California State University
DOMINGUEZ HILLS

Profit-Based File Replication in Data Intensive Cloud Data Centers

Project Report by: Muhannad Alghamdi

Dr. Bin Tang
Faculty advisor

Dr. Mohsen Beheshti
Committee Member

Dr. Jianchao Han
Committee Member

Acknowledgement

This report is written based on our paper "Profit-Based File Replication in Data Intensive Cloud Data Centers" submitted to 2017 IEEE International Conference on Communication (ICC). I thank Dr. Bin Tang for the support, guidance, and assistance. Also, Dr. Bin Tang and I thank our committee members Dr. Jianchao Han and Dr .Mohsen Beheshti for their support.

Table of Contents

Abstract

In a cloud data center, many applications require processing of a large amount of data. Nowadays, the amount of energy consumed in a cloud data center became very high, and there are many solutions were proposed to reduce it. One of the problems that needs solutions. One of the solutions to reduce the energy consumption is the file replication strategies. File replication, which brings the data closer to the computing unit, helps minimizing network delays and bandwidth usage. we formulate the file replication problem (FRP) in data center, with the goal of minimizing the total energy consumption of data file access inside data centers. In contrast to all the existing work of data replication in data centers, which are mainly heuristic based, we design a time-efficient approximation algorithm with performance guarantee. The file replication algorithm is based on a novel concept called "profit", and optimizes over a submodular function that can be computed efficiently. We also design two energy- and time-efficient heuristic file replication algorithms. Via extensive simulations using CloudSim, a popular simulation framework for cloud computing, we compare all the algorithms under different network scenarios. We show that the approximation algorithm outperforms the other two under different network parameters, while all three effectively reducing the total energy consumptions of data access in data centers.

Chapter 1

Introduction

1. Introduction

Cloud computing, which provides computing applications, platforms, and infrastructures as services, has emerged as a popular and mainstream technology in today's IT industry. The current cloud data centers, such as Amazon EC2 and Microsoft Azure, support many Internet applications including social networks, video streaming, and search engines. The cloud-based data centers enable individual and business users to easily obtain aforesaid services with pay-as-you-go manner, thus saving the cost of maintaining their own compute infrastructure. All above applications process extreme large amount of data [16]. When users submit jobs to cloud data centers for processing, virtual machines are allocated to execute corresponding application programs, which process the large amount of data. A virtual machine (VM) running on top of physical machine (PM) is an OS environment with its dedicated resources such as CPU cycles, memory, and bandwidth, and is isolated from other parts of the PM. Such isolation enables multiple OS environments on the same PM, allowing that applications previously running on multiple PMs to be consolidated into a single PM. With virtualization, a cloud data center can allocate and utilize its resources more efficiently and provide services to user applications in an effective manner. The execution of the user applications needs that the input data of the application to be available locally for its allocated VM. Therefore how to efficiently locate and access the data for the VMs becomes very important in data centers. Meanwhile, power consumption is still one of the biggest concerns in any data center [14]. Consequently, in cloud data center with thousands of PMs and switches and hundreds of thousands of network links, data file access could consume large amount of energy power in data center. Data replication, which brings data files closer to the computing VMs, is an effective strategy that reduces the data access latencies and bandwidth consumption, thus saving energy in data centers. There have been a few researches that employ data replication techniques to reduce the energy consumption [2], [3], [9], [5], data access delay [2], [3], [12], as well as achieving fault tolerance [8] in data centers. However, almost all of them design heuristic algorithms that do not offer any performance guarantee. Consequently, it is not clearly how performance improvement can be achieved all the time with those heuristic algorithms. In contrast, we design a time efficient approximation algorithm with performance guarantee. We prove that our data replication algorithm reduces the total energy consumption of data access in data center by at least half of that achieved by an optimal replication solution. Based on a novel concept called “profit”, it optimizes over a submodular function that can be computed efficiently. We also design two other energy-efficient heuristic data replication algorithms based on the access patterns of pods and PMs in the data centers. We show that the approximation algorithm outperforms the other two under different network parameters, while all three effectively reducing the total energy consumptions of data access in data centers.

