IMPROVED SERVER CONSOLIDATION ALGORITHMS IN DATA CENTERS

Project

Presented

to the Faculty of

California State University Dominguez Hills

_____

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

_____

by

Shadi Shiri

Fall 2016

IMPROVED SERVER CONSOLIDATION ALGORITHMS IN DATA CENTERS

AUTHOR: SHADI SHIRI

APPROVED:

Bin Tang, PhD
Faculty Adviser

Mohsen Beheshti, PhD
Committee Member

Jianchao “Jack” Han, PhD
Committee Member

ACKNOWLEDGMENTS

First, I would like to thank my project adviser, Dr. Bin Tang, for his helpful feedback and ideas. His comments and suggestions have really helped me with my project.

I would also like to sincerely thank Dr. Mohsen Beheshti, professor and department chair of computer science. I must express my very profound gratitude for his wonderful support and encouragement.

Also, I would like to express my thanks to my committee member, Dr. Jianchao "Jack" Han, professor of computer science, for all his guidance throughout my years of study.

I would like to thank my family for supporting me throughout my studying years.

Finally, and the most importantly, I would like to thank my husband for his unfailing support, understanding, and patience during these past years. I thank him for his compatibility and compromise to provide me a proper environment for studying.

TABLE OF CONTENTS

PAGE

LIST OF TABLES

PAGE

LIST OF CHARTS

PAGE

LIST OF FIGURES

PAGE

ABSTRACT

This project starts with an introduction to data center and its power consumption and then moves to explaining virtual machine replication and providing a detailed description of the three most famous replication algorithms, which are minimum-cost flow, first-fit, and greedy.

The main part of this project is about server consolidation. In server consolidation, we try to create more inactive physical machines from the left active physical machines after virtual machine replication and turn them off to save energy and have a more efficient data center. I explain an existing consolidation algorithm and its drawback. And then I propose two consolidation packages which improve the existing outputs and talk about their features.

In the next part, I run proposed consolidation algorithms on many different data centers with different specifications and compare the final number of turned-off PMs to find the highest number.

In the last part of the project, I talk about the final cost of different data centers with different virtual machine replication algorithms and server consolidation to find the most efficient virtual machine replication [20] and consolidation algorithm.

CHAPTER 1

INTRODUCTION AND BACKGROUND

Through the boom of the microcomputer industry, which started around the 1980s, users began to use computer everywhere without enough consideration of its operating requirements, although by expanding the complexity of the information technology operations, organizations started to think about the need of controlling information technology supplies. Around the 1970s, the development of OS UNIX resulted in the increasing availability of Linux, which was adjustable to the Windows operating system PC through the 1990s. This technology is named server with a Linux operating system, which is a time-sharing operating system based on a client server model to share resources among multiple users. A data center is an equipment that centralizes appliances, tools, and IT operations; and an organization's computer systems and related components can be hosted by using a data center. This component includes storage systems and telecommunications. It should have a backup for communication connections and power supplies. In addition, it should include environmental controls such as fire suppression and air-conditioning. Various security devices and tools are one of the most necessary parts of a data center.

Large data centers are operating in the scale of an industrial environment, and the electricity they use is very close to the usage of a small town. In most companies, a data center is a place where the most critical processes are running, and it is the brain of a company.

There are thousands of server machines in data centers, and a large number of Internet services such as search engines [20], social networks, and video streaming are supported by them. Recently, to use resources and operations more effectively and reduce costs, data centers use server virtualization technologies.

In computer science, virtualization is making a virtual version of anything, such as a computer network, storage device, computer hardware, and operating system. It started in the 1960s. It is a style of dividing system resources prepared by a mainframe computer and used between different applications. Virtual machine replication (VM replication) is a method of protecting a VM in addition to expanding the availability of a data center [18]. This can be done by taking a VM and copying it into another VM. In this topic, there is another concept that is named server consolidation. Moving VM copies into a smaller number of PMs while still meeting constraints and preserving the initial cost of the VM replication to reduce power consumption is named server consolidation.

Fig.1. Data Center [24]

CHAPTER 2

VIRTUALIZATION AND DATA CENTER POWER CONSUMPTION

What Is Virtualization?

Virtualized [15] data centers are being used more and more because of the fast growth of cloud [17] service requests. This results in the establishment of large-scale virtualized data centers. In data centers, virtual machines are used to handle the service requests of the user. One problem is failure of a VM. If one needed VM fails, a user's request cannot be completed. To reduce the impact of a failure, replication mechanisms can be a very good solution.

On the other hand, high operating costs are one of the parameters of large data centers because they use a very large amount of energy. The infrastructure of a data center is the place for processing user requests, and as a result, VM replication [8] is an important factor during the time needed for job completion, and it means time performance. In addition, it can affect energy consumption.

Recently, virtual machine replication [19] and placing them in data centers has been the center of attention in the research community. One of the most common parameters in data centers is failure. Human errors and rack failures because of hardware, server, link, switch, software, and power outage problems can be a cause of failure. Individual server and switch failures [12], [16] can become the norm rather than the exception in data centers by growing the size of data centers. One solution to control fault tolerance is to have redundancy in the hardware and software. User requests to the virtual

machines can be distributed in different physical machines by replicating virtual machines (VMs) and placing their replication copies in data center networks, and this can reduce server load. In addition, all fault tolerance can be achieved by having redundant copies of a VM on different servers. On the other hand, the cost of implementing DR (disaster recovery) can be reduced, and it can prepare increased flexibility and ensure the protection of recovery time objectives (the time needed to restore a service after a disaster or disruption to prevent consequences related to a break is named recovery time objective [RTO]).

Virtual machine replication [14], [19] is important in the smooth operation of data centers. Because physical devices and platforms are the factors of a data center for functioning virtual machines, a small problem in the physical server can become a big problem in virtualization-based cloud computing data centers.

Business continuity and disaster recovery are the main purposes of a virtual machine replication technology design. You should make sure your data in disaster situations are preserved. In virtual machine replication, a very simple level is one type of VM protection, and it involves making a copy of the VM when there is no problem and putting it in another VM for when the time disaster happens.

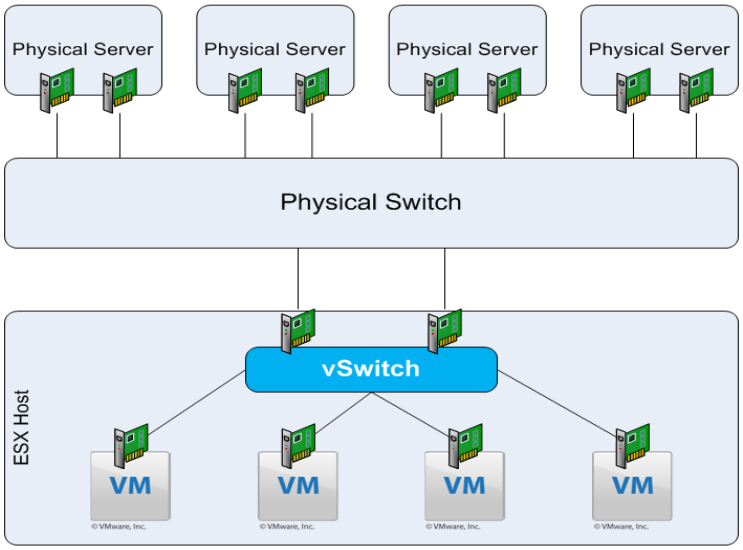


Fig.2. Virtual Machine Replication [25]

There are different famous virtualization technologies, such as Microsoft Virtual Servers [2], VMware [3], and Xen [9]. By enabling and installing different OS environments on the same physical server, it is possible to incorporate applications running on multiple physical (PM) servers into a single physical server. One of the desirable results is being able to turn off some servers and reducing power consumption in a large data center, which is one of the concerns of people who are using large data centers. Virtualization provides the environment for dividing the hardware sources of a PM such as CPU cycles, memory, and bandwidth into several smaller separated computing units, which are named virtual machines (VMs). They can be rented to different tenants, and the customer has to pay in a pay-as-you-go manner. One of the

samples of a web service for preparing a computed capacity in a cloud that is resizable is Amazon Elastic Compute Cloud (Amazon EC2).

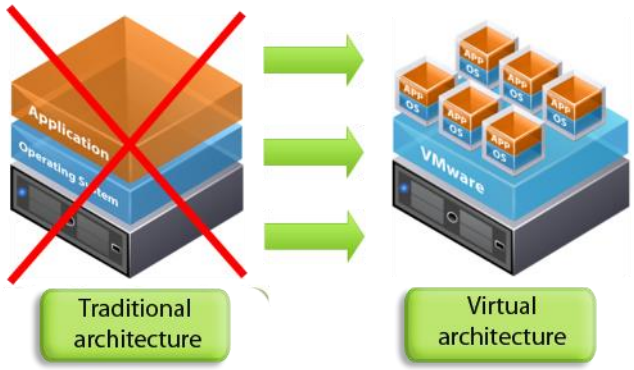


Fig.3. Virtual Architecture [26]

Any failures in the system components can result in an interrupted/preempted job. Execution of an interrupted job affects not only the results but also the energy consumption and increases job completion time performance. As a result, power consumption is a big concern in any data center.

Data Center Power Consumption

The truth about data centers is that they are growing unexpectedly regardless how correctly and efficiently we run them. This results in the increase of the amounts of power consumption. In fact, efficiency improvements contribute to the rapid growth of data centers. Studies have proved that equipment, such as servers, storages, and network devices, and cooling are the two largest parts of power consumption in any data center. Each of them uses around 75% of the total power consumption in a data center. One-third

or half of the power consumption costs of the servers and storages are due to switches, routers, and various links that are different network devices in data centers.

Another research [4] proves that network devices use almost 50% of the total power in a data center [11] if the system is not used as expected, and the servers are fully energy proportional, which means that servers are consuming nearly no power when idle and gradually consume more power as the activity level increases.

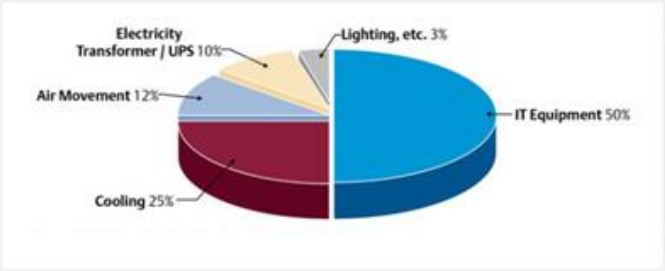


Fig.4. Data Center Power Consumption [27]

Data Center Topology

There are different famous data center topologies [7]. In this project, we focus on the fat-tree network, which is extensively used in data centers to interconnect different parts of the environment, such as commodity Ethernet switches. It is a regular network for unchangeable practical communication, which is a type of the three-stage Clos network [10]. A Clos network is a type of a multistage network that has circuit switching, which is rearrangeably nonblocking with an oversubscription ratio.

The Architecture of Fat-Tree Topology

A k-array fat tree is shown in Fig. 5. In a fat tree, k is the number of ports of each switch, and in this sample k = 4. A fat tree has three layers of switches: (1) edge switch, (2) aggregation switch, and (3) core switches from bottom to top. Core switches consume a lot of energy power because they are used for handling a huge amount of traffic in the whole data center. On the other hand, less amount of traffic is handled by aggregate and edge switches, and as a result, they use less amount of energy.

There are k pods in aggregate and edge switches, which are the lower two layers. In Fig. 5, a fat tree has three layers, and each layer has k/2=2 aggregation switches and k/2=2 edge switches. They form a complete bipartite graph in between. In the architecture of a fat tree, each edge switch is connected to both a physical machine and aggregation switches, it is connected to k/2=2 physical machines, and the other k/2=2 ports are connected to each of the k/2=2 aggregation switches in the same pod. There are $(k/2)^2$ k-port core switches; each of them is connected to each of the k pods. In general, a fat tree that has k-port switches supports $k^3/4$ physical machines. In the small data center in Fig. 5, there are 16 physical machines.

All bandwidth is available to the end hosts, and it can always be saturated for any request patterns. The worst scenario is the ratio of the accessible bandwidth of the aggregate bandwidth among the end hosts to the total bisection bandwidth of a specific communication topology.

An oversubscription can happen in a situation wherein all hosts may potentially communicate with any other hosts in the full bandwidth usage of their network interface.

In fact, three pods are different units of network, computer, and storage, which will be designed together as a unit in a data center. In general, the total number of physical machines (PM) that can be supported by a fat tree with k-port switches is $k^3/4$.

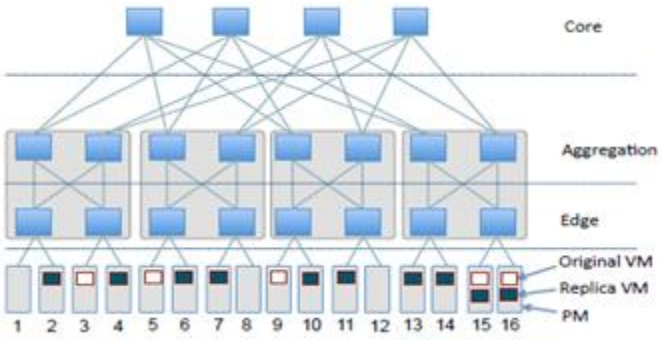


Fig.5. Fat-Tree Topology Architecture. A k-ary fat-tree topology with k = 4 and 16 physical machines (PMs). There are p = 5 original virtual machines (VM): (v1, v2, ..., v5) from left to right, located at PM 3, 5, 9, 15, 16, respectively [23].

CHAPTER 3

VIRTUAL MACHINE REPLICATION AND THE PROPOSED SERVER CONSOLIDATION ALGORITHM

Virtual Machine Replication Algorithms

There are many VM replication algorithms. In this project, we focus on (1) minimum-cost flow [5], [21] algorithm, (2) first-fit algorithm, and (3) greedy algorithm. These are the most famous algorithms in virtual machine replication methods.

Replication Constraint of VMs

There is one basic rule that should be protected during the replication all VMs through the data center. Assume that there are R copies of each VM that should be copied and placed in different physical machines in the data center network. It is not possible to copy more than one of the same VM in the same PM. In this way, it is possible to provide fault tolerance for the whole data center. This rule has two results: (1) The number of replica copies of each VM cannot be more than the total number of all physical machines. (2) Each PM (including the source PM) is able to store p separate VMs at max, although the storage capacity of a PM can be larger than the total size of the p virtual machines. As a result, we need to define an effective storage capacity.

Effective Storage Capacity of a PM

The effective storage capacity of PM i, denoted as mei, is the maximum storage capacity of a PM(i) that can be used to store virtual machines in VM replication. As a result, it is not possible to exceed the capacity of the PM by copying too much VM on that PM.

Minimum-Cost Flow Algorithm

The cheapest possible way of sending a certain amount of flow through a network can be found by using a minimum-cost flow problem (MCFP) [22]. It is an optimization and decision problem. One of the best uses of the minimum-cost flow [1], [13] algorithm is finding the best route for sending delivery from a factory to a warehouse. In this problem, each road has a specific capacity and a special cost.

Since most other types of problems can be mapped to a minimum-cost flow problem, it can be resolved very efficiently by using a network simplex algorithm. Among all the flow and circulation problems, the minimum-cost flow problem is one of the most substantial.

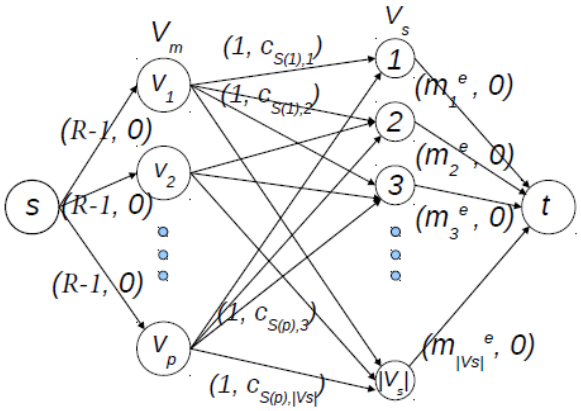


Fig.6 VM Replication and Transformation. The VM replication and transferring problem is equivalent to a minimum-cost flow problem. In each parenthesis, the first value is the capacity of the edge and the second is the cost of the edge. Note that it is not a complete graph between VM and Vp, with the following edges missing: (VM1, S(vm1)), (VM2, S(vm2)), ..., (VMp, S(vmp)) [23].

Transformation

In the first step, the data center network in Fig. 5, G (V; E), should be transferred to a flow network G0 (V 0; E0). The new graph has the following specifications [23]:

1. $V^{/}$ = {s} ∪ {t} ∪ VM ∪ Vp, where s is the new source node, t is the new sink node, and VM = $\{vm_1, vm_2, ..., vm_p\}$is a set of p new nodes. Like before, vm_i represents virtual machine i.

2. $E^{/}$ = {(s, i) | i ∈ VM} ∪ {(j, t) | j ∈ Vp} ∪ {(i, j) | i ∈Vp, j ∈ VM} − {(vm_1, S(vm_1)), (vm_2, S(vm_2)),...,(vm_p, S(vm_p))}. Here, an edge does not exist between node vm_i and S(vm_i), the source node (PM) of vm_i. This is because the original copy of each VM does not need to be transferred, and only K – 1 copies are transferred for each VM.

3. For each edge (s, i), set its capacity as K − 1 and its cost 0. For each edge (j, t), set its capacity as $m_i^{/}$ and its cost 0.

4. For all other edges (i, j), i ∈ VM, j ∈ Vp, we set its capacity as 1 and its cost as Cij, the minimum energy cost sending k-Byte information from physical machine i to physical machine j. This minimum energy cost can be calculated using all pairs minimum cost paths (Floyd algorithm). Together with 2, it guarantees that the K − 1 copies of each VM are migrated to K − 1 different physical machines other than the source physical machine.

5. For simplicity, we consider the transferring cost between a PM and an edge switch as 1. The transferring cost between an edge switch and an aggregation switch is 5. The transferring cost between an aggregation switch and a core switch is 10.

First-Fit Algorithm

In the first-fit algorithm, all VMs are being copied in the first available place that meets the condition of the VM and PMs. Assume that all the existing PMs are well organized from left to right in the fat-tree data center topology. Until all the VMs have their original replica copies located in the data center, it starts to duplicate each of the original VM and put their R-1 replica copies on the first accessible PM, the second

available PM, and so on. Remember that for copying each VM on one specific PM, the limitation of that PM should be met, which means that it is not possible to copy different VMs on a PM more than its storage capacity. On the other hand, more than one copy of one VM cannot be placed on the same PM. The time needed to check the available capacity of one PM is a constant number.

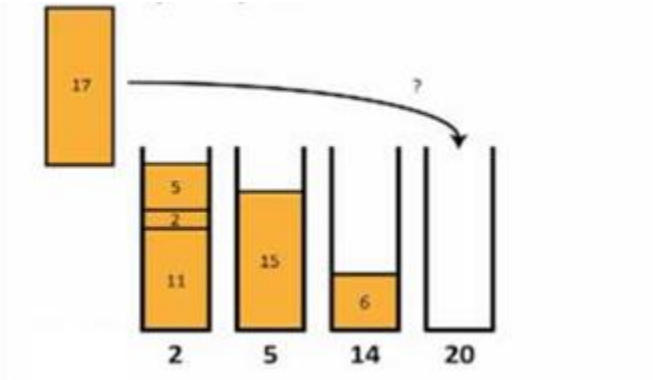


Fig.7. First-Fit Algorithm [28]

Greedy Algorithm

In the greedy algorithm, each replica copy of a VM is placed on the closest PM, which results in less power consumption for copying and allocating VM on different PMs. Again, here both constraints of VMs and PMs should be met. This allocation continues until all copies of different VMs are placed in their appropriate PMs. Most of the time, the greedy algorithm is not able to find the best solution for the whole problem because at each point, it just focuses on finding the best answer for that situation, and it does not care about finding the best overall answer.

