2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing

Power-efficient Virtual Machine Placement and Migration in Data Centers

Shuo Fang[1], Renuga Kanagavelu[2], Bu-Sung Lee[1], Chuan Heng Foh[3], Khin Mi Mi Aung[2]
[1]School of Computer Engineering, Nanyang Technological University, Singapore
[2]Data Storage Institute, A*STAR, Singapore
[3]Centre for Communication Systems Research, Department of Electronic Engineering University of Surrey, United Kingdom

***Abstract*—In this paper, we propose a power-efficient solution for virtual machine placement and migration in a fat tree data center network. This solution reduces power consumption as well as job delay by aggregating virtual machines to a few hypervisors and migrating communicating parties to close locations. In this work, we consider OpenFlow as the implementation protocol. In an OpenFlow environment, a centralized controller oversees job loads, virtual machine requirements and hardware availability. Given observation of such global knowledge, the OpenFlow controller can schedule jobs and distribute virtual machines accordingly. As jobs change and flows shift, the OpenFlow controller dynamically adjusts virtual machine assignments by aggregating virtual machines to close locations in order to save energy. With this placement and migration proposal, more jobs can operate concurrently with close sources and destinations of flows, thus both job and flow delay can be reduced.**

***Index Terms*—Power Efficiency, Virtual Machine Migration, Delay Minimization**

I. Introduction

Virtualization has become a popular data center implementation technology. As it enables isolation of multiple instances under the same physical device or infrastructure, virtualization increases utilization efficiency thus reduces cost and simplifies operation. It also offers more flexible and secure environment. In a data center, virtualization concerns three tiers, namely server virtualization, network virtualization and storage virtualization. Among them, server virtualization is the most mature technology and it is the fundamental of the other two. Server virtualization is realized by implementation of virtual machines (VMs). Multiple VMs may run concurrently on a single physical machine (PM). The strategy for placing VMs among PMs plays an important role in optimizing data center performance. However, even the initial placement optimizes the network usage, as network status varies over time, its initial outcome may no longer be optimal. A well-designed VM migration algorithm needs to accommodate up-to-date network status, thus maintains an optimized distribution of VMs.

VMs are placed with two criteria, hardware resource usage and power consumptions. Hardware limits the number of VMs that can be allocated and their placement. Power consumption influences the operating cost of a data center and it attracts more and more research these years. By migrating VMs to a portion of the PMs, fewer PMs will be activated. Consequently, fewer number of routers and switches are needed. In [1] [2], studies have shown that equipment including servers, storages and network devices consume around 45% of the total power in data centers. Besides, they also point out that another 45% of the power are consumed for cooling down these equipment. Thus, server virtualization not only saves energy on powering equipment but also reduces cooling cost.

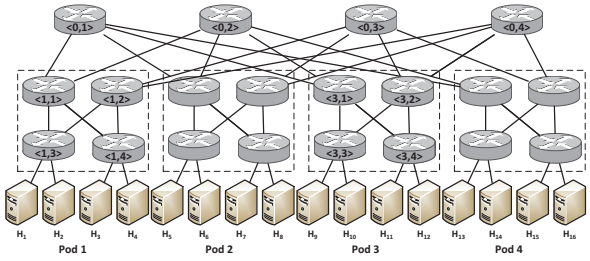

Fig. 1. Illustration of an $N = 2$ fat tree topology.

In a study subject to equipment power consumption [2], it has been shown that servers and network devices consume 40% and 23% of equipment power respectively. The server and network power is mainly consumed on devices such as servers and switches. From the view of a whole data center, minimizing the number of active servers and switches saves the major equipment power consumption as well as cooling cost. Therefore, power consumption can be significantly reduced.

However, various models of servers and switches consume power at different levels. A high-end server consumes from 2,000-15,000 Watts [3]. According to [4] and [5], a volume server chassis consisting of 14 blade servers consumes around 4,500 Watts under full workload. Similarly, switch power consumption also varies, as a high-end switch such as Cisco Catalyst 6500 series consume 400-4,000 Watts and an edge switch such as Dell PowerConnect series consumes around 100 Watts only.