1.2 Related Work

Ping et al. [12] was one of the first that proposed to replicate data across data centers. Their proposed data replica placement algorithm can efficiently achieve near optimal data access delay. The location of replicas for each data object is determined by periodically processing a log of recent data accesses, and by employing a weighted k-means clustering of user locations and deploying replica closer to the centroid of each cluster. Li et al. [8] proposed a replication-based reliability model, which analyzes data storage failures and data loss probability to determine where to create replica copies. Dong et al. [5] proposed replication strategy to minimize power consumption in the backbone network across multiple data centers. They formulated the problem as linear programming and determined optimal points of replication based on the data center traffic demands and popularity of data objects. Boru et al. [2], [3] proposed a data replication technique for cloud computing data centers for joint optimization of energy consumption and bandwidth capacity of data centers as well as inside each datacenter. Lin et al. [9] proposed a replication placement scheme called eStor, under which data was placed in a constrained layout. Some replicas are placed in a sequential way, while other replicas are placed in a random fashion. eStore allows users to configure the replication level and number of replicas, and turn off some nodes without data loss. However, almost all above design heuristic algorithms without any performance guarantee. In current cloud data centers, enormous user data and complex applications call for new replication algorithms. In this paper, we propose a time efficient approximation algorithm with provable performance guarantee. Using a novel concept called “profit”, we prove that our algorithm obtains the profit by at least half of what achieved by an optimal algorithm.

Chapter 2

The Fat-tree topology

2. The Fat-tree topology

In this project, we used the fat-tree network [1] as the cloud data center topology, as it is widely used in data centers to interconnect commodity Ethernet switches. However, the FRP and its algorithms are applicable to any types of data center topologies. A k-ary fat-tree is shown in Fig. 1 with k = 4, where k is the number of ports of each switch. There are three layers of switches: edge switch, aggregation switch and core switch from bottom to top. The lower two layers are separated into k pods. A pod is a modular unit of compute, storage, and networking resources that works as a unit in data center. Each pod contains $\frac{k}{2}$ aggregation switches and $\frac{k}{2}$ edge switches, which form a complete bipartite graph in between. Each edge switch is directly connected to k / 2 physical machines; and each of its remaining k / 2 ports is connected to each of the k / 2 aggregation switches from the same pod. There are $\frac{k^2}{4}$ k-port core switches, each of which is connected to each of k pods. In general, a k-ary fat-tree data center contains $\frac{k^3}{4}$ physical machines. The data center has its own database called Data Center DB, as shown in Fig. 1. The Data Center DB stores all the data files that are needed by the user applications running on this data center. It is connected to all the core switches. This applies to applications such as search engine wherein information is only queried buy users, and is in consistent with the data center layout proposed in [2], [3]. However, our problem formulation and solutions work for a more general scenario, wherein the data files are initially randomly placed on PMs. This applies to applications such as social networking where information is generated by users. Since in both scenarios, the data files are read much more frequently than updated, we assume that data replicas need not be updated.