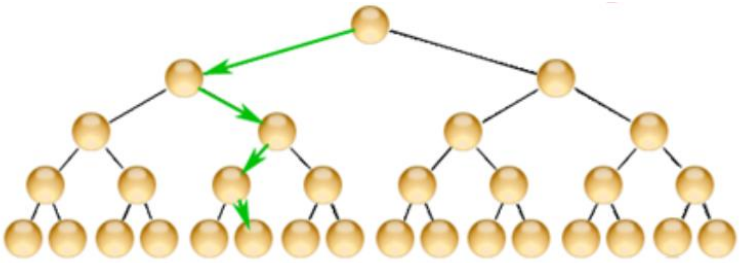


Fig.8. Greedy Algorithm [29]

Server Consolidation Algorithm

Remember that after all replications are done, all those physical machines are empty, which means that they are inactive and will be turned off. In server consolidation [6], we plan to create more inactive physical machines from the left active physical machines and turn them off to save energy and have a more efficient data center. The key factor for this movement is just looking at all PMs one by one and trying to find a new active PM as the target for each VM of that PM. We continue this process until we can move all VMs of a specific PM and turn it off.

There is one basic rule in this process, and it is protecting the first replication cost. It means that during consolidation [23], the cost of moving that VM to a new target should be the same as the original replication cost. On the other hand, storage and replication constraint should be met too.

As an example, in Fig. 5, it is possible to move two replication copies that are placed in physical machine numbers 13 and 14 to one of those PMs, such as PM# 13, and turn off the other one. Remember that in this example, this movement is possible because the replication cost is the same. In addition, these two are a replication copy of two

different VMs. On the other hand, PM #13 has enough space for both of these two VMs while maintaining the same total replication cost and satisfying the replication constraint of VMs (since they are replica copies of different original VMs). By turning off this physical machine, saving more energy in the data center would be possible.

Moving original VMs is not allowed in server consolidation. In addition, note that it is not possible to move VMs to any available PM because the main goal is to protect the original replication cost.

There are some definitions in server consolidation: physical machine X can be a potential target for replicating one VM if (1) the cost of moving is the same as the original replication cost, (2) if that PM has enough space for storing this new VM , and (3) if it does not store a copy or the original version of the same VM.

Consolidating physical machine (CPM). We can consider a physical machine a CPM if it does not store a source VM and it is active, which means that it just has some copies of different VMs. Such PM has the potential to be turned off and inactivated if we can move all its VMs to different PMs by meeting all the limitations and constraints.

Fig.9. Server Consolidation [30]

Existing Consolidation Algorithm and Its Drawbacks

The existing consolidation algorithm that was presented in Payman Khani's paper works as follows:

It starts to check the situation of all PMs from the first one. If there are three or fewer VMs, it tries to find another active PM for each VM. As soon as it can find at least one potential PM that meets all the storage and replication constraints, it moves that VM to the new target PM. After checking all the VMs of that specific PM, it checks whether the algorithm was able to move all the VMs of the PM, and that physical machine will be turned off and inactivated.

Note that in server consolidation, the ultimate location of the VM replica copies will be determined after the server consolidation is done, which means that the VM replicas are not actually located in the data center after VM replication. In fact, the server consolidation algorithm will further try to find a new solution to consolidate some physical machines and turn off those more inactive PMs. The duplicated copies of each VM will be finally transferred from their source PMs to the ultimate destination PMs that were determined as the new target for each VM after running the server consolidation algorithm.

Algorithm 1: Server Consolidation Algorithm.
Input: VM replica placement from VM replication algorithm
Output: Number of IPMs.

```
0.  Notations:
    m: largest number of replica VMs a CPM stores
    N_ipm = 0: number of IPMs
1.  for (i = 1 to m)
2.    for each of the CPMs that has i replica VMs
3.      flag = true;
4.      for each of the replica VMs
5.        if it can find a TPM
6.          move the replica VM to the TPM
7.        else
8.          flag=false;
9.          break;
10.       end if;
11.     end for;
12.     if (flag = true)
13.       N_ipm + +; /*This CPM can be turned off */
14.   end for;
15. end for;
16. RETURN N_ipm. /*Return number of inactive PMs */
```

Fig.10. Existing server consolidation algorithm [23]

The current algorithm has the following drawbacks:

1. It just checks those PMs that have three or fewer VMs.

2. It moves any VM to another PM only if the cost remains the same regardless of whether it is possible to turn off that PM or not.

3. It does not check the status of the target PM.

In our project, we propose the following set of improved PM consolidation algorithms to resolve the problem of the existing solution and improve the result. In the following, I will explain each of the five algorithms and their positive points.

Proposed Consolidation Algorithms

In this project, five algorithms are proposed.

1. Dynamic_Consolidation

2. OptimizedDynamic_Consolidation

3. Sorted_Consolidation

4. MostFilledPM_Consolidation

5.SortedMostFilledPM_Consolidation

Dynamic_Consolidaton Algorithm

In the first algorithm, we check all PMs to see whether it is possible to move the VMs to another PM regardless of the number of VMs that are copied in that PM.

OptimizedDynamic_Consolidation Algorithm

This algorithm includes the first one too. It means that we check all PMs regardless of their VM numbers. In the second algorithm, we check all VMs in one PM and move its VMs only if it is possible to move all of them. Otherwise, we do not move any of them.

Sorted_Consolidation Algorithm

In the third algorithm, we check PMs in an ascending order based on the number of VMs on the PMs; that is, we start with those PMs that have just one VM, and then those with two VMs, and so on. Again, this algorithm includes the first two algorithms.

MostFilledPM_Consolidation Algorithm

In the fourth algorithm, we move the VMs of a PM to a target PM that has the most number of VMs. Sorted_Consolidation includes the first two algorithms as well.

SortedMostFilledPM_Consolidation

It is combination of Sorted_Consolidation and MostFilledPM_Consolidation. This algorithm acts like a tuning part of the second one, which is

OptimizedDynamic_Consolidation. This is the final solution, and it includes all the other four algorithms.

Consolidation Algorithm Specifications

In all these algorithms, we preserve the original replication cost and storage and virtual machine constraints, which means that, during consolidation, we can move one VM to a new PM if and only if the transition cost to the new target PM is the same as the old location, which means that we will find the PM location of the original file of the VM and the cost of moving from that PM to the first place. After that, we will check all PMs that have the same cost of copying from the original location to those PMs.

The other consideration concerns VM replication and protecting fault tolerance, which means that it is not possible to have more than one copy of a VM on a specific PM. As a result, the total number of copies of a PM cannot be more than the number of PMs – 1. In addition, we cannot copy VMs on a PM more than its storage capacity.

For comparing different consolidation results, we run three replication algorithms—minimum-cost flow, first-fit, and greedy—to scatter all the original virtual machines; and then we run different PM consolidation algorithms under different data center scenarios with different physical machines and virtual machine numbers, different numbers of switch ports, and different numbers of copies of each virtual machine. We will compare the result, which is the number of those PMs that can be turned off on each algorithm, and compare them with the existing algorithm to see which of them works better.

Detailed Explanation of the Proposed Algorithms

Dynamic_Consolidaton Specifications

In the first step, all VMs should be scattered through the data center, and we run all the three VM allocation algorithms that were mentioned previously. The input of this program is the number of VMs, the number of switch ports of each physical machine, and the number of copies for each VM. By having this data, the total number of physical machines is calculated. Note that R, which is the number of copies for each VM, cannot be more than P, where P is the number of physical machines, which is calculated based on the switch port by using the $k^3/4$ formula. The result, which is the number of active PMs after replicating all the VMs, is different in these three algorithms. After replicating all the VM copies, the consolidation algorithm should run on the data center to turn off all potential PMs.

As mentioned, the existing consolidation algorithm works on those physical machines that have three or fewer virtual machines. The Dynamic_Consolidaton algorithm checks all the PMs from the first to the last regardless the number of VMs that are copies on it.

For each PM, first, it makes sure that there is no original copy of any VM on this machine because if it has even just one original VM, we do not need to try to move the VMs of this PM since we cannot turn off one PM that has at least one original copy. Then, it starts with the first VM, which is copied on that. In the beginning, it finds the original location of this VM to calculate the original cost of the replication of that VM on the first PM. And then it checks all the PMs from the beginning to the end to find a new

location for that VM. Moving the VM replication to this new location should meet all the storage and VM constraints. This limitation includes the following rules:

1. It is not possible to have more than one copy of a VM on one physical machine, which means that any target PM should be checked to make sure there is not already a copy of the same VM on that PM.

2. It is not possible to copy different VMs on one PM more that its capacity, which means that before moving, the free space of the PM should be checked to make sure that it has enough free space.

3. The cost of moving to this new location should be the same as the cost of moving to the original PM.

4. The new place should not be already turned off. Otherwise, there is no point moving one VM to a PM that was already turned off.

If this new location can be found and meets all the limitations, this VM is moved to this new place and a free space of the current PM and the new PM will be updated. The same process continues for all the VMs on this PM, and in the end, it checks whether it was possible to move all the VMs of this PM, and it inactivates this PM and turns it off. In the following, you can see the pseudocode of the program.

Input: VM replica placement from VM replication algorithm

Output: Number of IPMs.

0. Notations:

m: Last number of PM

Nipm = 0: number of IPMs

```
1.      for (PM number i = 1 to m)
2.          for each of VM on PM number i
3.              flag = true;
4.                  if it can find a TPM
5.                      move the replica VM to the TPM
6.                  else
7.                      flag=false;
8.                      break;
9.              end if;
10.         end for;
11.         if (flag = true)
12.             Nipm + +; /*This CPM can be turned off */
13.         end if;
14.     end for;
15.     RETURN Nipm. /*Return number of inactive PMs */
```

OptimizedDynamic_Consolidation Specifications

This algorithm includes the first one, which means that we try to consolidate all PMs regardless the number of VMs on them. And we follow the same process steps mentioned in the Dynamic_Consolidation algorithm, which means that, first, all VM replications should be scattered throughout the whole data center by using the minimum-cost flow algorithm, and then run the optimized consolidation to turn off potential PMs. The difference between this algorithm and the previous one is in checking the number of

moved VMs on each PM at the end of the PM consolidation, which means that after checking the situation of all VMs on one PM and moving those that have a potential target, we check whether the number of moved VMs is the same as all VMs on that PM. If yes, it will be turned off; otherwise, it will return all the previously moved PMs to their original locations and then check the situation of the next PM. At the end, it will check how many PMs were turned off. If we do not return those moved VMs without the possibility of turning off that PM, basically, we have changed the specification of the data center without the possibility of turning off that PM. In the following, you will see the pseudocode of the program.

Input: VM replica placement from VM replication algorithm

Output: Number of IPMs.

0. Notations:

m: Last number of PM

Nipm = 0: number of IPMs

```
1.          for (PM number i = 1 to m)
2.              for each of VM on PM number i
3.                  flag = true;
4.                   if it can find a TPM
5.                       move the replica VM to the TPM
6.                  else
7.                      flag=false;
8.                      break;
```

```
9.              end if;
10.           end for;
11.           if (flag = true)
12.              Nipm + +; /*This CPM can be turned off */
13.           else
14.              Return back all the moved VMs to their original locations.
15.          end if;
16.       end for;
17.       RETURN Nipm. /*Return number of inactive PMs */
```

Sorted_Consolidation Specifications

This algorithm includes the first two algorithms, which means that we try to consolidate all the PMs regardless the number of VMs on that PM. In addition, if it was not possible to move all the VMs of one PM, we return all the moved VMs to their original place to avoid changing the data center specification for no reason. But in Sorted_Consolidation, we check the PMs in a sorted way based on the number of VMs on them; that is, we start with those PMs that have just one VM, and then those with two VMs, and so on. After moving all the VMs of one PM and turning off the PM, we resort the order of the servers based on their VM numbers to have the updated situation of the whole data center. In this algorithm, we consider the situation of the source server. In the following, you can see the pseudocode of the program.

Input: VM replica placement from VM replication algorithm

Output: Number of IPMs.

0. Notations:

m: Last number of PM

SortedPM Array: this array stores the PM in a sorted way based on their VM numbers

Nipm = 0: number of turned off PMs

```
1.          for each of the SortedPM Array cell
2.              for replication in SortedPM Array
3.                  flag = true;
4.                      if it can find a TPM
5.                          move the replica VM to the TPM
6.                          Change SortedPM data based on this new movement
7.                          Resort SortedPM array
8.                      else
9.                         flag=false;
10.                         break;
11.                     end if;
12.             end for;
13.             if (flag = true)
14.                 Nipm + +; /*This CPM can be turned off */
15.             else
14.                 Return back all the moved VMs to their original locations.
```

16. end if;

17. end for;

18. RETURN Nipm. /*Return number of inactive PMs */

MostFilledPM_Consolidation Specifications

This algorithm includes the first two algorithms. Again, we check all PMs regardless their VM numbers and do not move any VM of one PM if we cannot move all of them. The part that has been added to the previous ones concerns checking all potential target PMs and then comparing the number of their VMs. In the existing algorithms, as soon as we find a potential target that meets all the requirements, the VM is moved to that PM. But in this algorithm, first, we find all of them and then compare their VM numbers to find the one with more VMs. The reason for this is that the probability of turning off the PM with more VMs is less than the probability of turning off one PM with fewer VMs. As a result, we move VMs to the PM that is less likely to be turned off. In this algorithm, we consider the situation of the target server despite the previous one where we considered the situation of the source PM. In the following, you can see the pseudocode of the program.

Input: VM replica placement from VM replication algorithm

Output: Number of IPMs.

0. Notations:

m: Last number of PM

n: Maximum of VM number in one PM

K: Best target which is the PM with the most VM.

TargetVMs array[][] : A two dimensional array which stores PM number and its associated VM numbers of all capable targets

Nipm = 0: number of IPMs

```
1.      for (PM number i = 1 to m)
2.          for VM number j = 1 to n)
3.              flag = true;
4.                  if it can find a TPM
5.                      TargetVMs[Target][0] = The index of Target PM.
6.                      TargetVMs[Target][1] = The VM number of that Target
7.                     flag=false;
8.                     break;
9.                end if;
10.               k= The index of Target PM with more VMs
11.               Move the replica VM to the TPM number k
12.           end for;
13.           if (flag = true)
14.               Nipm + +; /*This CPM can be turned off */
15.           else
16.               Return back all the moved VMs to their original locations.
17.       end if;
18.   end for;
```

19. RETURN Nipm. /*Return number of inactive PMs */

SortedMostFilledPM_Consolidation Specifications

In this algorithm, we combine algorithm numbers 4 and 4 to get the best result, which means that not only we start to check the situation of the source PM in a sorted way based on the number of VMs on one PM but also we check the situation of the target PMs and move the VM to the target PM that has the most number of VMs. In the following, you can see the pseudocode of the program.

Input: VM replica placement from VM replication algorithm

Output: Number of IPMs.

0. Notations:

m: Last number of PM

n: Maximum of VM number in one PM

K: Best target which is the PM with the most VM.

TargetVMs array[][] : A two dimensional array which stores PM number and its associated VM numbers of all capable targets

SortedPM Array: this array stores the PM in a sorted way based on their VM numbers

Nipm = 0: number of IPMs

1. for each of the SortedPM cells

2. for each of the cell of SortedPM array that has i replica VMs

```
3.              flag = true;
4.                 if it can find a TPM
5.                    TargetVMs[Target][0] = The index of Target PM.
6.                    TargetVMs[Target][1] = The VM number of that Target
7.                 else
8.                    flag=false;
9.                    break;
10.                end if;
11.               k= the index of the maximum number of cells in array TargetVMs
12.               move the replica VM to the TPM number k
13.               Change SortedPM data based on this new movement
14.               Resort SortedPM array
15.            end for;
16.            if (flag = true)
17.               Nipm + +; /*This CPM can be turned off */
18.            else
19.               Return back all the moved VMs to their original locations.
20.         end for;
21.      end for;
22.      RETURN Nipm. /*Return number of inactive PMs */
```

CHAPTER 4

PERFORMANCE EVALUATION

Running the Program on Different Data Centers

To test the program extensively with a different data center specification, all the VMs were scattered through the data centers by using the minimum-cost flow, first-fit, and greedy algorithms; and in the next step, I ran a server consolidation program to turn off more PMs. To have a more accurate output, I ran it five times for each data center specification and calculated the average.

Consolidation on MCF Replicated Data Center

In this sample first Vms are replicated by MCF and then consolidation is running. As an example of a data center with 200 virtual machines, 16 switch ports, and 5 copies for each virtual machine, I ran the program five times and calculated the average output. In the following, you can see the output for thirteen different data centers. AMP refers to the active number of PMs after running the replication algorithm. Existing refers to the number of turned-off PMs after running the existing consolidation algorithm. OD is the number of turned-off PMs after running OptimizedDynamic_Consolidation. SMFT refers to the number of turned-off PMs after running SortedMostFilledPM_Consolidation. Cost is the original cost of the virtual machine replication, FNP is the final number of active PMs after running the consolidation, and FC is the final cost after the consolidation, which is the sum of the replication cost and the final number of active PMs because we considered one unit for keeping one PM on.

Table 1
Consolidation Results in MCF Replication (k=16, R=5, VM=100)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	564	20	20	20	400	544	944
SecondOutput	570	18	18	18	400	552	952
ThirdOutput	524	19	19	19	400	505	905
FourthOutput	592	21	21	21	400	571	971
FifthOutput	554	17	17	18	400	536	936
Average	560	19	19	19.2	400	541	941
PercentImprovement			0.00	1.05			

Chart 1
Trend of Average Turned-off PMs in MCF Replication (k=16, R=5, VM=100)

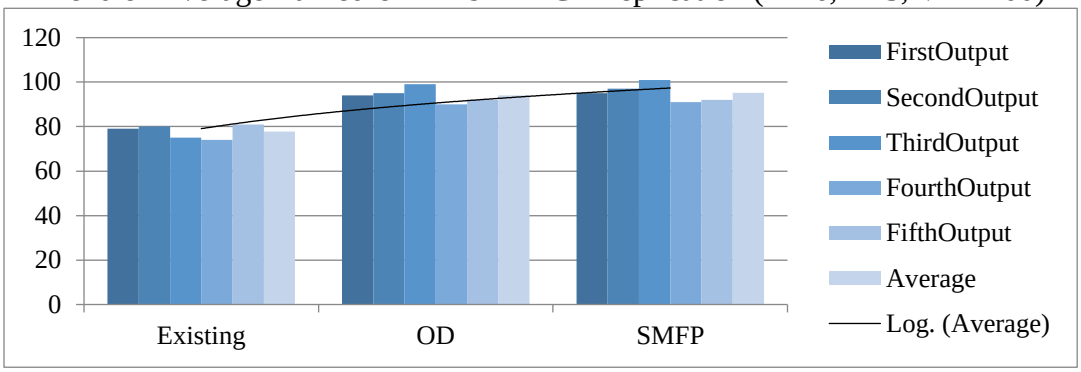


Table 2
Consolidation Results in MCF Replication (k=16, R=5, VM=300)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	699	79	94	95	1200	604	1804
SecondOutput	700	80	95	97	1280	603	1883
ThirdOutput	650	75	99	101	1240	549	1789
FourthOutput	660	74	90	91	1290	569	1859
FifthOutput	680	81	92	92	1250	588	1838
Average	678	78	94	95	1252	583	1835
PercentImprovement			20.82	1.28			

Chart 2
Trend of Average Turned-off PMs in MCF Replication (k=16, R=5, VM=300)