A typical data center consists of three layers of switches, namely, edge switches, aggregate switches and core switches. Core switches usually use high-end switches which handles huge volume of traffic across the whole data center, while edges switches cost less on investment and consume less power. A source-destination VM pair that close to each other can communicate through edge switches instead of travelling through core switches. By migrating VMs of communicating parties to a close location, core switches usage can be reduced. Thus, data center can save energy with the same workloads.

In addition, aggregating source-destination VMs contributes more to power saving than simply aggregating traffic. Our previous study [6] found out that a switch consumes power at different levels according to its number of active links and their rates. In this work, the power consumption on Dell PowerConnect 8024F switch was studied. It also revealed that

978-0-7695-5046-6/13 $26.00 © 2013 IEEE
DOI 10.1109/GreenCom-iThings-CPSCom.2013.246

the base consumption takes the main portion of the total cost. To be specific, an idle PowerConnect 8024F switch consumes around 100 Watts, while it consumes around 130 Watts at the full load. Suppose an $N = 2$ fat tree network topology [7], source-destination VMs are at PMs of H_1 and H_9 separately in Fig. 1. By aggregating traffic, at least five active switches are needed, *i.e.*, source edge switch $< 1, 3 >$, source aggregation switch $< 1, 1 >$, core switch $< 0, 1 >$, destination aggregation switch $< 3, 1 >$ and destination edge switch $< 3, 3 >$. However, aggregating VMs to a closer location, for example, migrating the destination VM to H_2 requires only one edge switch $< 1, 3 >$ active. Thus, aggregating VMs may save more power than simply aggregating traffic.

In this paper, we aim to design a VM placement and migration solution that ensures power efficiency in data centers. Our proposal is based on fat tree topology, an attractive solution for modern data centers, and OpenFlow Protocol, a software defined networking technology for customize network protocols. This proposal can adapt to most of data center architectures with simple modifications. The rest of the paper is organized as follows. Section II introduces background information by describing fat tree topology and its construction rules, as well as OpenFlow protocols. Section III describes general mechanisms of this design, based on fat tree topology. Section IV analyzes the network model and formulates an optimization problem. After discussion on formulation, we develop heuristic algorithms from three approaches in Section V. We further test our solution with NS 3 simulator in Section VI and present results regarding simulation tests with different settings. Important conclusions are drawn in Section VII.

II. Background

In this section, we describe background information for our proposals. We will firstly introduce the fat tree topology and then the concept of Software-defined Networking (SDN) and OpenFlow project.

A. Fat Tree

Clos network topology [8] has been raised half a century ago for telephone switches, it aims to deliver high network throughput with commodity switches. Recently, in [7], Al-Fares *et al.* adopt a standard instance of a Clos topology called fat tree, which provides non-blocking network connection with 1:1 oversubscription ratio [7]. We based our work on this topology since it is widely acknowledged. However, with some adjustments our proposal can be applied to other topologies. Details about fat tree as a data center architecture can be found in [7]. However, for simplicity of reading, we summarize the important terms here. In such a data center of an $N-$ary fat tree as shown in Fig. 1, which is made up by $2N$-port switches, where N should be a positive number. A fat tree topology contains three layers of switches, namely edge layer, aggregation layer and core layer from bottom to top in the above figure. The lower two layers are separated into $2N$ pods, each containing two layers of N switches, with lower edge switches and upper layer aggregation switches. In the upmost layer, there are N^2 core switches, each core switch has a connection towards each of the $2N$ pods. Since each edge switch connects to N end stations and each pod has N edge switches, their are N^2 end stations in a pod and $2N^3$ end stations in an $N = 2$ data center fat tree topology.

