the whole data center, while the third column show the average VM under-allocation counts of each application, which is the actual influencing factor for user experience. We can find that our algorithm can keep in very low resource under-allocation ratio even under severe prediction error. The reason of how our algorithm can achieve low energy consumption while keeping in low resource-allocation ratio is the good side-effect of concerning break-even time. In a break-even time event, we may choose to keep that temporarily unnecessary resource, thus avoiding some resource under-allocation if a prediction error occurs and the resource demand does not really go down.

Table 6. VM under-allocation under different AWGN variances over 7 time slots

Variance of AWGN	**Average VM under-allocation (VM / time slot)**		**Average VM under-allocation per application (VM / (time slot × application))**	
	Local search	**TD-D**	**Local Search**	**TD-D**
0.02	0.27	0.30	0.01	0.01
0.04	0.86	0.76	0.03	0.03
0.06	1.20	1.13	0.04	0.04

Chapter 6

Conclusion

6.1 **Concluding Remarks**

In this paper, we introduce our minimum energy consumption resource allocation algorithm for cloud data centers called *Time-Directed Dijkstra* (*TD-D*). It can produce optimal solution by utilizing the existing load prediction approaches. We first characterize the difficulties of resizing at both VM and server levels, and then come up with an optimal algorithm that can seek the best trade-off between operating cost and switching cost to achieve minimum energy consumption. We demonstrate the correctness of our algorithm and show that even such high complexity problem can be completed by commodity machine in reasonable computing time. Compared with representative best-effort dynamic resource allocation algorithm, our optimal algorithm can save more energy under different workload fluctuation level. We also demonstrate the robustness and energy efficiency of our algorithm to prediction error.

6.2 **Future Work**

The next step is to use the workload traces from real world to further evaluate our algorithm. Another future work is the cluster version of our algorithm. Since there are more and more large scale data center, for reliability and scalability, the cluster version must be developed to build a decentralized resource control system.

References

[1] John J. Prevost, KranthiManoj Nagothu, Brian Kelley and Mo Jamshidi, “Prediction of Cloud Data Center Networks Loads Using Stochastic and Neural Models,” *Proc. of the 6th International Conference on System of Systems Engineering*, 2011, pp. 276-281.

[2] Truong Vinh Truong Duy, Yukinori Sato, Yasushi Inoguchi, “Performance Evaluation of a Green Scheduling Algorithm for Energy Savings in Cloud Computing,” *International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, 2010.

[3] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, Alfons Kemper, "Workload Analysis and Demand Prediction of Enterprise Data Center Applications," *IEEE 10th International Symposium on Workload Characterization*, 2007.

[4] Arijit Khan, Xifeng Yan, Shu Tao, Nikos Anerousis, "Workload Characterization and Prediction in the Cloud: A Multiple Time Series Approach," *IEEE Network Operations and Management Symposium (NOMS)*, 2012.

[5] Minghong Lin, Adam Wierman, Lachlan L. H. Andrew, and Eno Thereska, “Dynamic Right-Sizing for Power-Proportional Data Centers,” *IEEE INFOCOM*, 2011.

[6] Chunqiang Tang, Malgorzata Steinder, Michael Spreitzer, and Giovanni Pacifici, “A Scalable Application Placement Controller for Enterprise Data Centers,” *ACM Proceedings of the 16th international conference on World Wide Web*, 2007.

[7] Dara Kusic, Jeffrey O. Kephart, James E. Hanson, Nagarajan Kandasamy, Guofei Jiang, “Power and Performance Management of Virtualized Computing Environments via Lookahead Control,” *Cluster Computing*, vol. 12, no. 1, pp. 1-15, March 2009.

[8] Anton Beloglazov, Jemal Abawajy, Rajkumar Buyya, “Energy-aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing,” *Future Generation Computer Systems*, vol. 28, no. 5, May 2012, pp. 755–768, 2012.

[9] Danilo Ardagna, Barbara Panicucci, Marco Trubian, and Li Zhang, “Energy-Aware Autonomic Resource Allocation in Multitier Virtualized Environments,” *IEEE Transactions on Services Computing*, vol. 5, no. 1, 2012.

[10] Vinicius Petrucci, Orlando Loques, Daniel Mossé , “A Dynamic Optimization Model for Power and Performance Management of Virtualized Clusters,” *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, 2010, pp. 225-233.

[11] Anton Beloglazov, Rajkumar Buyya1, Young Choon Lee, and Albert Zomaya, “A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems,” *Advances in Computers*, vol. 82, 2011.

[12] Andrew Krioukov, Prashanth Mohan, Sara Alspaugh, Laura Keys, David Culler, Randy Katz, "NapSAC: Design and Implementation of a Power-Proportional Web Cluster," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 102-108, January 2011.

[13] Norman Bobroff, Andrzej Kochut, Kirk Beaty, “Dynamic Placement of Virtual Machines for Managing SLA Violations,” *10th IFIP/IEEE International Symposium on Integrated Network Management*, 2007.