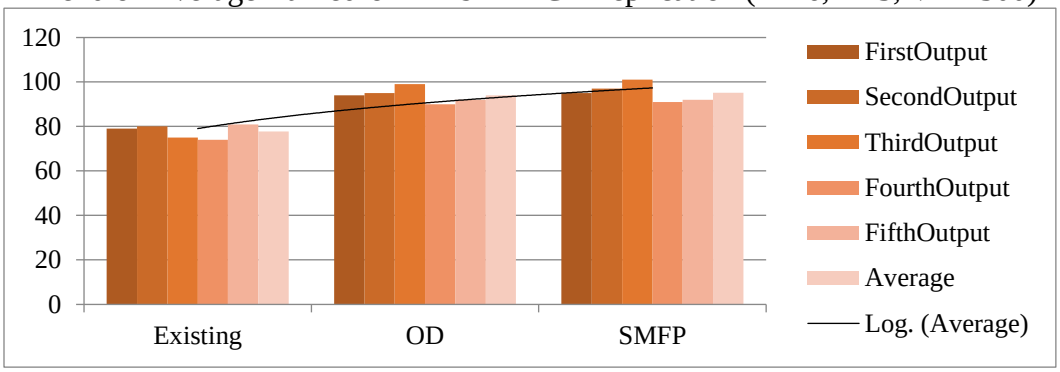


Table 3
Consolidation Results in MCF Replication (k=16, R=5, VM=400)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	749	86	126	129	1600	620	2220
SecondOutput	714	75	112	113	1620	601	2221
ThirdOutput	745	86	115	118	1668	627	2295
FourthOutput	736	80	110	111	1676	625	2301
FifthOutput	737	88	122	127	1679	610	2289
Average	736	83	117	120	1649	617	2265
PercentImprovement			40.96	2.22			

Chart 3
Trend of Average Turned-off PMs in MCF Replication (k=16, R=5, VM=400)

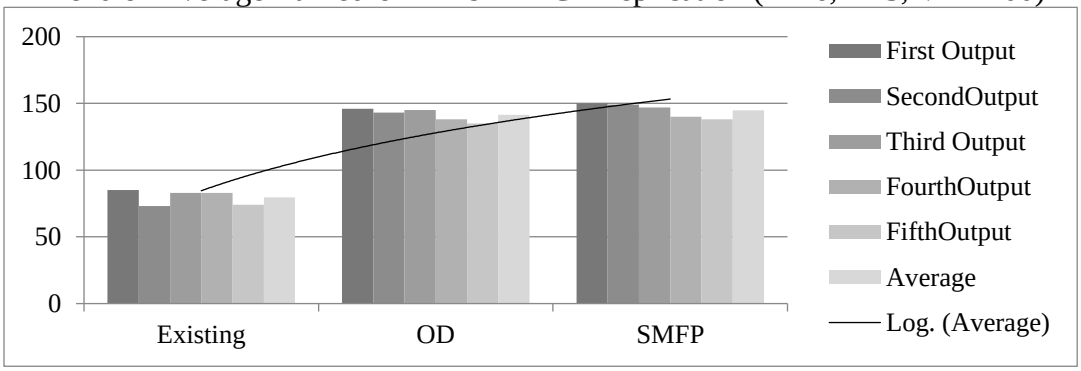


Table 4
Consolidation Results in MCF Replication (k=16, R=5, VM=500)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	787	85	146	150	2292	637	2929
SecondOutput	787	73	143	149	2340	638	2978
ThirdOutput	772	83	145	147	2260	625	2885
FourthOutput	781	83	138	140	2350	641	2991
FifthOutput	762	74	135	138	2308	624	2932
Average	778	80	141	145	2310	633	2943
PercentImprovement			77.64	2.40			

Chart 4
Trend of Average Turned-off PMs in MCF Replication (k=16, R=5, VM=500)

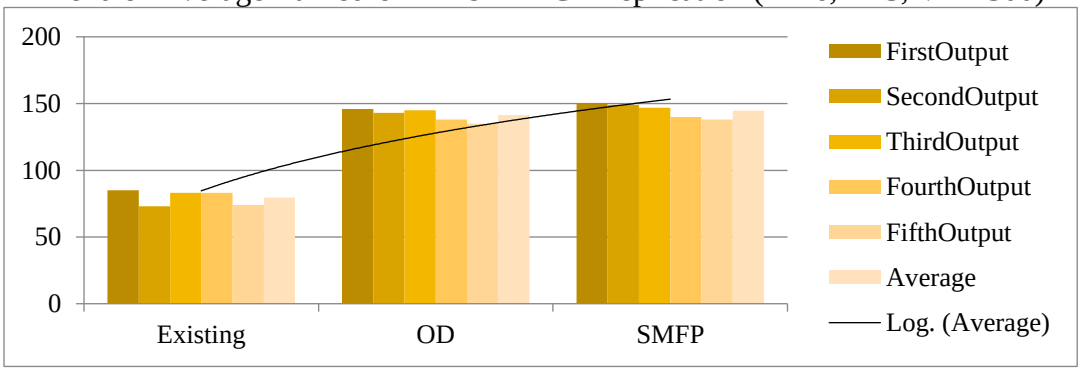


Table 5
Consolidation Results in MCF Replication (k=16, R=5, VM=600)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	826	68	165	170	3012	656	3668
SecondOutput	833	82	164	170	3044	663	3707
ThirdOutput	819	74	154	158	2860	661	3521
FourthOutput	823	87	166	172	2964	651	3615
FifthOutput	805	71	151	154	3052	651	3703
Average	821	76	160	165	2986	656	3643
PercentImprovement			109.42	3.00			

Chart 5
Trend of Average Turned-off PMs in MCF Replication (k=16, R=5, VM=600)

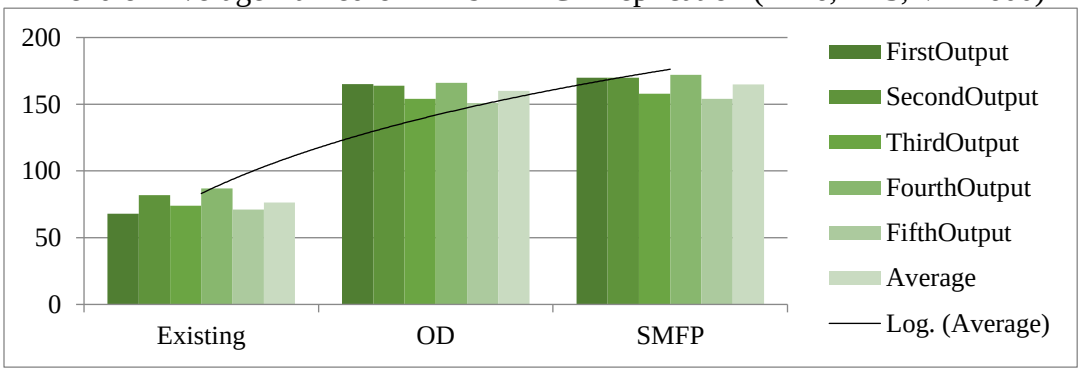


Table 6
Consolidation Results in MCF Replication (k=16, R=5, VM=700)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	840	63	176	181	3772	659	4431
SecondOutput	834	80	159	164	3852	670	4522
ThirdOutput	837	85	166	174	3756	663	4419
FourthOutput	838	69	162	171	3700	667	4367
FifthOutput	842	81	172	178	3796	664	4460
Average	838	76	167	174	3775	665	4440
PercentImprovement			120.90	3.95			

Chart 6
Trend of Average Turned-off PMs in MCF Replication (k=16, R=5, VM=700)

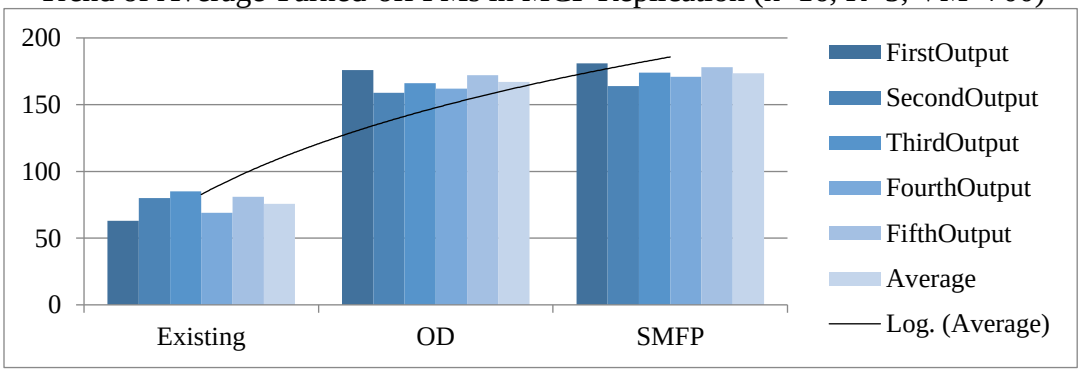


Table 7
Consolidation Results in MCF Replication (k=16, R=5, VM=800)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	844	91	200	207	4540	637	5177
SecondOutput	882	82	195	207	4644	675	5319
ThirdOutput	885	85	193	204	4356	681	5037
FourthOutput	882	90	194	204	4436	678	5114
FifthOutput	874	91	188	200	4431	674	5105
Average	873	88	194	204	4481	669	5150
PercentImprovement			120.96	5.36			

Chart 7
Trend of Average Turned-off PMs in MCF Replication (k=16, R=5, VM=800)

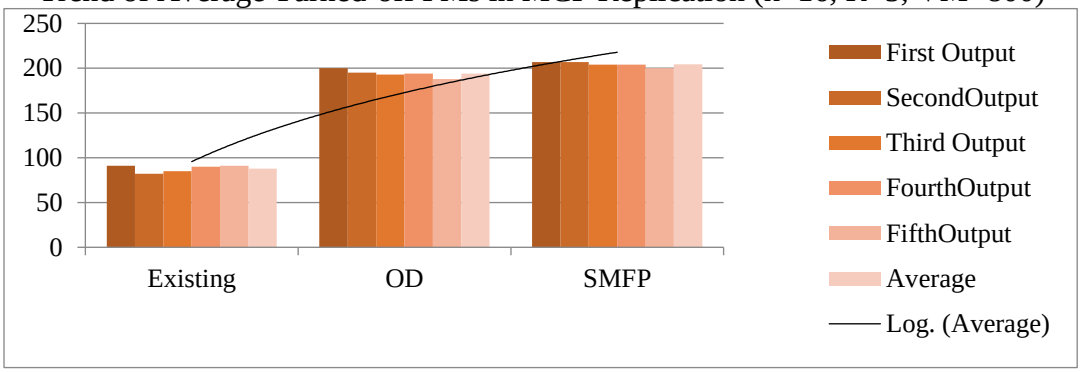


Table 8
Consolidation Results in MCF Replication (k=16, R=5, VM=900)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	897	83	190	200	3516	697	4213
SecondOutput	904	83	197	210	5332	694	6026
ThirdOutput	908	99	211	217	5412	691	6103
FourthOutput	901	82	193	205	5292	696	5988
FifthOutput	917	98	200	215	5244	702	5946
Average	905	89	198	209	4959	696	5655
PercentImprovement			122.70	5.65			

Chart 8
Trend of Average Turned-off PMs in MCF Replication (k=16, R=5, VM=900)

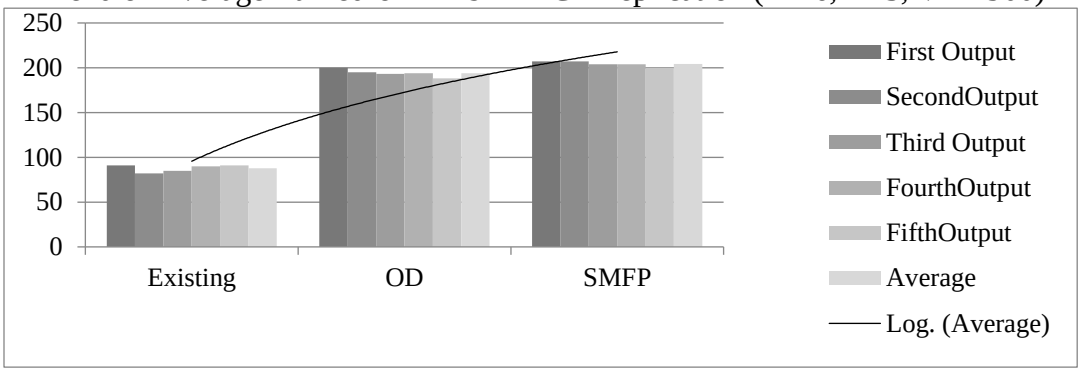


Table 9
Consolidation Results in MCF Replication (k=16, R=5, VM=950)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	917	82	210	216	5788	701	6489
SecondOutput	899	75	185	193	5780	706	6486
ThirdOutput	897	77	184	195	5716	702	6418
FourthOutput	914	89	200	214	5800	700	6500
FifthOutput	903	84	204	215	6044	688	6732
Average	906	81	197	207	5826	699	6525
PercentImprovement			141.52	5.09			

Chart 9
Trend of Average Turned-off PMs in MCF Replication (k=16, R=5, VM=950)

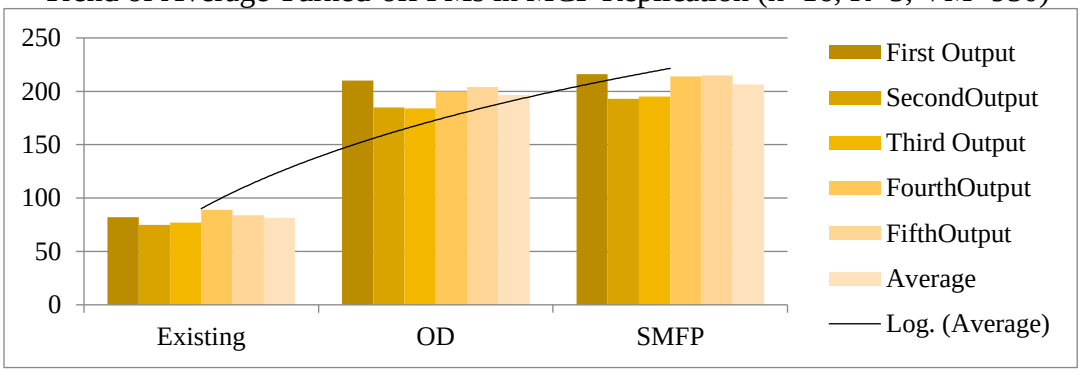
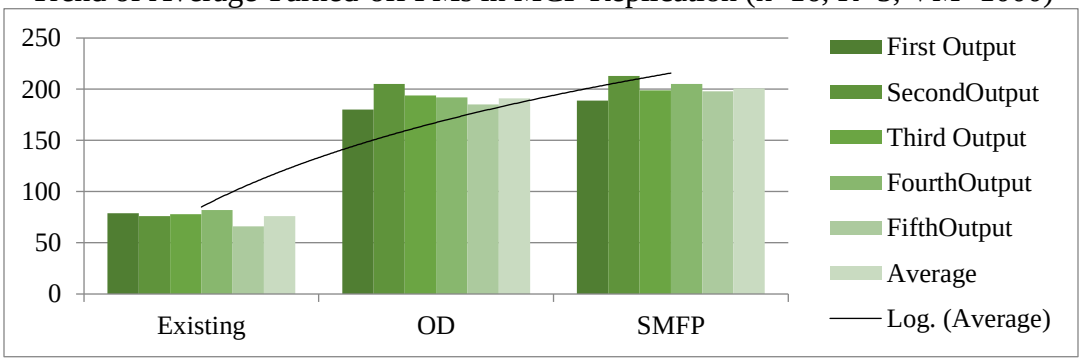


Table 10
Consolidation Results in MCF Replication (k=16, R=5, VM=1000)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	901	79	180	189	6124	712	6836
SecondOutput	921	76	205	213	6122	708	6830
ThirdOutput	909	78	194	199	6123	710	6833
FourthOutput	911	82	192	205	6120	706	6826
FifthOutput	913	66	185	198	6122	715	6837
Average	911	76	191	201	6122	710	6832
PercentImprovement			150.92	5.02			

Chart 10
Trend of Average Turned-off PMs in MCF Replication (k=16, R=5, VM=1000)

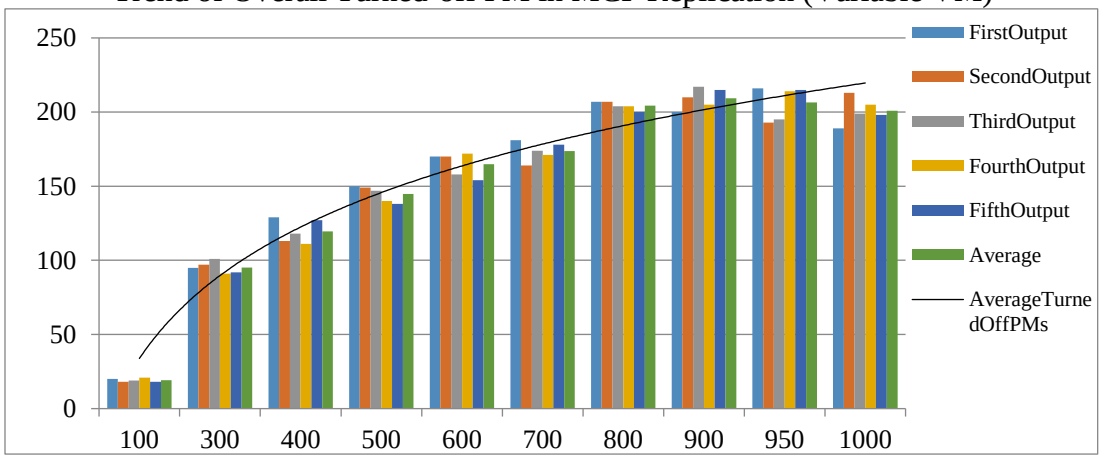


If you compare the different outputs in the different data centers, you will see that when the data center is pretty empty, the algorithm cannot turn off too many PMs. In addition, when the data center is becoming more and more crowded, the number of PMs can be turned off using the second algorithm, which acts like a tuning part that does not follow an increasing order, and it becomes almost constant. For example, when the data center has 900 virtual machines, number of PMs it can turn off is 209; however, for a data center with 950 virtual machines this number is 207, which means that in a very crowded data center, the tuning part does not play a specific role in the consolidation.

Table 11
Overall Turned-Off PM in MCF Replication (Variable VM)

VMs	100	300	400	500	600	700	800	900	950	1000
Final Turned Off PMs	20	95	129	150	170	181	207	200	216	189
	18	97	113	149	170	164	207	210	193	213
	19	101	118	147	158	174	204	217	195	199
	21	91	111	140	172	171	204	205	214	205
	18	92	127	138	154	178	200	215	215	198
Average	33	129	120	145	165	174	204	209	207	201

Chart 11
Trend of Overall Turned-off PM in MCF Replication (Variable VM)



To make the results clearer, I ran the program for a constant number of VMs and switch ports and changed the number of copies for each virtual machine.