B. Software-Defined Networking and OpenFlow

The concept of Software-Defined Networking (SDN) comes from Software Defined Radio, which modulates/demodulates using software instead of hardware as in Cognitive Radio. Similarly, different from traditional network in which data plane and control plane are implemented on hardware, SDN decouples intelligence that controls the flow to a distinct software application and handles packet traffic on hardware. This separation opens up the control flow to be easily managed and remotely accessed. Employing a central controller, which amasses knowledge of routing information on switches and network status, SDN manages routing algorithms separately while keeps data flow running on original network paths. Hence, SDN provides much easier modification to switch algorithms, as well as faster control over network status changes.

OpenFlow project, the most popular enabler for SDN, is proposed by McKeown et al. [9]. Installing a small piece of OpenFlow firmware gives engineers access to flow tables, rules for routing traffic. Moreover, this feature presents OpenFlow as an ideal tool for academic and research institutions to obtain network statistics and explore network performance as well as innovative algorithms, while protecting the proprietary routing instructions that differentiate one company's hardware from another. Many vendors such as Cisco, Hewlett-Packard have announced their intention to support OpenFlow, thus make it a step further to industry availability.

III. Power-efficient Virtual Machine Placement and Migration Scheme

The proposal aims to schedule VMs with power efficient concerns. It attempts to utilize fewer PMs and schedule VMs to migrate when necessary. In this section, we first present terms used in our design, and then illustrate the VM placement and migration solution.

A. Terms

A **job** describes the work load in a data center. Jobs usually come from Internet request such as Email service, VoIP service or file transfer. To complete a job, different types of VMs are required. A job may evoke one or more traffic flows among its VMs. Job information is usually described with three entries Job ID, Job Priority and VM List. *Job ID* denotes the job sequence number depending on job arrival time. *Job Priority* denotes job priority level, which corresponds to requirements on response time, delay and throughput. *VM list* lists number of VMs on each type that are required by this job. Each list entry is under the format of *VM Type ID: VM number*.

A **Virtual Machine (VM)** is a working unit which requires several system resources. Different types of VM demands various system resource. A type of VM is defined by parameters such as VM Type ID, CPU, RAM, storage and bandwidth. *CPU*, *RAM*, *storage* and *bandwidth* parameters denote respectively resources required by the type of VM that is specified in the *VM Type ID* field.

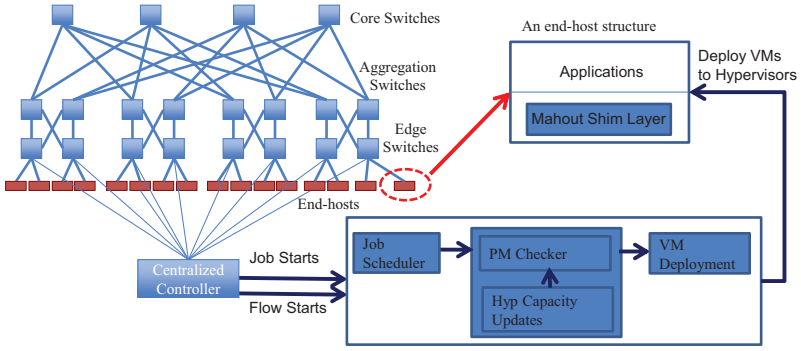

Fig. 2. Power-efficient VM placement and migration illustration.

A **hypervisor** is a piece of computer software, firmware or hardware that creates and runs virtual machines. A hypervisor is running on a PM and creates VMs on it. Hypervisors track PM resources in a structure including Hypervisor ID, CPU, RAM, storage and bandwidth. A hypervisor is identified by its *Hypervisor ID*. *CPU*, *Memory*, *Storage* and *Bandwidth* fields denote the total CPU, RAM, storage and bandwidth units of its hosting PM. The last field *Local Capped Utilization* specifies a percentage as the threshold to indicate approaching of full utilization. In each job, **flows** are generated during its execution. As defined by OpenFlow protocol, each job only maintains at most one flow at the same time. That means, communications among different VM pairs are sequentially conducted. In our VM migration mechanism, migration can only be scheduled when a flow starts.