Table 12
Consolidation Results in MCF Replication (k=16, R=2, VM=950)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	714	49	87	87	1120	627	1747
SecondOutput	714	49	87	87	1120	627	1747
ThirdOutput	714	49	87	87	1120	627	1747
Average	714	49	87	87	1120	627	1747
PercentImprovement			77.55	0			

Chart 12
Trend of Average Turned-off PMs in MCF Replication (k=16, R=2, VM=950)

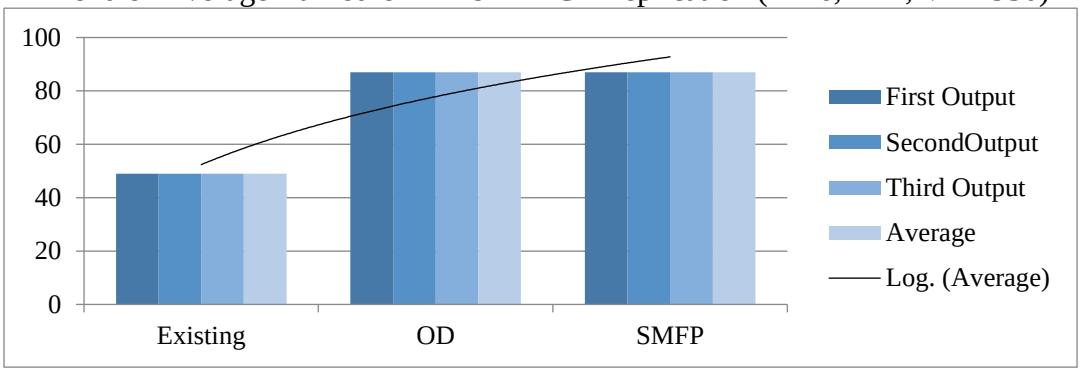


Table 13
Consolidation Results in MCF Replication (k=16, R=3, VM=950)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	803	84	162	165	2636	638	3274
SecondOutput	784	73	140	145	2648	639	3287
ThirdOutput	782	80	158	160	2684	622	3306
Average	790	79	153	157	2656	633	3289
PercentImprovement			94.09	2.17			

Chart 13
Trend of Average Turned-off PMs in MCF Replication (k=16, R=3, VM=950)

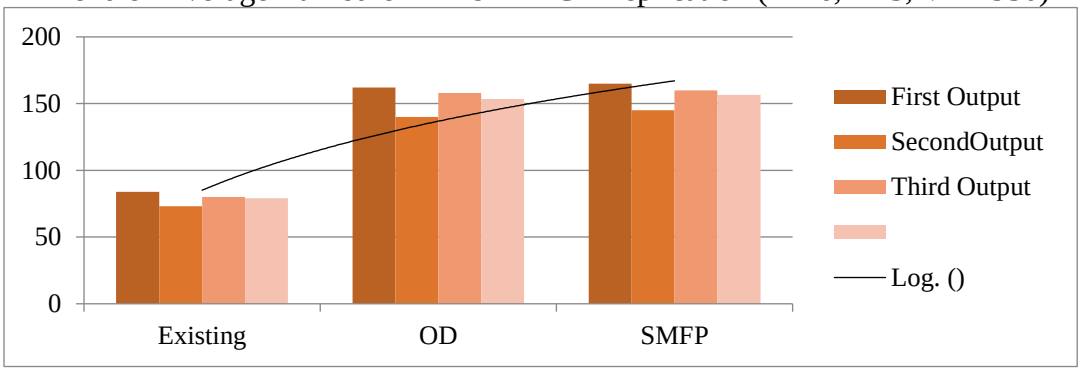
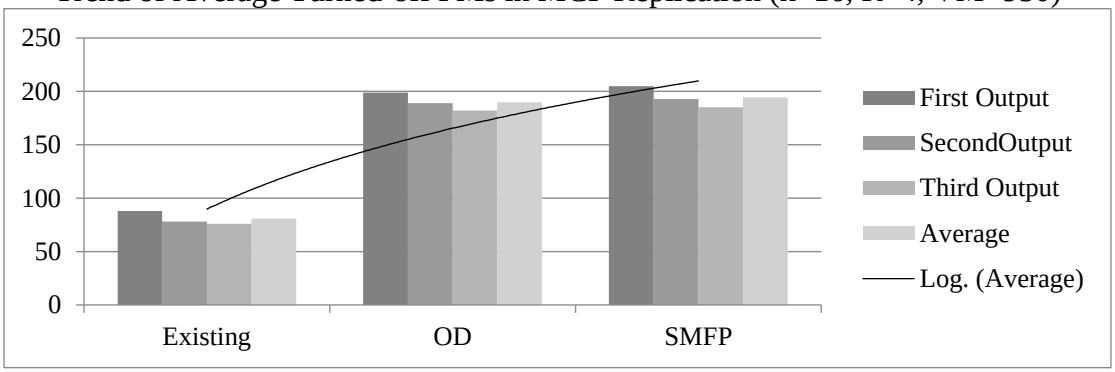


Table 14
Consolidation Results in MCF Replication (k=16, R=4, VM=950)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	862	88	199	205	4104	657	4761
SecondOutput	845	78	189	193	4242	652	4894
ThirdOutput	837	76	182	185	4218	652	4870
Average	848	81	190	194	4188	654	4842
PercentImprovement			135.54	2.28			

Chart 14
Trend of Average Turned-off PMs in MCF Replication (k=16, R=4, VM=950)

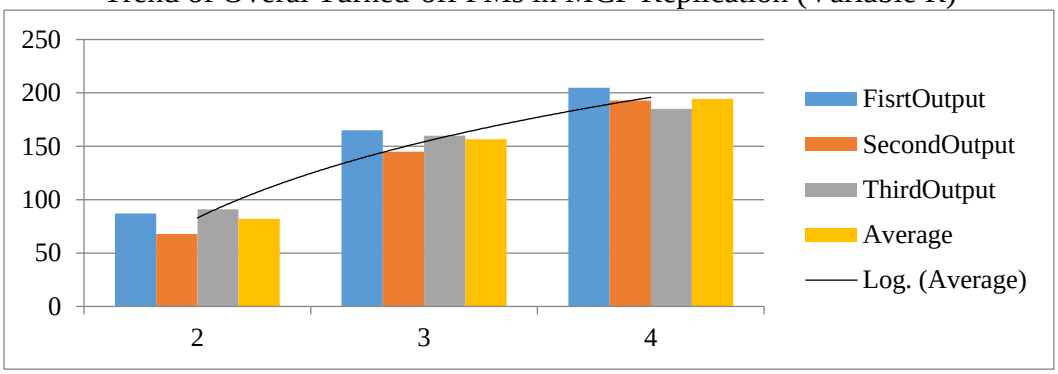


In the following table, you will find a comparison all turned-off VM numbers on different data centers.

Table 15
Overall Turned-Off PM in MCF Replication (Variable R)

Copy Number	2	3	4
Final Turned-Off PMs	87	165	205
	68	145	193
	91	160	185
Average	82	157	194

Chart 15
Trend of Overal Turned-off PMs in MCF Replication (Variable R)



Consolidation on First Fit Replicated Data Center

In this part, we use the first-fit algorithm to scatter all VMs on the data center instead of the minimum-cost algorithm and then running the consolidation program on them to see what the result will be and find the best solution for VM replication and PM consolidation. In the following, you will see some samples of the output result for different algorithms on a data center in which its VMs have been replicated by the first-fit algorithm.

Table 16
Consolidation Results in First-Fit Replication (k=16, R=5, VM=100)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	115	0	15	15	1994	100	2094
SecondOutput	113	0	15	15	1990	98	2088
ThirdOutput	114	0	14	14	1989	100	2089
FourthOutput	115	0	14	14	1990	101	2091
Average	114	0	15	15	1991	100	2091
PercentImprovement			1450	0			

Chart 16
Trend of Average Turned-off PMs in First Fit Replication (k=16, R=5, VM=100)

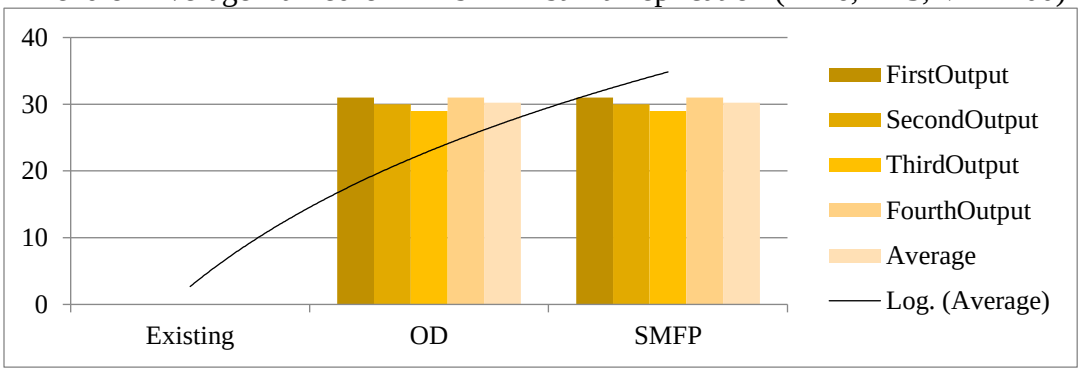


Table 17
Consolidation Results in First-Fit Replication (k=16, R=5, VM=300)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	285	0	31	31	5336	254	5590
SecondOutput	286	0	30	30	5340	256	5596
ThirdOutput	283	0	29	29	5342	254	5596
FourthOutput	286	0	31	31	5338	255	5593
Average	285	0	30	30	5339	255	5594
PercentImprovement			3025	0			

Chart 17
Trend of Average Turned-off PMs in First Fit Replication (k=16, R=5, VM=300)

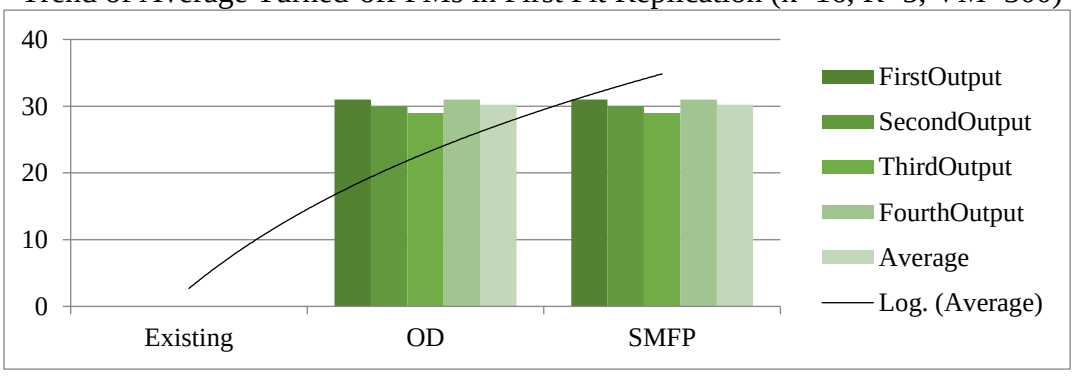


Table 18
Consolidation Results in First-Fit Replication (k=16, R=5, VM=400)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	372	0	40	40	7214	332	7546
SecondOutput	371	0	40	40	7180	331	7511
ThirdOutput	372	0	36	36	7212	336	7548
FourthOutput	336	0	37	37	7148	299	7447
Average	363	0	38	38	7189	325	7513
PercentImprovement			3825	0			

Chart 18
Trend of Average Turned-off PMs in First Fit Replication (k=16, R=5, VM=400)

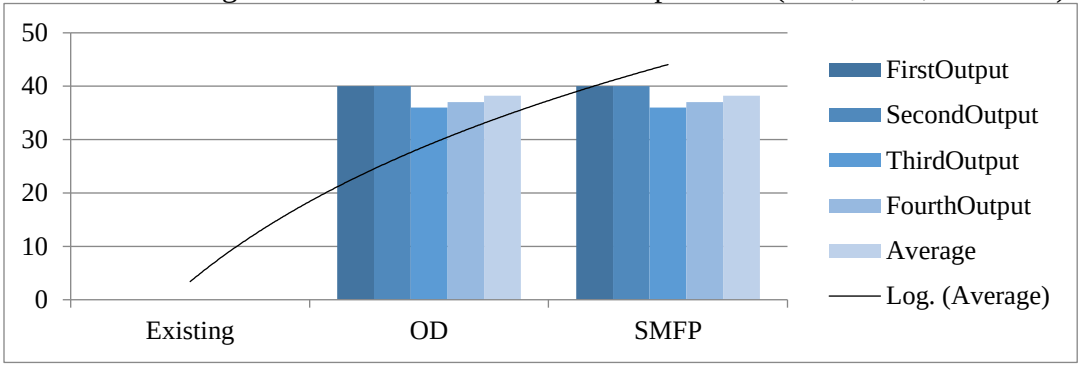


Table 19
Consolidation Results in First-Fit Replication (k=16, R=5, VM=500)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	426	0	35	35	8670	391	9061
SecondOutput	430	0	45	45	8708	385	9093
ThirdOutput	445	0	42	42	8972	403	9375
FourthOutput	431	0	38	38	8840	393	9233
Average	433	0	40	40	8798	393	9191
PercentImprovement			4000	0			

Chart 19
Trend of Average Turned-off PMs in First Fit Replication (k=16, R=5, VM=500)

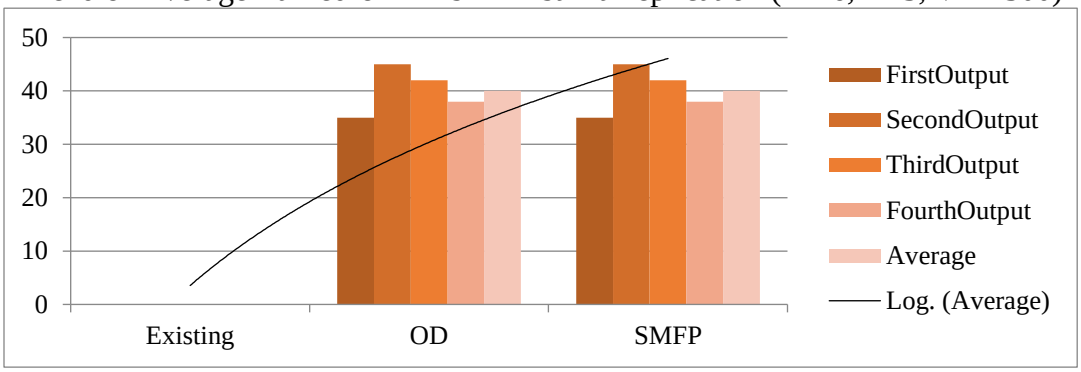


Table 20
Consolidation Results in First-Fit Replication (k=16, R=5, VM=600)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	504	0	46	46	10544	458	11002
SecondOutput	515	0	50	50	10588	465	11053
ThirdOutput	484	0	42	42	10300	442	10742
FourthOutput	526	0	49	49	10720	477	11197
Average	507	0	47	47	10538	461	10999
PercentImprovement			4675	0			

Chart 20
Trend of Average Turned-off PMs in First Fit Replication (k=16, R=5, VM=600)

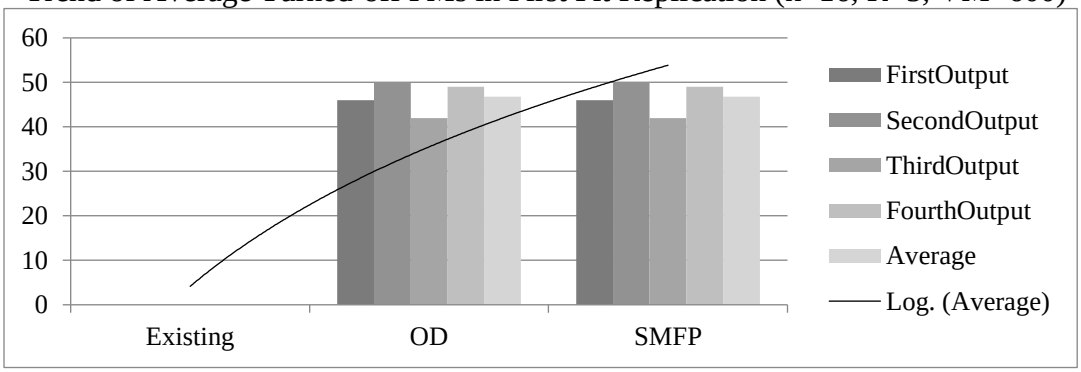


Table 21
Consolidation Results in First-Fit Replication (k=16, R=5, VM=700)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	565	0	50	50	12228	515	12743
SecondOutput	552	0	49	49	12150	503	12653
ThirdOutput	566	0	48	48	12266	518	12784
FourthOutput	549	0	46	46	12168	503	12671
Average	558	0	48	48	12203	510	12713
PercentImprovement			4825	0			

Chart 21
Trend of Average Turned-off PMs in First Fit Replication (k=16, R=5, VM=700)

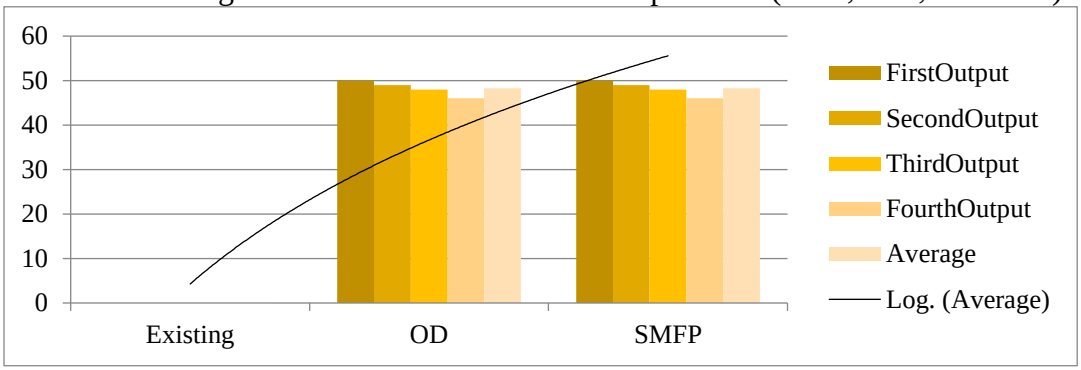


Table 22
Consolidation Results in First-Fit Replication (k=16, R=5, VM=800)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	619	0	52	52	14014	567	14581
SecondOutput	609	0	52	52	13942	557	14499
ThirdOutput	619	0	50	50	13920	569	14489
FourthOutput	590	0	53	53	13908	537	14445
Average	609	0	52	52	13946	558	14504
PercentImprovement			5175	0			

Chart 22
Trend of Average Turned-off PMs in First Fit Replication (k=16, R=5, VM=800)

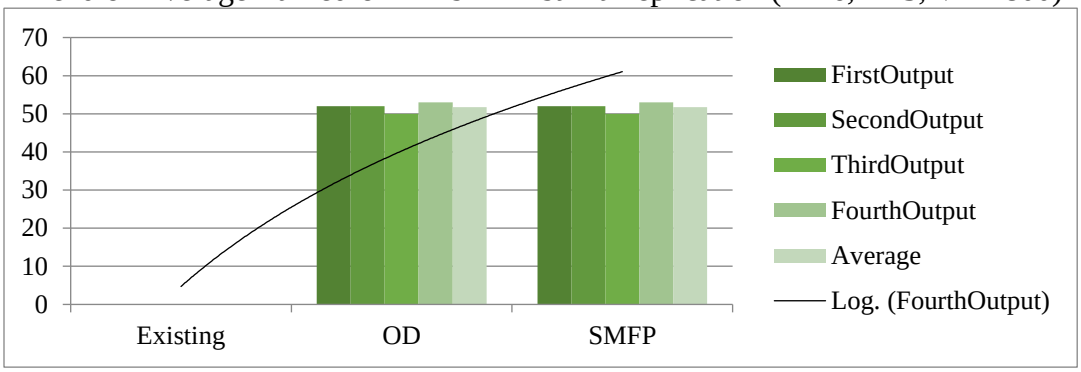


Table 23
Consolidation Results in First-Fit Replication (k=16, R=5, VM=900)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	643	0	53	53	15770	590	16360
SecondOutput	651	0	56	56	15800	595	16395
ThirdOutput	643	0	54	54	15860	589	16449
FourthOutput	650	0	52	52	15708	598	16306
Average	647	0	54	54	15785	593	16378
PercentImprovement			5375	0			

Chart 23
Trend of Average Turned-off PMs in First Fit Replication (k=16, R=5, VM=900)

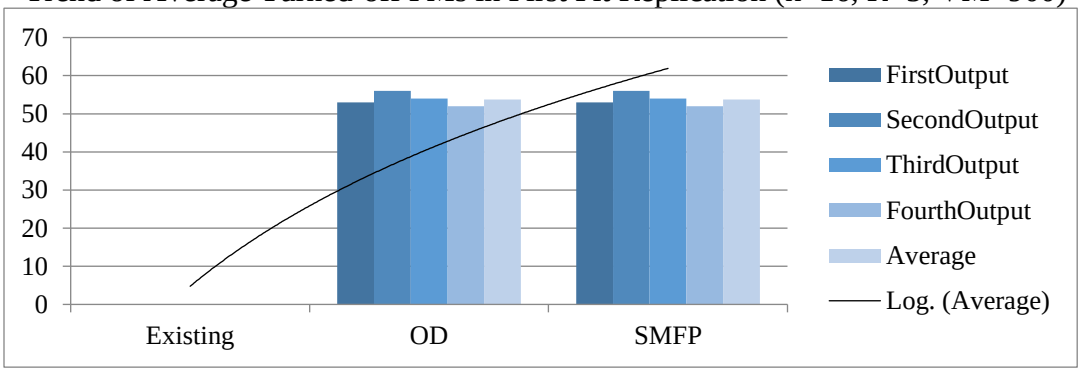


Table 24
Consolidation Results in First-Fit Replication (k=16, R=5, VM=950)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	673	0	54	54	16562	619	17181
SecondOutput	672	0	55	55	16518	617	17135
ThirdOutput	671	0	53	53	16500	618	17118
FourthOutput	681	0	53	53	16584	628	17212
Average	674.3	0.0	53.8	53.8	16541.0	620.5	17161.5
PercentImprovement			5375.00	0.00			