B. Job Requirements and Process

In a data center, jobs are processed upon arrival as shown in Fig. 2. To process a job, the OpenFlow controller deploys all the VMs required. To deploy a VM on a PM, current resources on PM must be sufficient to meet VM requirement. A job can only work when all its VMs are deployed. If any one of the VM cannot be deployed, this job will suspend for a random time before checking to execute again. In such a case, some jobs may be executed later than the subsequent jobs. This happens when a job cannot be scheduled and a later job require less resources, or when a job reschedules to a later time and another job released just before a new job comes. In our design, OpenFlow controller also periodically checks job status, PM utilization, and network traffic volume. With these information collected, the controller schedules VM Migration when necessary. Therefore, a job is processed through two stages, namely, VM placement and VM migration. The VM placement stage happens when a job is launched. In this stage, VMs are initially scheduled and deployed. The VM migration stage happens during job execution when flows are initiated, VMs might be rescheduled to form more inner pod flows thus less core switches are needed.

IV. Problem Formulation

In this section, we present our problem of power-efficient VM placement and migration with the goal of reducing the network energy and the end-to-end delay in a fat tree data center network.

A. Problem Statement

Our problem statement can be briefly described as follows: we model a data center network using a directed graph $G = (V, E)$, in which each node $v \in V$ represents a network switch, PM or an external client. Each directed edge $(u, v) \in E$ represents a physical link between the switches and between a switches and a host. Each link (u, v) has a nonnegative capacity $c(u, v) \geq 0$. We implicitly assume that $c(u, v) = 0$ for $(u, v) \notin E$.

We consider a set of t hosts are available and their resource capacities given along CPU, memory, storage and Network bandwidth dimensions. There are m VMs to be placed. The requirements of these VMs are given along the dimensions of CPU, memory, storage and network bandwidth. We have to find a mapping between VMs and PMs that satisfies the VMs' resource requirements while minimizing the number of PMs used, minimizing data center network energy, and minimizing the end-to-end delay.

Input: let there be t PMs $\mathcal{S} = \{S_1, S_2...S_t\}$ and each host $S_i = \{c_i, r_i, s_i, b_i, u_i\}$ with different resource capacity attributes: c_i for CPU, r_i for RAM, s_i for storage, and b_i for bandwidth. Each server has a Capped Resource Utilization value, u_i (default at 95%) beyond which no more resources are allocated.

We consider a set of m different VM types depend on their resource needed (VMs) $\mathcal{V} = \{V_1, V_2...V_m\}$. Each VM type $V_i = \{c_{vi}, r_{vi}, s_{vi}, b_{vi}\}$ with different resource usage attributes: c_{vi} for CPU, r_{vi} for RAM, s_{vi} for storage, and b_{vi} for bandwidth.

We are given the set of n jobs from tenants need to be assigned to a data center, each job J_i with a list of VM types and number of requested VMs for that type.

$\mathcal{J} = \{J_1, J_2...J_n\}$ where $J_i = \{x_1, x_2...x_m\}$ and $\forall k$, x_k is the number of VM type V_k needed by Job J_i.

The network traffic source or destination is one of the m VMs. Here, over a link $(u, v) \in E$, we reserve the bandwidth $b_i(u, v)$ for the demand n. $b(u, v) = \sum_{i=1}^{i=n} b_i(u, v)$ is then the aggregated bandwidth over the link (u, v). For each job j, we need to find m_k feasible PMs to support its physical resource requirements, as well as a set of path in G to route

its traffic among VMs. Each path consists of a set of links that inter-connect switches and PMs.