Chart 24
Trend of Average Turned-off PMs in First Fit Replication (k=16, R=5, VM=950)

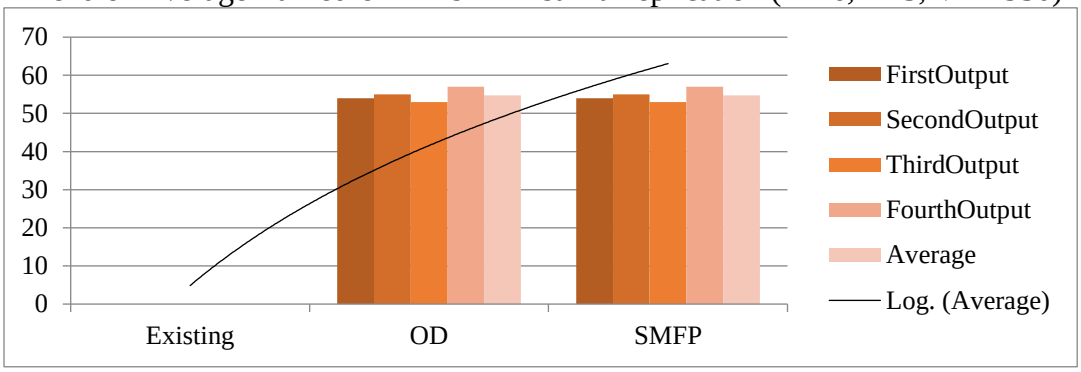
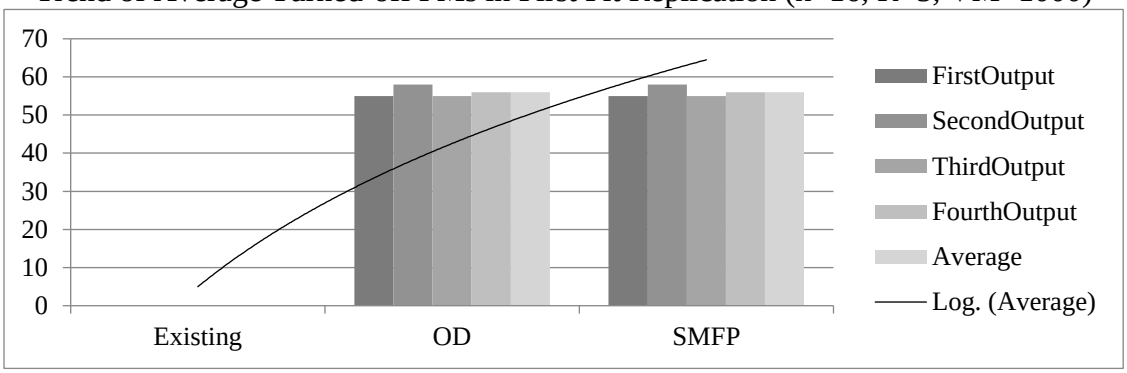


Table 25
Consolidation Results in First-Fit Replication (k=16, R=5, VM=1000)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	696	0	54	54	17628	642	18270
SecondOutput	692	0	52	52	17698	640	18338
ThirdOutput	688	0	55	55	17521	633	18154
FourthOutput	693	0	52	52	17532	641	18173
Average	692.3	0.0	53.3	53.3	17594.8	639.0	18233.8
PercentImprovement			5325.00	0.00			

Chart 25
Trend of Average Turned-off PMs in First Fit Replication (k=16, R=5, VM=1000)

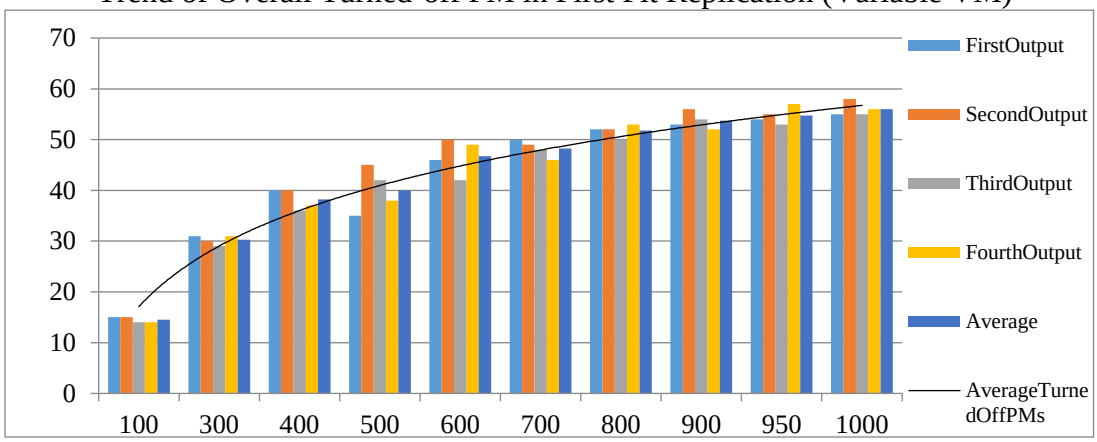


By using the first-fit algorithm to scatter original VMs and then running the consolidation algorithm, the trend of turning off PMs by using the first algorithm is again increasing. But the second algorithm, which is a tuning program, does not turn off any PMs. The reason is the first-fit method, which was used in the beginning for replication. Since it fills all PMs from the beginning and copies VMs in the first available place, there is no room for that tuning algorithm.

Table 26
Overall Turned-Off PM in First-Fit Replication (Variable VM)

VMs	100	300	400	500	600	700	800	900	950	1000
Final Turned-Off PMs	15	31	40	35	46	50	52	53	54	54
	15	30	40	45	50	49	52	56	55	52
	14	29	36	42	42	48	50	54	53	55
	14	31	37	38	49	46	53	52	53	52
Average	15	30	38	40	47	48	52	54	53	53

Chart 26
Trend of Overall Turned-off PM in First Fit Replication (Variable VM)



To make the results clearer, I ran the program for the constant number of VMs and switch ports and changed the number of copies for each VM.

Table 27
Consolidation Results in First-Fit Replication (k=16, R=2, VM=950)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	633	0	12	12	3798	621	4419
SecondOutput	631	0	12	12	3748	619	4367
ThirdOutput	610	0	9	9	3738	601	4339
Average	625	0	11	11	3761	614	4375
PercentImprovement			1100	0			

Chart 27
Trend of Average Turned-off PMs in First Fit Replication (k=16, R=2, VM=950)

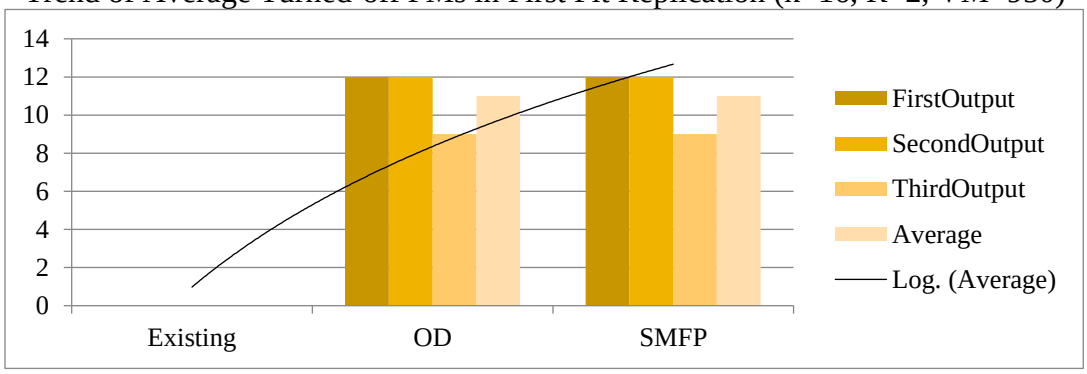


Table 28
Consolidation Results in First-Fit Replication (k=16, R=3, VM=950)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	646	0	26	26	7828	620	8448
SecondOutput	642	0	24	24	7796	618	8414
ThirdOutput	640	0	25	25	7818	615	8433
Average	643	0	25	25	7814	618	8432
PercentImprovement			2500	0			

Chart 28
Trend of Average Turned-off PMs in First Fit Replication (k=16, R=3, VM=950)

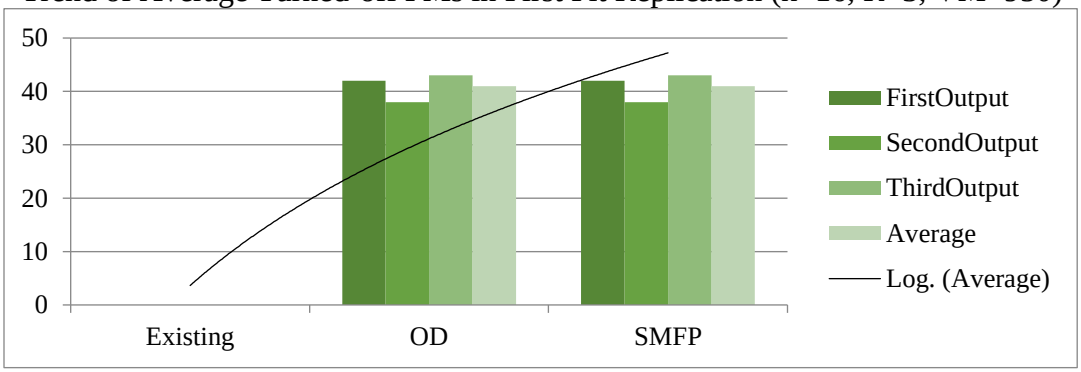
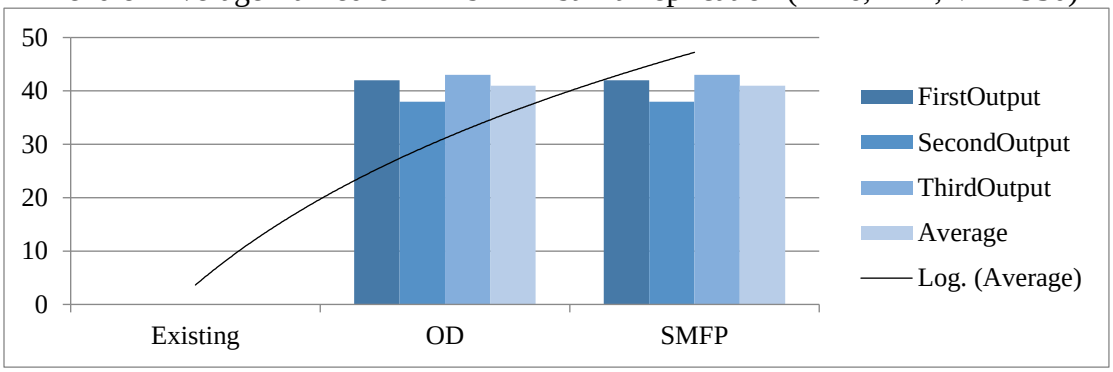


Table 29
Consolidation Results in First-Fit Replication (k=16, R=4, VM=950)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	671	0	42	42	12168	629	12797
SecondOutput	657	0	38	38	11948	619	12567
ThirdOutput	656	0	43	43	12054	613	12667
Average	661	0	41	41	12057	620	12677
PercentImprovement			4100	0			

Chart 29
Trend of Average Turned-off PMs in First Fit Replication (k=16, R=4, VM=950)

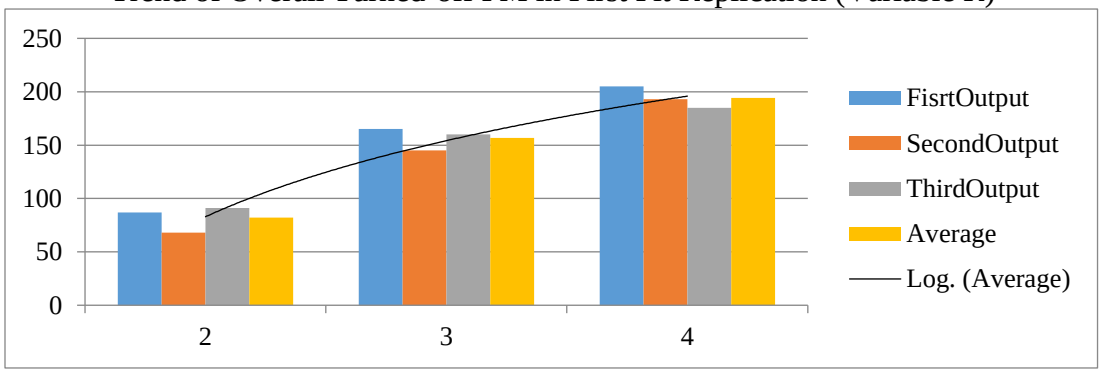


In the following table, you will find a comparison of all turned-off VM numbers on different data centers.

Table 30
Overall Turned-Off PM in First-Fit Replication (Variable R)

Copy Number	100	300	400
Final Turned-Off PMs	12	26	42
	12	24	38
	9	25	43
Average	11	25	32

Chart 30:
Trend of Overall Turned-off PM in First Fit Replication (Variable R)



Consolidation on Greedy Replicated Data Center

In this part, we use the greedy algorithm to scatter all VMs on the data center instead of the minimum-cost flow algorithm and then run the consolidation program on them to see what the result will be and find the best solution for VM replication and PM consolidation. In the following, you will see some samples of the output result for different data centers that have been replicated using the first-fit algorithm.

Table 31
Consolidation Results in Greedy Replication (k=16, R=5, VM=100)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	352	10	12	12	400	340	740
SecondOutput	369	19	19	19	400	350	750
ThirdOutput	350	15	15	15	400	335	735
FourthOutput	363	18	18	18	400	345	745
FifthOutput	354	20	20	20	400	334	734
Average	358	16	17	17	400	341	741
PercentImprovement			2.44	0.00			

Chart 31
Trend of Average Turned-off PMs in Greedy Replication (k=16, R=5, VM=100)

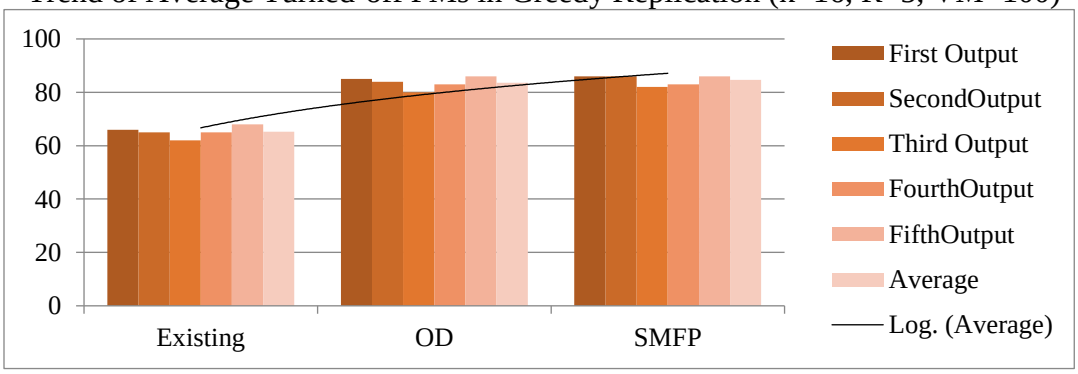


Table 32
Consolidation Results in Greedy Replication (k=16, R=5, VM=300)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	651	66	85	86	1214	565	1779
SecondOutput	648	65	84	86	1250	562	1812
ThirdOutput	640	62	80	82	1220	558	1778
FourthOutput	655	65	83	83	1230	572	1802
FifthOutput	660	68	86	86	1260	574	1834
Average	651	65	84	85	1235	566	1801
PercentImprovement			28.22	1.20			

Chart 31
Trend of Average Turned-off PMs in Greedy Replication (k=16, R=5, VM=300)

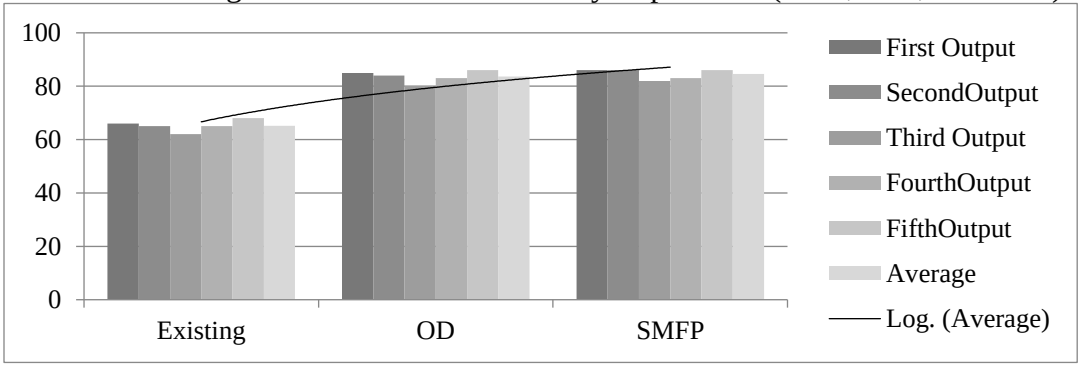


Table 33
Consolidation Results in Greedy Replication (k=16, R=5, VM=400)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	696	73	103	104	1656	592	2248
SecondOutput	747	76	118	120	1716	627	2343
ThirdOutput	729	69	108	109	1684	620	2304
FourthOutput	740	71	112	113	1700	627	2327
FifthOutput	742	64	104	104	1716	638	2354
Average	731	71	109	110	1694	621	2315
PercentImprovement			54.39	0.92			

Chart 33
Trend of Average Turned-off PMs in Greedy Replication (k=16, R=5, VM=400)

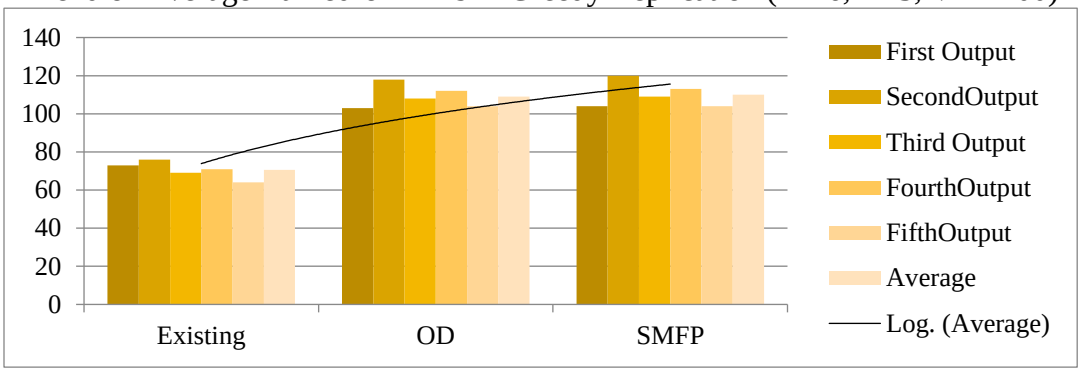


Table 34
Consolidation Results in Greedy Replication (k=16, R=5, VM=500)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	782	83	142	142	2332	640	2972
SecondOutput	765	72	139	141	2332	624	2956
ThirdOutput	767	76	127	128	2316	639	2955
FourthOutput	770	72	129	131	2322	639	2961
FifthOutput	779	75	147	149	2364	630	2994
Average	773	76	137	138	2333	634	2968
PercentImprovement			80.95	1.02			

Chart 34
Trend of Average Turned-off PMs in Greedy Replication (k=16, R=5, VM=500)