Furthermore we have defined $Host_i = \{host_{i1}, ..., host_{im}\}$ for each server S_i that denotes the set of VMs assigned to that server, where $host_{im} = 1$ if the node S_i is hosting the VM m and $host_{im} = 0$ otherwise. The power consumed by server S_i depends on its physical component as well as on the set of VMs present:

$$P(S_i) = P(S_i, Host_i) \tag{1}$$

The overall Energy cost for moving the VMs from one host S_i to another host S_j is calculated by summing the cost of every movement M as

$$P_{mig} = \sum_{l \in L} Migrationcost(S_i, S_j, V_l) \cdot M_l \tag{2}$$

As hosts are interconnected by networks, the P_{delay} network delay/cost is required to transmit the VM from previously allocated host to the new host.

The linear function for power-efficient VM placement and Migration can be written as,

$$\min P(S_i) + P_{mig} + P_{delay} \tag{3}$$

B. Constraints

We define the constraints that need to be satisfied during VM placement with traffic routing on multiple paths and write them in the form of inequalities.

- Capacity constraints: the total resource requirement of the VMs placed on a host should not exceed its capacity. For each resource capacities of a given host j, the sum of the resource requirements of all VMs placed on it should be less than or equal to the total available capacity of the host.
  $\sum_i r_{ik} x_{ij} \leq 1, \forall k \forall j$ where r_{ik} is the fractional resource requirement of an individual VM i along the dimension k. Currently, k is equal to 4 since we consider dimensions of CPU, RAM, storage and network bandwidth.
- Placement constraint: all virtual machines should be placed.
  $\sum_i x_{ij} = 1, \forall i$ where each x_{ij} indicates whether VM i is placed on the host j.
- Bandwidth constraint: the aggregated bandwidth $\sum_{i=1}^{i=n} b_i(u, v)$ over link (u, v) should not exceed its capacity $c(u, v)$.
  $\sum_{i=1}^{i=n} b_i(u, v) \leq c(u, v)$ for each $u, v \in V$.
- No loss constraint: for every switch u in the Data Center that is not the source switch or the destination switch of the job i, the data flowing into it must be equal to the data flowing out of it (i.e., no packet loss).
  $\sum_{v \in V} b_i(u, v) - \sum_{v \in V} b_i(v, u)$ for each $i = 1, 2 \ldots n$.
- Bandwidth demand constraint: the data flowing out of the source switch s_i across multiple paths of the job i must be equal to the job's bandwidth demand b_i.
  $\sum_{v \in V} b_i(s_i, v) - \sum_{v \in V} b_i(v, s_i) = d_i$ for each $i = 1, 2 \ldots n$.

The problem formulation even when offline is NP-hard. In next section, we present heuristic algorithms to tackle VM placement and Migration.

V. Heuristic Algorithms

In this section, we introduce three heuristic algorithms for VM placement and VM migration, namely First Fit (FF), Best Fit (BF) and Worst Fit (WF). We will elaborate in details in the following.

A. Virtual Machine Placement

This stage is invoked when a job is launched. In this stage, two mechanisms are designed and discussed, namely Random Placement Mechanism and Power-efficient Placement Mechanism.

To deploy a VM, Random Placement Mechanism, randomly selects a hypervisor; if the selected hypervisor cannot satisfy the VM requirements, the mechanism continuously explore other hypervisors for t times. The parameter of t is called *times to try*. To simplify the denotation, we name this Random Placement as *Random*.

Similarly, Power-efficient Placement Mechanism also aims to find out a proper hypervisor to deploy a VM. But it firstly checks active hypervisors and consults if the VM can be assigned to any of them. A new PM only starts when all the active hypervisors are unable to accommodate the VM. We name this mechanism as *Power-efficient*.

Since this mechanism first targets on active PMs, it not only aggregate traffic to a smaller number of active PMs, but also assures the active PMs are close enough to fully utilization before activating a new PM. Thus the overall system utilization rate is also higher than Random mechanism. In Power-efficient mechanism, three algorithms are discussed.