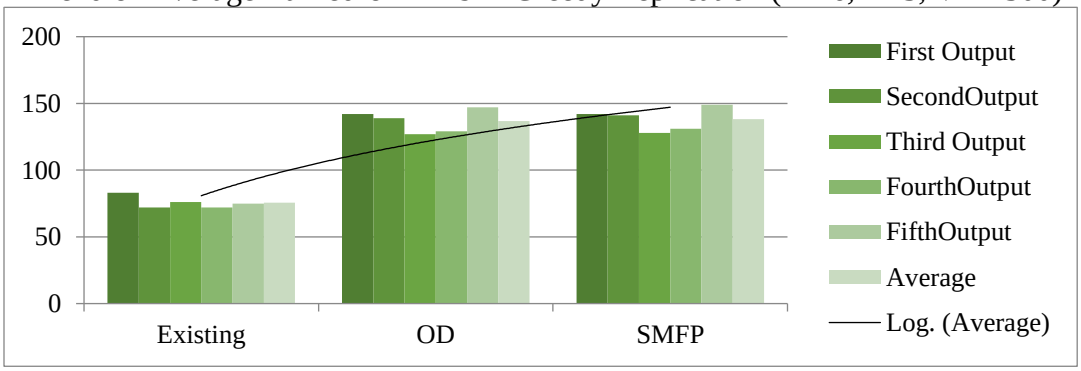


Table 35
Consolidation Results in Greedy Replication (k=16, R=5, VM=600)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	799	70	149	152	2988	647	3635
SecondOutput	805	77	151	155	3104	650	3754
ThirdOutput	786	66	135	138	3114	648	3762
FourthOutput	793	66	152	154	3080	639	3719
FifthOutput	805	80	153	158	3076	647	3723
Average	798	72	148	151	3072	646	3719
PercentImprovement			106.13	2.30			

Chart 35
Trend of Average Turned-off PMs in Greedy Replication (k=16, R=5, VM=600)

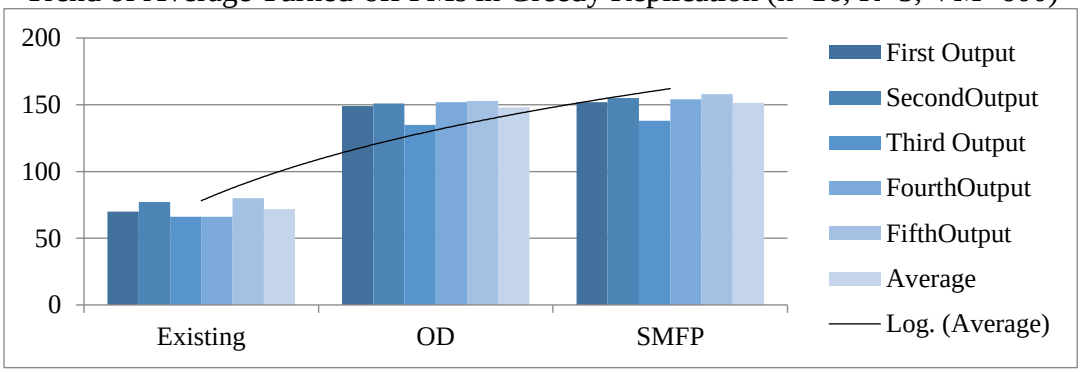


Table 36
Consolidation Results in Greedy Replication (k=16, R=5, VM=700)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	829	65	161	164	3672	665	4337
SecondOutput	834	69	171	171	3876	663	4539
ThirdOutput	829	59	162	164	3842	665	4507
FourthOutput	826	68	158	159	3718	667	4385
FifthOutput	831	63	160	161	3860	670	4530
Average	830	65	162	164	3794	666	4460
PercentImprovement			150.62	0.86			

Chart 36
Trend of Average Turned-off PMs in Greedy Replication (k=16, R=5, VM=700)

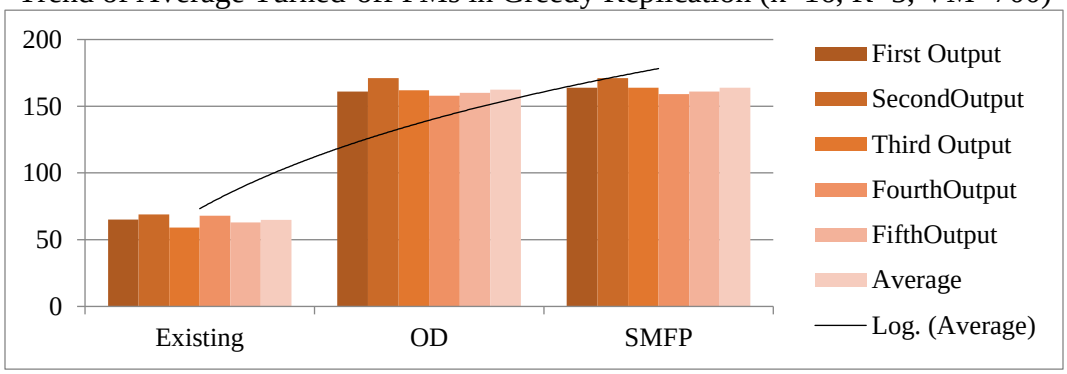


Table 37
Consolidation Results in Greedy Replication (k=16, R=5, VM=800)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	875	60	184	186	4436	689	5125
SecondOutput	851	60	166	169	4674	682	5356
ThirdOutput	859	59	176	181	4510	678	5188
FourthOutput	852	55	167	170	4580	682	5262
FifthOutput	851	54	179	181	4644	670	5314
Average	858	58	174	177	4569	680	5249
PercentImprovement			202.78	1.72			

Chart 37
Trend of Average Turned-off PMs in Greedy Replication (k=16, R=5, VM=800)

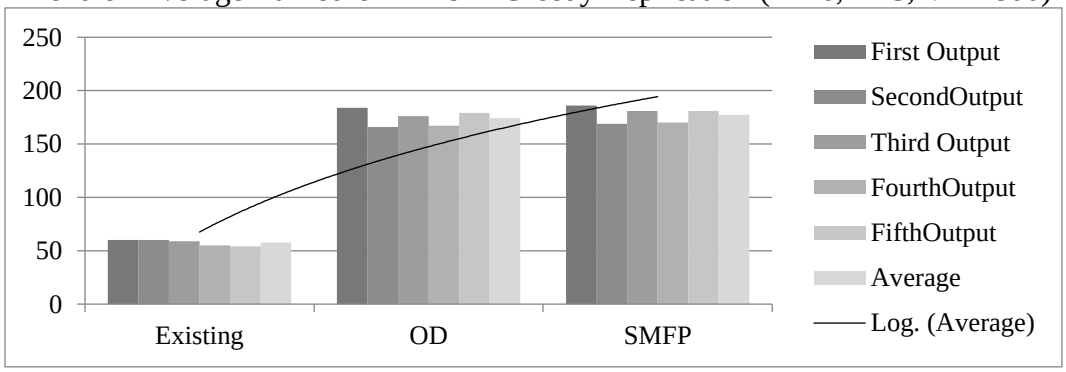


Table 38
Consolidation Results in Greedy Replication (k=16, R=5, VM=900)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	864	55	170	179	5526	685	6211
SecondOutput	874	47	176	181	5294	693	5987
ThirdOutput	861	46	158	160	5522	701	6223
FourthOutput	867	47	164	166	5386	701	6087
FifthOutput	885	59	179	185	5442	700	6142
Average	870	51	169	174	5434	696	6130
PercentImprovement			233.46	2.83			

Chart 38
Trend of Average Turned-off PMs in Greedy Replication (k=16, R=5, VM=900)

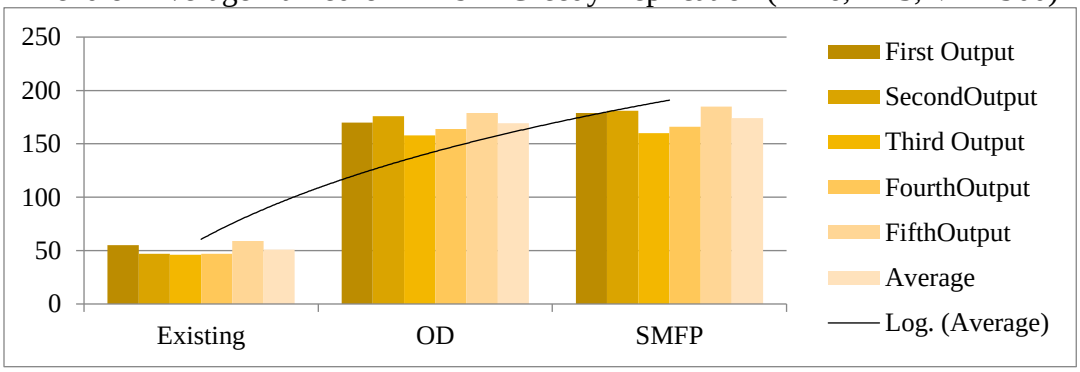


Table 39
Consolidation Results in Greedy Replication (k=16, R=5, VM=950)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	869	41	162	164	6016	705	6721
SecondOutput	867	46	160	163	5984	704	6688
ThirdOutput	888	44	177	182	5830	706	6536
FourthOutput	896	46	172	176	5734	720	6454
FifthOutput	877	43	161	163	6026	714	6740
Average	879	44	166	170	5918	710	6628
PercentImprovement			278.18	1.92			

Chart 39
Trend of Average Turned-off PMs in Greedy Replication (k=16, R=5, VM=950)

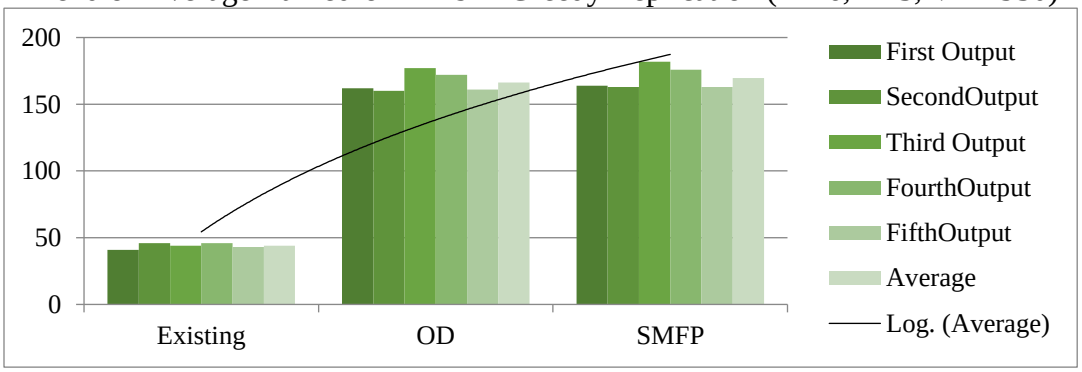
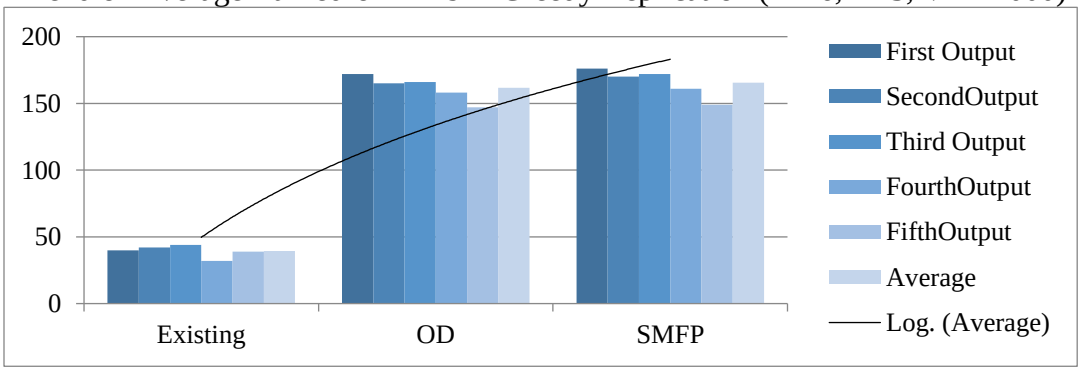


Table 40
Consolidation Results in Greedy Replication (k=16, R=5, VM=1000)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	887	40	172	176	6288	711	6999
SecondOutput	883	42	165	170	6244	713	6957
ThirdOutput	883	44	166	172	6412	711	7123
FourthOutput	886	32	158	161	6216	725	6941
FifthOutput	868	39	147	149	6274	719	6993
Average	881	39	162	166	6287	716	7003
PercentImprovement			310.15	2.48			

Chart 40
Trend of Average Turned-off PMs in Greedy Replication (k=16, R=5, VM=1000)

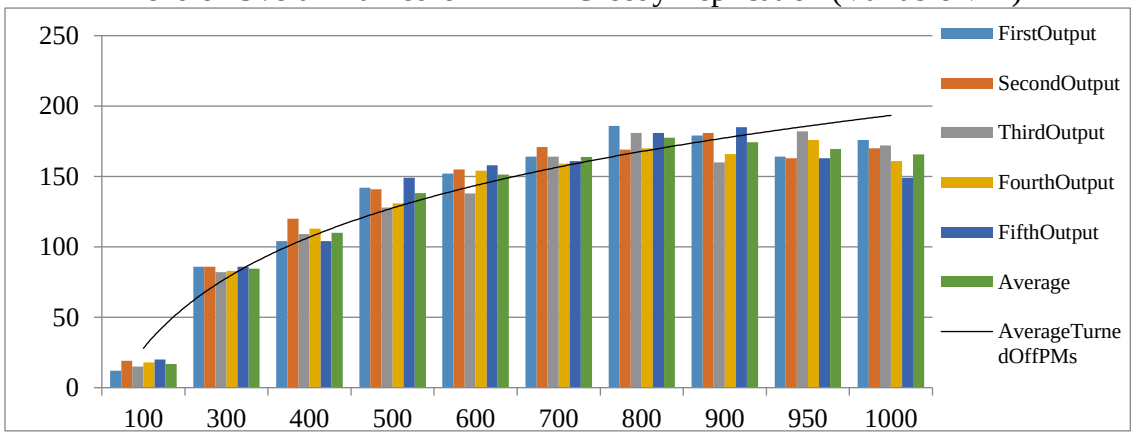


By using the greedy algorithm to scatter original VMs and then running consolidation algorithms, the first algorithm, which is OptimizedDynamic_Consolidation, turns off PMs in an ascending trend, but there is no specific trend in the second algorithm, which is SortedMostFilledPM_Consolidation. Although it increases the number of turned-off PMs in a specific data center, if you compare the percent improvement, it does not follow a certain trend in different data centers. The reason is obvious: the scattering algorithm is not too smart, and the consolidation tuning algorithm does not follow a trend for turning off PMs.

Table 41
Overall Turned-Off PM Trend in Greedy Replication (Variable VM)

VMs	100	300	400	500	600	700	800	900	950	1000
Final Turned-Off PMs	12	86	104	142	152	164	186	179	164	176
	19	86	120	141	155	171	169	181	163	170
	15	82	109	128	138	164	181	160	182	172
	18	83	113	131	154	159	170	166	176	161
	20	86	104	149	158	161	181	185	163	149
Average	17	85	110	138	151	164	177	174	170	166

Chart 41
Trend of Overall Turned-off PM in Greedy Replication (Variable VM)



To make the results clearer, I ran the program for the constant number of VMs and switch ports and changed the number of copies for each virtual machine:

Table 42
Consolidation Results in Greedy Replication (k=16, R=2, VM=950)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	708	39	80	81	1096	627	1723
SecondOutput	706	45	80	81	1112	625	1737
ThirdOutput	673	37	73	74	1174	599	1773
Average	696	40	78	79	1127	617	1744
PercentImprovement			92.56	1.29			

Chart 42
Trend of Average Turned-off PMs in Greedy Replication (k=16, R=2, VM=950)

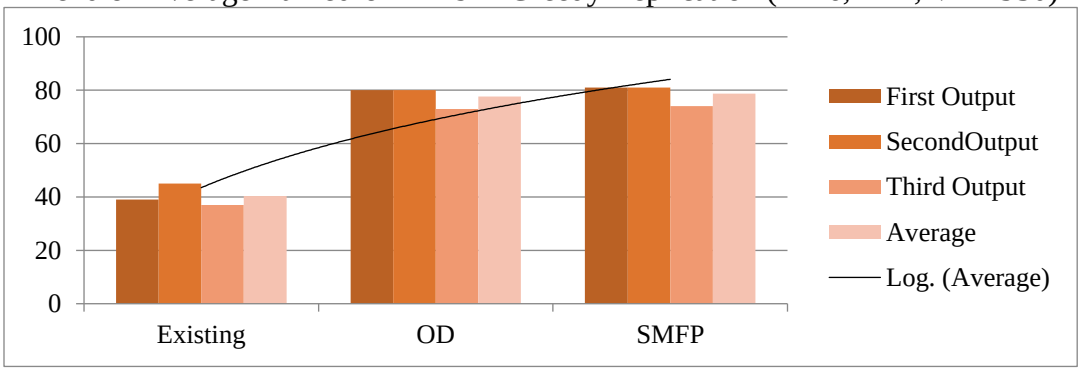


Table 43
Consolidation Results in Greedy Replication (k=16, R=3, VM=950)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	761	58	140	143	2752	618	3370
SecondOutput	765	66	142	148	2676	617	3293
ThirdOutput	763	62	139	140	2680	623	3303
Average	763	62	140	144	2703	619	3322
PercentImprovement			126.34	2.38			

Chart 43
Trend of Average Turned-off PMs in Greedy Replication (k=16, R=3, VM=950)

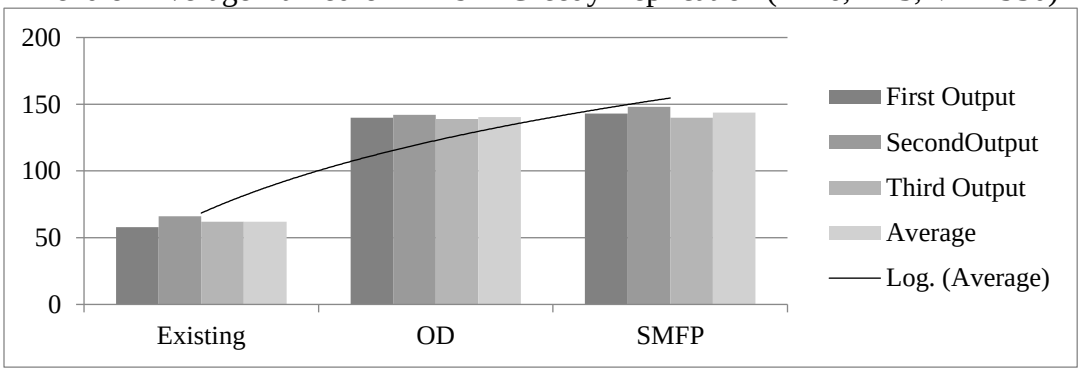
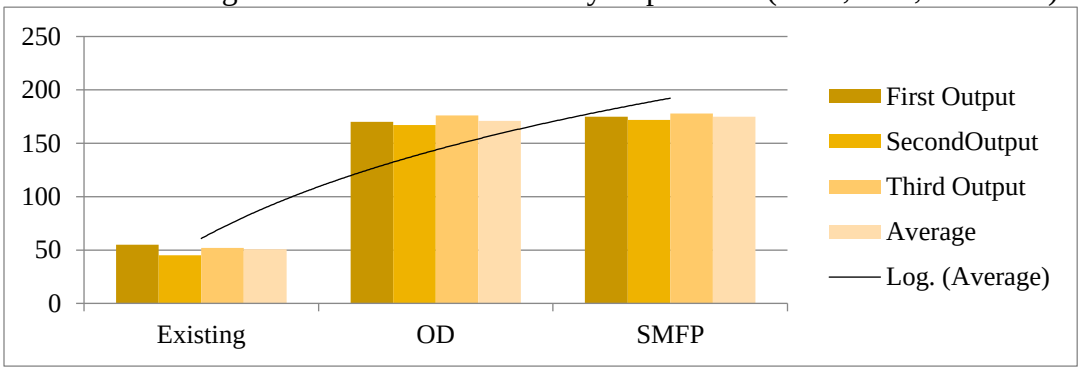


Table 44
Consolidation Results in Greedy Replication (k=16, R=4, VM=950)

	APM	Existing	OD	SMFP	Cost	FNP	FC
FirstOutput	836	55	170	175	4158	661	4819
SecondOutput	836	45	167	172	4182	664	4846
ThirdOutput	829	52	176	178	4140	651	4791
Average	834	51	171	175	4160	659	4819
PercentImprovement			70.37	2.29			

Chart 44
Trend of Average Turned-off PMs in Greedy Replication (k=16, R=4, VM=950)



Output Analysis

Number of Tuned-Off PMs

As you saw in the different tables, the output includes the results of running three replication programs (minimum-cost flow, first-fit, and greedy) to scatter VMs and then running two consolidation programs (OptimizedDynamic_Consolidation and SortedMostFilledPM_Consolidation), which actually involves five proposed consolidation algorithms. In the first series of the output, the number of VMs is changing and the other variables, which are K (switch port) and R (number of copy), are constant. In the next samples, the number of VMs is constant, which is 950, and the number of switch ports is 16, and we change the number of copies from 2 to 4. Results show an increasing number of turned-off PMs after running both consolidation algorithms.

1. By using the MCF algorithm to scatter original VMs in the data center and running the consolidation algorithm, we are able to turn off PMs in an increasing trend by using the first consolidation algorithm, which is OptimizedDynamic_Consolidation. The number of turned-off PMs increases by increasing the number of copies of each virtual machine. But the second algorithm turns off in an increasing order when the data center is not too crowded. As you can see, when the VM number is more than 950, the second algorithm, which is SortedMostFilledPM_Consolidation, cannot add a bigger turned-off PM number to the result. On the other hand, when the data center is almost empty, the number of PMs that can be turned off is not too big, but when we have a large data center with many VMs, the proposed algorithms work better.

2. By using the greedy algorithm to scatter original VMs and then running consolidation algorithms, the first algorithm, which is OptimizedDynamic_Consolidation, turns off PMs in an ascending trend, but there is no specific trend with the second algorithm, which is SortedMostFilledPM_Consolidation. Although it increases the number of turned-off PMs in a specific data center, if you compare the percent improvement in different data centers, it does not follow a certain trend. The reason is obvious: the scattering algorithm is not too smart, and the consolidation tuning algorithm does not follow a trend for turning-off the PMs.

3. By using the first-fit algorithm to scatter original VMs and then running the consolidation algorithm, the trend of turning off PMs by using the first algorithm is again increasing. But the second algorithm does not turn off any PMs. The reason is the first-fit method, which was used in the beginning to scatter original VMs. Since it fills all PMs from the beginning and copies VMs in the first available place, there is no room for the tuning algorithm.

Cost Analysis

In the last column, we calculate the total cost. In the beginning, a cost was calculated for virtual machine replication. After running the consolidation algorithm, some PMs are turned off. As mentioned before, we consider one unit cost for keeping one PM turned on. As a result, the last column, which is the final cost, is calculated for adding the final turned-on PMs to the replication cost. If we consider this column, it is clear that MCF is still the best algorithm for virtual machine replication and consolidation. For example, consider a data center with 800 VMs, 16 switch ports, and 5

copies for each virtual machine. By paying attention to the last column, you see that the average final cost in first-fit is 14504, in greedy 5249, and in MCF 5150. In the following you will see the chart of cost comparison with three assumption of the cost; in the first one we considered the cost of keeping one PM up equal to 1. In the second one it is 100 and in the third one it is 1000.

Table45
Average of Final Cost in First Fit (PM Cost =1)

VMs	100	300	400	500	600	700	800	900	950	1000
Final Cost	2090	5593	7513	9190	10998	12712	14503	16377	17161	18233

Table46
Average of Final Cost in Greedy (PM Cost =1)

VMs	100	300	400	500	600	700	800	900	950	1000
Final Cost	740	1801	2315	2967	3718	4459	5249	6130	6627	7002

Table47
Average of Final Cost in Minimum Cist Flow (PM Cost =1)

VMs	100	300	400	500	600	700	800	900	950	1000
Final Cost	941	1834	2265	2943	3642	4439.	5150	5655	6525	6832

Table48
Average of Final Cost in First Fit (PM Cost =100)

VMs	100	300	400	500	600	700	800	900	950	1000
Final Cost	11965	30814	39638	48097	56588	63178	69696	75074	78951	81494

Table49
Average of Final Cost in Greedy (PM Cost =100)

VMs	100	300	400	500	600	700	800	900	950	1000
Final Cost	34480	57854	63774	65773	67692	70393	72588	75034	76898	77866

Table50
Average of Final Cost in Minimum Cost Flow (PM Cost =100)

VMs	100	300	400	500	600	700	800	900	950	1000
Final Cost	54560	59512	63308	65610	68626	70235	71381	74559	75765	77142

Table51
Average of Final Cost in First Fit (PM Cost =1000)

VMs	100	300	400	500	600	700	800	900	950	1000
Final Cost	101740	260089	331688	401797	471038	521953	571446	608784	637041	656594

Table52
Average of Final Cost in Greedy (PM Cost =1000)

VMs	100	300	400	500	600	700	800	900	950	1000
Final Cost	341200	567434	622494	636733	649272	669793	684768	701434	715718	722086

Table53
Average of Final Cost in Minimum Cost Flow (PM Cost =1000)

VMs	100	300	400	500	600	700	800	900	950	1000
Final Cost	542000	583852	618248	635310	659386	668375	673481	700959	705225	716322

As you see here when the cost for each PM is 1000 the best result which is the minimum cost is generated by First Fit algorithm. But if cost is 1 or 100 still the best algorithm for virtual machine replication and then consolidation is still Minimum cost flow.

CHAPTER 5

CONCLUSION

Virtualized data centers are being used more and more because of the fast growth in cloud service requests. This results in the establishment of large-scale virtualized data centers. In data centers, virtual machines are used to handle the service requests of the user. One problem is failure of a VM. If one needed VM fails, a user's request cannot be completed. To reduce the impact of such failure, replication mechanisms can be a very good solution. The fact is data centers are growing unexpectedly regardless how correctly and efficiently we run the data center. This results in increasing amounts of power consumption.

There are many VM replication algorithms. In this project, we focus on the (1) minimum-cost flow algorithm, (2) first-fit algorithm, and (3) greedy algorithm. These are the most famous algorithms in virtual machine replication.

After all replications are done, all those physical machines that are empty—which means that they are inactive will be turned off. In server consolidation, we plan to create more inactive physical machines from the left active physical machines and turn them off to save energy and have a more efficient data center. The key to this movement is just looking at all PMs one by one and trying to find a new active PM as the target for each VM of that PM. We continue this process until we can move all VMs of a specific PM and turn it off.

In this project, I proposed five algorithms for server consolidation: (1) Dynamic_Consolidation, (2) OptimizedDynamic_Consolidation, (3) Sorted_Consolidation, (4) MostFilledPM_Consolidation, and (5) SortedMostFilledPM_Consolidation.

Optimized_Consolidation includes Dynamic_Consolidation and SortedMostFilledPM_Consolidation, which is a combination of Sorted_Consolidation and MostFilledPM_Consolidation. This algorithm acts like a tuning part for the first one, which is OptimizedDynamic_Consolidation. In all output tables presented in this project, we calculate the number of turned-off PMs in OptimizedDynamic_Consolidation and SortedMostFilledPM_Consolidation.

By using the MCF algorithm to scatter original VMs in the data center and running consolidation algorithms, we are able to turn off PMs in an increasing trend by using the first consolidation algorithm, which is OptimizedDynamic_Consolidation. The number of turned-off PMs increases by increasing the number of copies of each virtual machine. But the second algorithm turns off in an increasing order if the data center is not too crowded. As you can see, when the VM number is more than 950, the second algorithm, which is SortedMostFilledPM_Consolidation, cannot have a bigger turned-off PM number to the result. On the other hand, when the data center is almost empty, the number of PMs that can be turned off is not too big, but when we have a large data center with many VMs, the proposed algorithms work better.

By considering one unit cost for keeping one PM on and calculating the number of active PMs after consolidation, it would be clear that MCF is still the best algorithm for virtual machine replication and consolidation.

When the cost for each PM is 1000 the best result which is the minimum cost is generated by First Fit algorithm. But if cost is 1 or 100 still the best algorithm for virtual machine replication and then consolidation is Minimum cost flow.

CHAPETR 6

FUTURE WORKS

I am going to work on finding a more real number for the cost of keeping one physical machine on to have a more accurate number for the cost of replication and consolidation, which results in the finding of the best algorithm for virtual machine replication in big data centers. Basically the goal is using multi-objective optimization [31] to solve the combined VM replication and server consolidation problem. Multi-objective optimization (also known as multi-objective programming, vector optimization, multi criteria optimization, multi attribute optimization or Pareto optimization) is an area of multiple criteria decision making, that is concerned with mathematical optimization problems involving more than one objective function to be optimized simultaneously

REFRENCES

REFERENCES

[1] Andrew Goldberg's network optimization library. http://www.avglab.com/andrew/soft.html.

[2] Microsoft virtual server. http://www.microsoft.com/windowsserversystem/virtualserver/.

[3] Vmware Inc. http://www.vmware.com.

[4] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu. Energy proportional datacenter networks. *In Proceedings of ISCA, 2010.*

[5] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows: Theory, algorithms, and applications. *Prentice-Hall, Inc., 1993*.

[6] Y. Ajiro and A. Tanaka. Improving packing algorithms for server consolidation. *In Proceedings of the 33rd International Computer.* Measurement Group Conference, 2007.

[7] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev., 38(4):63–74, 2008.*

M. Alicherry and T.V. Lakshman. Optimizing data access latencies in cloud systems by intelligent virtual machine placement. *In Proceedings of IEEE INFOCOM, 2013.*

[9] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *In Proceedings of ACM SOSP, 2003.*

[10] C. Clos. A study of non-blocking switching networks. *Bell System Technical Journal, 32(2):406–424, 1953.*

[11] S. Fang, R. Kanagavelu, B. Lee, C. H. Foh, and K. M. M. Aung. Power-efficient virtual machine placement and migration in data centers. *In Proceedings of the 2013 IEEE*

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

[12] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. *SIGCOMM Comput. Commun. Rev., 41(4):350–361, 2011.*

[13] A. V. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *J. Algorithms, 22:1–29, 1997.*

[14] H. Goudarzi and M. Pedram. Energy-efficient virtual machine replication and placement in a cloud computing system. *In Proceedings of IEEE Cloud, 2012.*

[15] C. Guo, G. Lu, H. J. Wang., S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. Secondnet: A data center network virtualization architecture with bandwidth guarantees. *In Proceedings of CoNEXT, 2010.*

[16] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. Dcell: A scalable and fault-tolerant network structure for data centers. *In Proceedings of the ACM SIGCOMM 2008.*

[17] M. Li, D. Subhraveti, A. R. Butt, A. Khasymski, and P. Sarkar. Cam: A topology aware minimum-cost flow based resource manager for mapreduce applications in the cloud. *In Proceedings of HPDC.*

[18] X. Li, J. Wu, S. Tang, and S. Lu. Let's stay together: Towards traffic aware virtual machine placement in data centers. *In Proceedings of IEEE INFOCOM, 2014.*

[19] F. Machida, M. Kawato, and Y. Maeno. Redundant virtual machine placement for fault-tolerant consolidated server clusters. *In Proceedings of the 12th IEEE/IFIP NOMS 2010, miniconference.*

[20] J. Mudigonda and P. Yalagandula. Spain: Cots data-center Ethernet for multipathing over arbitrary topologies. *In Proceedings of USENIX NSDI, 2010.*

[21] C. H. Papadimitriou and K. Steiglitz. Combinatorial optimization: Algorithms and complexities. *Prentice Hall, 1982.*

[22] B. Tang, N. Jaggi, and M. Takahashi. Achieving data k-availability in intermittently connected sensor networks. *In Proceedings of the IEEE ICCCN, 2014.*

[23] P. Khani, B. Tang, J. Han, and M. Beheshti. Power-efficient virtual machine replication in data centers, *Department of Computer Science, California State University Dominguez Hills, 2009, pp. 1–5.*

[24] Data Center. http://www.centralcolo.com/5-core-elements-to-look-for-in-a-data-center/.

[25] Virtual Machine. http://mobilevirtualmachin.blogspot.com/.

[26] Virtualization Technology. http://www.mat.co.th/en/products/si_consulting/virtual/.

[27] Data Center Power Consumption. http://www.blackmonservice.com/article-display/1167.

[28] First Fit Algorithm. http://slideplayer.com/slide/2328482/.

[29] Greedy Algorithm. https://www.devarticles.com/c/a/Development-Cycles/Greedy-Strategy-as-an-Algorithm-Technique/1/.

[30] Server Consolidation. http://www.lynxnetworks.co.uk/Server%20consolidation.aspx.

[31] Multi Objective Optimization. https://en.wikipedia.org/wiki/Multi-objective_optimization

APPENDIX

SOURCE CODE

First Package:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

package fat.tree;

/**
 *
 * @author Shadi Shiri
 */
import java.io.*;
import java.util.Scanner;

public class OptimizedDynamicConsolidation {

   public static int [] capable = new int [2000];
   public DynamicOptimizedConsolidation(int h,int k){
     capable[h]=k;
   }

     public static void main(String[] args) throws IOException
   {
      PrintWriter myfile = null;
      myfile = new PrintWriter("turnedOFF.txt", "UTF-8");
      FileReader file = new FileReader("textFile.txt");
      FileReader file2 = new FileReader("sample2.txt");
      FileReader file3 = new FileReader("sample3.txt");
      FileReader file4 = new FileReader("sample4.txt");
      int K;
      double A;
      int maxPMsize;
      int minVMsize;
      int VMnumbers;
      int returnedback=0;
      int notreturnedback=0;
      Scanner input2 = new Scanner(file2);
      Scanner input3 = new Scanner(file3);
      Scanner input4 = new Scanner(file4);
      K = input2.nextInt();
      A = input2.nextInt();
      maxPMsize = input2.nextInt();
      minVMsize = input2.nextInt();
      VMnumbers = input2.nextInt();
      int [] FinalMovedPM = new int [(int)A];
      int FinalMovedPMIndex=-1;
      int [] integers = new int [3];
      int [] movedPM = new int [(int)A];
      for ( int i = 0; i < (int)A; i++){
      movedPM[i]=-1;}
```

```
 Integer PM[][] = new Integer[(int)A][(maxPMsize/minVMsize)]; // VMs copies locations
 Integer InitialPM[][]= new Integer[(int)A][(maxPMsize/minVMsize)];// Original VMs location plus
 copies
 Integer FixedInitialPM[][]= new Integer[(int)A][(maxPMsize/minVMsize)];// Original VMs location
 int [] PMsize = new int [(int)A];
 int [] VMsize = new int [VMnumbers];
 for ( int b = 0; b < (int)A; b++){
   FinalMovedPM[b]= -1;

}
 int index ;
 try {
    Scanner input = new Scanner(file);
    while(input.hasNext()){
      int i=0;
      while(i!=3){
         integers[i] = input.nextInt();
         i++;
      }
      if(integers[2]==1){
         index = FatTreeGreedy.search3(PM,((integers[1])-VMnumbers));
         PM[(integers[1])-VMnumbers][index] = integers[0];
      }
    }
    input.close();

    int j =0;
    int index2=0;
    int check;
    while(input2.hasNext()){
    check = input2.nextInt();

      while(check!=1000){
          // here we are entering the original VMs
         FixedInitialPM[j][index2]=InitialPM[j][index2] = check;
         index2++;
         check = input2.nextInt();
      }
    j++;
    index2=0;
     }
     input2.close();

// PMs size
 for ( int i = 0; i < A; i++){
    PMsize[i] = input3.nextInt();
 }
// VMs size
 for ( int i = 0; i < VMnumbers; i++){
    VMsize[i] = input3.nextInt();
 }
  input3.close();
```

```
}
catch(Exception e)
{
   e.printStackTrace();
}
for ( int i = 0; i < (int)A; i++){
   System.out.print("VMs on PM with ID number "+i+": ");
   for ( int j = 0; j < (maxPMsize/minVMsize); j++){
      if(PM[i][j]!= null){
      System.out.print(PM[i][j]+" ");

      }
   }
System.out.println();

}
System.out.println();

 for ( int i = 0; i < (int)A; i++){
   System.out.print("Original VMs on PM with ID number "+i+": ");
   for ( int j = 0; j < (maxPMsize/minVMsize); j++){
      if(FixedInitialPM[i][j]!= null){
      System.out.print(FixedInitialPM[i][j]+" ");

      }
   }
System.out.println();

 }
System.out.println("VMs' size respectively:");
 for ( int i = 0; i < VMnumbers; i++){
   System.out.print(VMsize[i]+" ");
 }
 System.out.println();
  System.out.println("PMs' size respectively:");
 for ( int i = 0; i < A; i++){
   System.out.print(PMsize[i]+" ");
 }
 System.out.println();

 // combining originals and copies VMs arrays
 int vacant;
 for ( int i = 0; i < A; i++){
 vacant=FatTreeGreedy.search3(InitialPM,i);
 int j=0;

  while(j<30 && PM[i][j]!=null){
   if (vacant < 30){
  //  System.out.println("value of i && j && vacant "+i+"      " + j+ "           " + vacant);
InitialPM[i][vacant] = PM[i][j];
vacant++;
```

```
    }
    j++;
 }
}
// Calculating the free space of each PM
System.out.println();
int [] freeSpace = new int [(int)A];
for ( int i = 0; i < (int)A; i++){
    System.out.print("Free space of PM with ID number "+i+": ");
    int assignedSpace = 0;
    for ( int j = 0; j < (maxPMsize/minVMsize); j++){
        if(InitialPM[i][j]!= null){
         assignedSpace =   assignedSpace + VMsize[InitialPM[i][j]];

        }
    }
freeSpace[i] = PMsize[i] - assignedSpace;
System.out.println(freeSpace[i]);

}

System.out.println();
 System.out.print("1111All VMs on PM with ID number : ");
 for ( int i = 0; i < (int)A; i++){


        if(InitialPM[i][0]!= null){
        System.out.println("   "+i+": ");

       }

}

// All VMs on PMs
for ( int i = 0; i < (int)A; i++){
    System.out.print("these are on PMs "+i+": ");
    System.out.print("All VMs on PM with ID number "+i+": ");
    for ( int j = 0; j < (maxPMsize/minVMsize); j++){
        if(InitialPM[i][j]!= null){
        System.out.print(InitialPM[i][j]+" ");

        }
    }
System.out.println();

}

for ( int i = 0; i < (int)A; i++){

        if(InitialPM[i][0]!= null){
          System.out.print("these are on PMs "+i+": ");
```

```
            }

        System.out.println();

        }

       //Getting the cost array from FatTree class
       Integer cost[][] = new Integer[(int)A][(int)A];
       for ( int i = 0; i < A; i++){
           for ( int j = 0; j < A; j++){
              cost[i][j]= input4.nextInt();
           }
        }

       input4.close();

       // check how many active PM we have
       int activePM = 0;
       for ( int i = 0; i < (int)A; i++){
          if(InitialPM[i][0]!= null){
            activePM ++;
          }
       }
       int OriginalVMlocation;
       int tryCost ;// Cost betwwen Original VM and its alone copy on a PM
       int i;
       int totalcostreduced =0;
       int costreduced; // number of VMs that we moved on each PM that had less than 4 VMs
       int x;        int w;
      Integer [][] replaced = new Integer [6][30];
        int VMNumbr;
        int z ;
        int jprim, iprim, wprim, Initialindexprim, Pmindexprim;// variables for status of move PMs
outer: for ( i = 0; i < (int)A ; i++){
            for ( int ss = 0; ss < 6; ss++){
                for ( int pp = 0; pp < 30; pp++){
             replaced[ss][pp]= -1;
            }
        }

          w=0;
          costreduced =0;
          x=0;
          VMNumbr = 0;
          jprim=iprim=wprim=Initialindexprim =Pmindexprim= 0;
          if(FixedInitialPM[i][0]== null && PM[i][0]!= null){ // if we had one or more VM copy on a PM
  firstline:  while(w<30 && PM[i][w]!= null ){

              x++;
               // search to find the location(PM) of original VM of that VM copy
              OriginalVMlocation = FatTreeGreedy.search4(FixedInitialPM,PM[i][w]);
              tryCost = cost[i][OriginalVMlocation];
```