The differences among the three algorithms lies in the sequence when finding a proper PM. In FF, the controller deploys the VM directly to the first suitable PM, while the controller calculates a weight difference w_i in the other two algorithms. It deploys the VM in the PM with lowest weight in BF, in comparison, it deploys the VM in the PM with highest weight in WF. We calculation the weight difference can be varied, here we use

$$w_i = \Delta c_i / c_i + \Delta r_i / r_i + \Delta s_i / s_i + \Delta b_i / b_i, \tag{4}$$

where Δc_i, Δr_i, Δs_i, and Δb_i denote the unutilized resource of CPU, RAM, storage and bandwidth.

When a job is completed, it signals the OpenFlow controller to schedule a VM release event. After VM release, active hypervisors check the VMs assigned to its hosting PM. In case of none VMs assigned, the hypervisor instructs the PM to switch into the inactive mode.

B. Virtual Machine Migration

VM migration stage is invoked when a flow starts. In this stage, two communicating VMs might be moved into the same pod. Since each pod contains N^2 hosts, we use the PM index to check whether they are in the same pod. The mechanism first checks if the source and destination VM are in the same pod. Otherwise, they attempts to migrate either source VM or destination VM to an active PM in the same pod of the other. If neither source pod nor destination pod can accommodate the other VM, the algorithm tries to allocate either VM to an inactive PM. Depending which algorithm is implemented, the migration mechanism varies.

The three algorithms differentiate themselves when if an active PM can host VM. In FF, the controller checks sequentially and migrates the VM to the first one with sufficient resource. In the contrary, the BF and WF algorithms calculate the weight difference first, and migrate to the PM with least or most difference respectively.

When a migration is scheduled, the mechanism employs pre-copy memory migration [10]. In pre-copy migration, the hypervisor firstly copies all the memory from the migration source to the migration destination, while the VM is still operating on the migration source. The VM will be resumed on the migration destination when the copy at the destination is sufficient for VM execution.

The VM placement and migration solution aims to reduce number of active devices to save power consumption in a high performance data center network. It locates VMs closely on fewer number of PMs.

VI. Simulation Experiments

In this section, we evaluate the proposal through two sets of experiments. In the first experiment, the simulation runs under an $N = 4$ fat tree topology where hosts 128 PMs. In the second experiment, the simulation runs under an $N = 8$ fat tree topology where 1024 PMs are hosted. In each experiment, we evaluate them from two perspectives, namely *active PM number* and *inner pod flow percentage*. Active PM number indicates the server power consumption level in a data center. Inner pod flow percentages shows the percentage of flows that travels within a pod instead of traveling across different pods, which are called inter pod flows. Thus, a higher inner pod flow percentage indicates less core switch power consumption.

Experiment parameters are shown in Table I. In the first experiment, we randomly generate 20 jobs. Each job requires at most 10 VMs and each VM may belong to a preset 15 VM types. Moreover, each job contains at most 10 flows. In the second experiment, most parameters are the same with the previous one with the exception that only job numbers are increased. Similarly, in this experiment, 400 jobs are generated with at most 10 VMs from the preset 20 VM types. Each job contains at most 10 flows. The simulation time is capped at 800 seconds.

TABLE I
Parameter Setting in Experiment I and Experiment II

Parameters	Experiment I	Experiment II
Job Number	20	400
Duration	1-20	1-20
VM Number per Job	1-100	1-100
Flow per Job	1-10	1-10
Times to Try t	10	10