```
 for( int j = 0; j < (int)A; j++){
  if(cost[OriginalVMlocation][j]== tryCost){// finding other PMs with same cost
  //  System.out.println("Selected Physycal machine with the same cost..... "+j);
      if(InitialPM[j][0]!= null){
      //    System.out.println("Selected Physycal which is not off..... "+j);//check if it's not off
        if(VMsize[PM[i][w]] <= freeSpace[j]){                    // Check if there is enough space
       //   System.out.println("Selected Physycal has free space..... "+j);
            if(!(FatTreeGreedy.search2(InitialPM,PM[i][w],j))){
           // Check to make sure we don't already have the selected VM on that PM
           //   System.out.println("Selected Physycal doesnt have this VM..... "+j);
              index = FatTreeGreedy.search3(InitialPM,j);
              if (index < 30){
                 freeSpace[j]= freeSpace[j]-VMsize[PM[i][w]]  ;
                 freeSpace[i]= freeSpace[i]+VMsize[PM[i][w]]  ;
                 replaced[0][VMNumbr]= i;
                 replaced[1][VMNumbr]= w;
                 replaced[2][VMNumbr]= j;
                 replaced[3][VMNumbr]= index;
                 InitialPM[j][index] = PM[i][w];
                 index = FatTreeGreedy.search3(PM,j);
                 replaced[4][VMNumbr]= index;
                 replaced[5][VMNumbr]= PM[i][w];
                 PM[j][index] = PM[i][w];
                 PM[i][w]=InitialPM[i][w] = null;    // Removing from its old location
                 VMNumbr++;
                 costreduced ++;
                 System.out.println("Final Selected Physycal machine for moving is..... "+j);
                 break;
                }
           }

          }
      }
    }
  }
}
w++;
}
if (costreduced==x){
  totalcostreduced++;
  notreturnedback++;
  movedPM[i]=i;
  myfile.println(i);
  FinalMovedPMIndex++;
  FinalMovedPM[FinalMovedPMIndex]=i;
 }
 else{
     returnedback++;
     z=0;
     while(z<30 && replaced[0][z]!=-1){
          iprim = replaced[0][z];
          wprim = replaced[1][z];
          jprim = replaced[2][z];
```

```
                    Initialindexprim = replaced[3][z];
                    Pmindexprim = replaced[4][z];
                    PM[iprim][wprim]=replaced[5][z];
                    InitialPM[iprim][wprim]= replaced[5][z];
                    InitialPM[jprim][Initialindexprim]= null;
                    PM[jprim][Pmindexprim]= null;
                    freeSpace[jprim]++;
                    freeSpace[iprim]--;
                    z++;
                }
            }

            }
        }
       System.out.println();
       for ( i = 0; i < (int)A; i++){
           if(InitialPM[i][0]!= null){
             System.out.println(i) ;
           }
        }
       for ( i = 0; i < (int)A; i++){
           System.out.print("All VMs (after consolidation) on PM with ID number "+i+": ");
           if(InitialPM[i][0]!= null){
             System.out.println("this is value"+InitialPM[i][0]) ;
           }
           System.out.println("capable value "+capable[i]);
           for ( int j = 0; j < (maxPMsize/minVMsize); j++){
              if(InitialPM[i][j]!= null){
              System.out.print(InitialPM[i][j]+" ");

              }
           }
        System.out.println();
       }
        myfile.close();
     System.out.print("Number of active PMs :");
     System.out.println(activePM);
     System.out.print("Number of PMs that we turned off after consolidation :");
     System.out.println(totalcostreduced);
     System.out.println();
    }

}
```

Second Package

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */


package fat.tree;


/**
 *
 * @author Shadi Shiri
 */

import java.io.*;
import java.util.Scanner;
import java.util.Arrays;

public class SortedMostFilledPMConsolidation{

   public static void main(String[] args) throws IOException
   {

      FileReader file = new FileReader("textFile.txt");
      FileReader file2 = new FileReader("sample2.txt");
      FileReader file3 = new FileReader("sample3.txt");
      FileReader file4 = new FileReader("sample4.txt");
      int K;
      double A;
      int maxPMsize;
      int minVMsize;
      int VMnumbers;
      int SelectedTargerPM=-1;
      Scanner input2 = new Scanner(file2);
      Scanner input3 = new Scanner(file3);
      Scanner input4 = new Scanner(file4);
      K = input2.nextInt();
      A = input2.nextInt();
      maxPMsize = input2.nextInt();
      minVMsize = input2.nextInt();
      VMnumbers = input2.nextInt();
      int [] FinalMovedPM = new int [(int)A];
      int FinalMovedPMIndex=-1;
      for ( int b = 0; b < (int)A; b++){
        FinalMovedPM[b]= -1;

     }
//      System.out.print(K + " "+A + " "+ maxPMsize+" "+minVMsize+" "+VMnumbers);
//      System.out.println();
      int [] integers = new int [3];
```

```
int [] movedPM = new int [(int)A];
Integer SortedUsedArray[][] = new Integer[2][(int)A];
for ( int ff = 0; ff < A; ff++){
      SortedUsedArray[0][ff]= ff;
      SortedUsedArray[1][ff]= -1;
      movedPM[ff]=-1;
}
Integer PM[][] = new Integer[(int)A][(maxPMsize/minVMsize)]; // VMs copies locations
// Original VMs location plus copies
Integer InitialPM[][]= new Integer[(int)A][(maxPMsize/minVMsize)];
Integer FixedInitialPM[][]= new Integer[(int)A][(maxPMsize/minVMsize)];// Original VMs location
Integer [][] SelectedPM = new Integer[2][(int)A];
for (int y = 0; y<A; y++){
SelectedPM[0][y]= -1;
SelectedPM[1][y]= -1;
}
int [] PMsize = new int [(int)A];
int [] VMsize = new int [VMnumbers];
Integer SortedPM[] = new Integer[(int)A]; // Sorted VMs copies locations
int index ;
try {
   Scanner input = new Scanner(file);
   while(input.hasNext()){
     int i=0;
     while(i!=3){
        integers[i] = input.nextInt();
        i++;
     }
     if(integers[2]==1){
        index = FatTreeGreedy.search3(PM,((integers[1])-VMnumbers));
        PM[(integers[1])-VMnumbers][index] = integers[0];
     }
   }
   input.close();
   for(int s=0; s<A; s++){

   }
   int j =0;
   int index2=0;
   int check;
   while(input2.hasNext()){
   check = input2.nextInt();
   while(check!=1000){
         // here we are entering the original VMs
        FixedInitialPM[j][index2]=InitialPM[j][index2] = check;
        index2++;
        check = input2.nextInt();
   }
   j++;
   index2=0;
   }
   input2.close();
```

```
    // PMs size
    for ( int i = 0; i < A; i++){
    PMsize[i] = input3.nextInt();
    }
    // VMs size
    for ( int i = 0; i < VMnumbers; i++){
    VMsize[i] = input3.nextInt();
    }
 input3.close();
}
catch(Exception e)
{
    e.printStackTrace();
}
for ( int i = 0; i < (int)A; i++){
    System.out.print("VMs on PM with ID number "+i+": ");
    for ( int j = 0; j < (maxPMsize/minVMsize); j++){
        if(PM[i][j]!= null){
        System.out.print(PM[i][j]+" ");

        }
    }
System.out.println();
}
System.out.println();
for ( int i = 0; i < (int)A; i++){
    System.out.print("Original VMs on PM with ID number "+i+": ");
    for ( int j = 0; j < (maxPMsize/minVMsize); j++){
        if(FixedInitialPM[i][j]!= null){
        System.out.print(FixedInitialPM[i][j]+" ");

        }
    }
System.out.println();

}
System.out.println("VMs' size respectively:");
for ( int i = 0; i < VMnumbers; i++){
    System.out.print(VMsize[i]+" ");
}
System.out.println();
System.out.println("PMs' size respectively:");
for ( int i = 0; i < A; i++){
    System.out.print(PMsize[i]+" ");
}
System.out.println();
// combining originals and copies VMs arrays
int vacant;
for ( int i = 0; i < A; i++){
    vacant=FatTreeGreedy.search3(InitialPM,i);
    int j=0;
    while(j<(maxPMsize/minVMsize)&&PM[i][j]!=null){
```

```
    if (vacant < 30){
      InitialPM [i][vacant] = PM[i][j];
       vacant++;

    }
    j++;
    }
 }
 // Calculating the free space of each PM
 System.out.println();
 int [] freeSpace = new int [(int)A];
 for ( int i = 0; i < (int)A; i++){
    System.out.print("Free space of PM with ID number "+i+": ");
    int assignedSpace = 0;
    for ( int j = 0; j < (maxPMsize/minVMsize); j++){
       if(InitialPM[i][j]!= null){
        assignedSpace =   assignedSpace + VMsize[InitialPM[i][j]];
       }
    }
    freeSpace[i] = PMsize[i] - assignedSpace;
    System.out.println(freeSpace[i]);
 }
 System.out.println();
 System.out.print("1111All VMs on PM with ID number : ");
 for ( int i = 0; i < (int)A; i++){
       if(InitialPM[i][0]!= null){
       System.out.println("   "+i+": ");

       }
 }
 // All VMs on PMs
for ( int i = 0; i < (int)A; i++){
    System.out.print("All VMs on PM with ID number "+i+": ");
    for ( int j = 0; j < (maxPMsize/minVMsize); j++){
       if(InitialPM[i][j]!= null){
       System.out.print(InitialPM[i][j]+" ");

    }
}
 System.out.println();
 }
//Getting the cost array from FatTree class
 Integer cost[][] = new Integer[(int)A][(int)A];
 for ( int i = 0; i < A; i++){
    for ( int j = 0; j < A; j++){
       cost[i][j]= input4.nextInt();
    }
 }
input4.close();
// check how many active PM we have
int activePM = 0;
for ( int i = 0; i < (int)A; i++){
```

```
    if(InitialPM[i][0]!= null){
      activePM ++;
    }
}
// Check where we have just one VM copy
int OriginalVMlocation;
int tryCost ;// Cost betwwen Original VM and its alone copy on a PM
int i;
int totalcostreduced =0;
int costreduced; // number of VMs that we moved on each PM that had less than 4 VMs
int x; // to check how many VMs we had on those PMs with less than 4 VMs
int w;
SortedUsedArray = FatTreeGreedy.SneakySort(PM,(int)A);
for(int ll=0; ll<(int)A; ll++){
    System.out.println("sorted elements are "+SortedUsedArray[0][ll]+": ");
    System.out.println("VM numbers is "+SortedUsedArray[1][ll]+": ");
}
Integer [][] replaced = new Integer [6][30];
Integer [] TargetPM = new Integer [(int)A];
int VMNumbr,TargetNum;
int z,hh ;
int MovedVM=-1;
int workingPM=-1;
int jprim, iprim, wprim, Initialindexprim, Pmindexprim;
int Q;
for ( i = 0; i < (int)A ; i++){
    System.out.println("this is the value of I    "+ i);
    System.out.println("this is the value PM  "+ SortedUsedArray[0][i]+"...");
    System.out.println("this is the value VM#  "+ SortedUsedArray[1][i]);
    for ( int ss = 0; ss < 6; ss++){
      for ( int pp = 0; pp < 30; pp++){
        replaced[ss][pp]= -1;
      }
    }
    for ( int cc = 0; cc < A; cc++){
     TargetPM[cc]=-1;
    }
    Q = SortedUsedArray[0][i];
    TargetNum = 0;
    w=0;
    costreduced =0;
    x=0;
    VMNumbr = 0;
    jprim=iprim=wprim=Initialindexprim =Pmindexprim= -1; // variables for status of moveD VMs
    int E = 0;
    int TargetinSortedaary=-1;
    int SourcetinSortedaary=-1;
    if(FixedInitialPM[Q][0]== null && PM[Q][0]!= null){ // if we had one or more VM copy on a PM
        System.out.println("the first PM we start to work on is      "+Q+"...");
        System.out.println("this pm has       "+SortedUsedArray[1][i]+"VMMMM");
        while(w<30 && PM[Q][w]!= null ){
            System.out.println("this is VM we are working on it"+w);
```

```
      MovedVM=-1;
      E=-1;
      for (int y = 0; y<A; y++){
      SelectedPM[0][y]= -1;
      SelectedPM[1][y]= -1;
      }
      SelectedTargerPM=-1;
      x++;
      OriginalVMlocation = FatTreeGreedy.search4(FixedInitialPM,PM[Q][w]);
// search to find the location(PM) of original VM of that VM copy
 tryCost = cost[Q][OriginalVMlocation];
 for( int j = 0; j < (int)A; j++){
   if(cost[OriginalVMlocation][j]== tryCost){  // finding other PMs with same cost
  //   System.out.println("Selected Physycal machine with the same cost..... "+j);
       if(InitialPM[j][0]!= null){
    //      System.out.println("Selected Physycal which is not off..... "+j);//check if it's not off
            if(VMsize[PM[Q][w]] <= freeSpace[j]){                    // Check if there is enough space
        //  System.out.println("Selected PhyfreeSpace[j]sycal has free space..... "+j);
             if(!(FatTreeGreedy.search2(InitialPM,PM[Q][w],j))){
        // Check to make sure we don't already have the selected VM on that PM
         //   System.out.println("Selected Physycal doesnt have this VM..... "+j);
                index = FatTreeGreedy.search3(InitialPM,j);
                if (index < 30){
                   System.out.println("this is final selected PM"+j);
                   E++;
                   SelectedPM[0][E]= j;
                   SelectedPM[1][E]=FatTreeGreedy.search3(InitialPM, j);
                }
             }
          }
       }
   }
 }
 if (E>-1){
      int QQ = 0;
      int numberodVMS;
      SelectedTargerPM = SelectedPM[0][QQ];
      numberodVMS=SelectedPM[1][QQ];
      while (  QQ < A && SelectedPM[0][QQ]!=-1) {
            if (SelectedPM[1][QQ]>numberodVMS){
               SelectedTargerPM = SelectedPM[0][QQ];
               numberodVMS= SelectedPM[1][QQ];
             }
             QQ++;
       }
       freeSpace[SelectedTargerPM]= freeSpace[SelectedTargerPM]-VMsize[PM[Q][w]];
       freeSpace[Q]= freeSpace[Q]+VMsize[PM[Q][w]];
       index = FatTreeGreedy.search3(InitialPM,SelectedTargerPM);
       replaced[0][VMNumbr]= Q;
       replaced[1][VMNumbr]= w;
       replaced[2][VMNumbr]= SelectedTargerPM;
       replaced[3][VMNumbr]= index;
```

```
        // repalcing the alone VM copy to the new PM of finalized PM
        InitialPM[SelectedTargerPM][index] = PM[Q][w];
        index = FatTreeGreedy.search3(PM,SelectedTargerPM);
        replaced[4][VMNumbr]= index;
        replaced[5][VMNumbr]= PM[Q][w];
        PM[SelectedTargerPM][index] = PM[Q][w];
        PM[Q][w]=InitialPM[Q][w] = null;
        System.out.println("decreasing vm numbers ..... "+Q+" which
        hasssss"+SortedUsedArray[1][i]);
        for( int xx = 0; xx < (int)A; xx++){
             if(SortedUsedArray[0][xx]==Q){
                 workingPM=xx;
                 SortedUsedArray[1][i]=SortedUsedArray[1][i]-1;
                 System.out.println("now it has # vms  ..... "+SortedUsedArray[1][xx]);
                 if (SortedUsedArray[1][xx]<0){
                  System.out.println("This is PM number which is negative ..... "+xx+" which
                  hasssss"+SortedUsedArray[1][xx]);
                 }
                  break;
             }
        }
        for( int xx = 0; xx < (int)A; xx++){
            if(SortedUsedArray[0][xx]==SelectedTargerPM){
                System.out.println("target pm is   ..... "+SortedUsedArray[0][xx]);
                SortedUsedArray[1][xx]=SortedUsedArray[1][xx]+1;
                System.out.println("after moving it has vm #  ..... "+SortedUsedArray[1][xx]);
                if (SortedUsedArray[1][xx]>30){
                }
              break;
            }
        }
        TargetPM[TargetNum]=SelectedTargerPM;
        VMNumbr++;
        costreduced ++;
        TargetNum++;
        break;
   }
   w++;
}
//  System.out.println("the number ov VM of the " + Q + "th physycal machine is "+x);
System.out.println(" costreduced=   "+ costreduced);
System.out.println(" x=   "+ x);
if (costreduced==x){
System.out.println(" finally we could move pm numbwer"+ Q);
totalcostreduced++;
movedPM[i]=Q;
FinalMovedPMIndex++;
FinalMovedPM[FinalMovedPMIndex]=Q;
SortedUsedArray = FatTreeGreedy.SneakySort1(SortedUsedArray,(int)A,i);
if  (SortedUsedArray[0][i]==9){
   for ( int ff = 0; ff < A; ff++){
     if(SortedUsedArray[0][i]==SortedUsedArray[0][ff] && i!=ff)
```

```
            System.out.println("NOw we are opn index#    " +i);
            System.out.println("9 index is#    " +ff);
           }
         }

       }
       else{
          System.out.println("not moved Q valuee is      " +Q);
          for ( int sd = 0; sd < A; sd++){
             if(TargetPM[sd]!=-1){
                System.out.println("which hasss     " +SortedUsedArray[1][TargetPM[sd]]);
              }
           }
           for(hh=0; hh<A;hh++){
              if(TargetPM[hh]!=-1){
                 freeSpace[TargetPM[hh]]++;
                 freeSpace[Q]--;
                 for( int xx = 0; xx < (int)A; xx++){
                    if(SortedUsedArray[0][xx]==Q){
                       SortedUsedArray[1][xx]++;
                     }
                 }
                 for( int xx = 0; xx < (int)A; xx++){
                     if(SortedUsedArray[0][xx]==TargetPM[hh]){
                        SortedUsedArray[1][xx]--;
                     }
                 }
              }
           }
           z=0;
           while(z<30 && replaced[0][z]!=-1){
                     iprim = replaced[0][z];
                     wprim = replaced[1][z];
                     jprim = replaced[2][z];
                     Initialindexprim = replaced[3][z];
                     Pmindexprim = replaced[4][z];
                     PM[iprim][wprim]=replaced[5][z];
              InitialPM[iprim][wprim]= replaced[5][z];
              InitialPM[jprim][Initialindexprim]= null;
              PM[jprim][Pmindexprim]= null;
              z++;
           }
        }
       }
}
System.out.println();
System.out.print("2222All VMs (after consolidation) on PM with ID number ");
for ( i = 0; i < (int)A; i++){
      if(InitialPM[i][0]!= null){
        System.out.println(i);

      }
```

```
 }
 System.out.println();
     for ( i = 0; i < (int)A; i++){
      //    System.out.print("All VMs (after consolidation) on PM with ID number "+i+": ");
         for ( int j = 0; j < (maxPMsize/minVMsize); j++){
            if(InitialPM[i][j]!= null){
              System.out.print(InitialPM[i][j]+" ");
             }
          }
          System.out.println();
      }
      for ( int pp = 0; pp < (int)A; pp++){
         System.out.print("VMs on PM with ID number "+pp+": ");
         for ( int j = 0; j < (maxPMsize/minVMsize); j++){
            if(PM[pp][j]!= null){
            System.out.print(PM[pp][j]+" ");

            }
         }
          System.out.println();

      }
   System.out.print("Number of active PMs :");
   System.out.println(activePM);
   System.out.print("Number of PMs that we turned off after   consolidation :");
   System.out.println(totalcostreduced);

  }

}
```