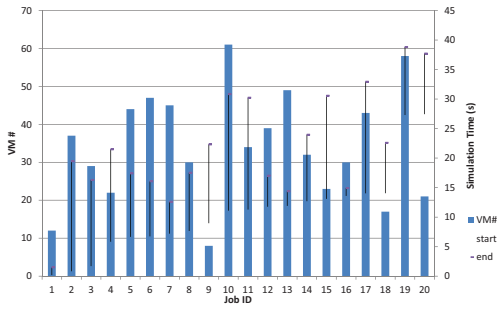

Fig. 3. Job start/end time and VM number illustration

Firstly, we show an illustration of job start time, end time and VM number in Fig. 3, so that it will be a clearer view of the job distribution. Secondly, active PM number of both $N = 4$ and $N = 8$ fat tree are shown in Fig. 4 and Fig. 5. In this report of active PM number, we focus on the performance comparison of Random mechanism and Power-efficient mechanism. For a center value of N, we run the same configuration for both mechanisms. Thus we do not add migration feature in this report. From these two figures, we can see that the active PM numbers on the Power-efficient mechanism are always fewer than the Random mechanism. At 5-15s and 20-30s, a significant difference can be observed between Random mechanism and Power-efficient mechanism.

For Power-efficient mechanism, there are no evident difference between the two schemes. Since there are five resources that limit VM assignment, the current weight difference function w may not sufficient enough to guarantee a fully utilization. As for BF algorithm, the 'hole' (we refer 'hole's as unutilized system resources) may not be small enough. As for WF algorithm, the 'hole' may not also big enough for another VM. Moreover, we can see that for FF, it performs better with less active PMs used when the jobs end. This tendency might come from the fact that it tends to allocate VMs used in a job the same set of PMs. Therefore, when a job end, a PM can be deactivated. In comparison, in BF and WF algorithms, VMs belong to the same job might be allocate to different PMs. When a job ends, the PMs which hosts VMs from other jobs

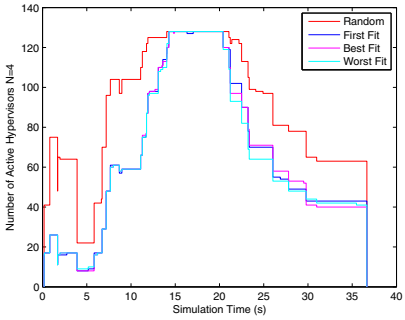

Fig. 4. Active PM number for N=4 fat tree.

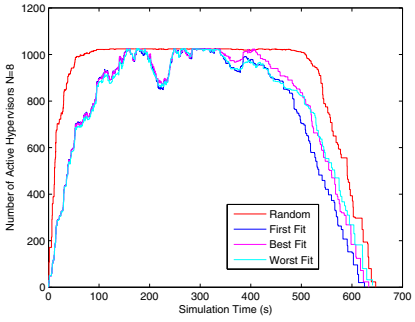

Fig. 5. Active PM number for N=8 fat tree.

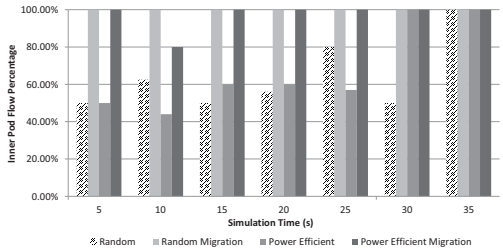

Fig. 6. Inner pod flow percentage when N=4.

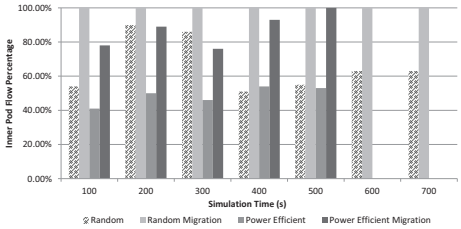

Fig. 7. Inner pod flow percentage when N=8.

cannot be deactivated. As a result, we just compare FF with Random mechanism in terms of inner pod flow percentage in the later illustration.

The drop in Fig. 5 at around 300s is due to the reschedule of an amount of later jobs when system capacity is full. As in the Power-efficient mechanism, the centralized controller first attempts to fill in all the active PMs. VMs with fewer resource requirements can schedule to active PMs before start another PM. Therefore, this mechanism firstly fills the 'hole's in active PMs. Comparatively, the Random mechanism does not give more priority for filling in the 'hole's than starting an inactive PM. Even when there are smaller VMs that can be assigned to an active PM, the VMs may be allocated to some inactive PMs. When a VM requires more system resources it is more probable that none of the PMs have enough resources to deploy it. Thus, there are bigger 'hole's in PMs and less VMs can be hosted in the Random mechanism. As the Random mechanism reaches full utilization earlier, it delays more jobs to a later time. When previous jobs complete, there are bigger gaps and delays before job retransmission. Therefore, it reacts slower when system resource release and results in longer delays.

Thirdly, we consider the job delay for both N=4 and N=8 experiments. Table. II shows the total time of job delay in both experiments. We can see from the results for the first example all of the algorithms delays for the same time. They have the same delay time because there is only one job delay due to fully utilization. However, for N=8, the difference between the algorithms becomes obvious, in which FF have the shortest overall delay of 55.06s while Random has the longest delay of 76.04. That means, it is much easier for FF to find sufficient resources for processing job and it consumes around 26% lesser of time. Fourthly, we show the inner pod flow percentage in Fig. 6 and Fig. 7. The two figures show the percentage of inner pod flow number to the overall flow number for both Random and Power-efficient mechanisms with or without VM migration. In VM migration mechanism, flows are categorized to inner pod flow and inter pod flow. An inner pod flow transits within a pod and does not go through core switches, while an inter pod flow transit across two differen pods and must go through a core switches to reach the destination. An inter pod flow which requires a core switch tends to consumes more power.

TABLE II
TOTAL TIME OF JOB DELAY IN EXPERIMENT I AND EXPERIMENT II

Parameters	Experiment I	Experiment II
Random	6.61s	76.04s
FF	6.61s	55.06s
BF	6.61s	63.25s
WF	6.61s	64.85s

In each figure, we present results from four schemes, namely, Random, Power-efficient, Random Migration and Power-efficient Migration. We show the inner pod flow percentage from 5 to 35 seconds with an interval of 5s in Fig. 6, and from 100 to 700 seconds with an interval of 100s in Fig. 7. We can see from the figures that with Migration algorithm, more than half of the flows can be migrated as inner pod flows. Thus, less core switches are needed in these schemes. The lack of data in Fig.7 on 600s and 700s dues to job completion in Power-efficient schemes.

VII. CONCLUSIONS

In this paper, we have proposed a power-efficient solution for virtual machine placement and migration. Our proposal saved energy by aggregating virtual machines to a few physical machines to reduce server power consumption. As network evolves, communicating virtual machines would be migrated into close locations to save link and switches power consumption. The simulation results have shown that this solution utilizes less number of physical machines and completes job faster. It achieves up to half number of activating physical machines and around 26% reduction in job delay as well as up to 50% flow migrating to inner pod flows. Considering that less equipment consumes less energy, server and core switches can consume less power in our proposal.

REFERENCES

[1] T. Babasaki, T. Tanaka, Y. Nozaki, T. Aoki, and F. Kurokawa, "Developing of higher voltage direct-current power-feeding prototype system," in *Telecommunications Energy Conference, 2009. INTELEC 2009. 31st International*. IEEE, 2009, pp. 1–5.
[2] J. Raji, "Evolving towards the GREEN Data Center," 2009.
[3] J. Koomey, "Estimating total power consumption by servers in the us and the world," 2007.
[4] R. B. John Beckett and the Dell Server Performance Analysis Team, "Power Efficiency Comparison of Enterprise-Class Blade Servers and Enclosures," 2007.
[5] "Blade Server Power Study," 2007.
[6] S. Fang, H. Li, C. Foh, W. Y., and K. Aung, "Energy Optimizations for Data Center Network: Formulation and its Solution," in *2012 IEEE Globecom Global Communication Conference, organization=IEEE*.
[7] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.
[8] C. Clos, "A study of non-blocking switching networks," *Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424, 1953.
[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.
[10] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 273–286.