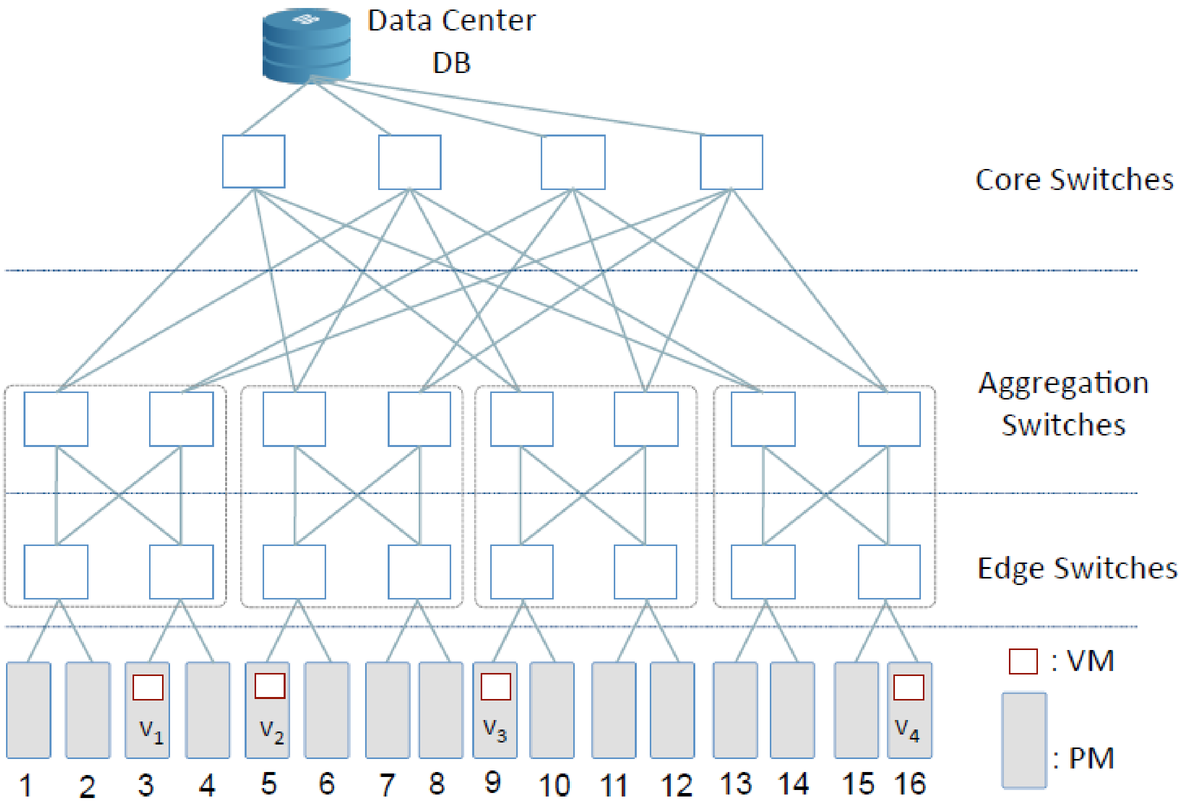


Figure 1 A k-ary fat tree data center with database (Data Center DB). Where K = 4.

Chapter 3

File Replication Problem (FRP)

3. File Replication Problem (FRP) in Data Center

3.1 System Model

We model a cloud data center as a graph G(V;E), where $V = V_p \cup V_s$ includes the set of PMs V_p and the set of (edge, aggregate, and core) switches V_s. Each edge in E connects either one switch to another switch or a switch to a PM. Without loss of generality, let $V_p = \{1, 2, ..., |V_p|\}$, and $Vs = \{|V_p|+1, |V_p|+2, ..., |V|\}$. There are N data files $F = \{f1, f2, f3, ..., f_N\}$ in the data center, where data file f_j ($1 \leq j \leq N$) is originally produced and stored at its source PM $S_j \in V_p$. The size of f_j is s_j . Note that a PM can be the source PM of multiple data files. Let m_i be the storage capacity of PM i. There are n user jobs that are submitted to the cloud data center, and the VMs in PMs are allocated to process these jobs. Suppose that PM i is allocated n_i jobs[1] $\{t_{i1}, t_{i2}, ..., t_{in}\}$. , wherein job t_{ik} ($1 \leq k \leq n_i$). requires some of the data files $F_{ik} \subseteq F$ as input files for execution. Let a_{ij} be the number of times that PM i needs to access data file f_j to execute all its n_i jobs. aij is also referred to as the **request frequency** of PM i to file f_j . A file with a larger request frequency therefore needs to be brought closer to the PMs that need them the most.

3.2 Energy Model

The measure the power consumption of one time access of data file f_j from PM **i** is the minimum number of switches existing between PM i and S_j , the source PM of f_j . This is in accordance to the findings made by Meng et al. [10], which observes that the energy consumption of communication inside data center is proportional to the number of switches the communication traverses. However, our problem and algorithm can be easily adjusted to accommodate he scenario that different switches consumes different amount of energy (for example, high-end core switches consume more power than aggregation and edge switches.).Let e_{ij} denote the energy consumption between any two nodes (switches or PMs) $i \in V$ and $j \in V$. First, we calculate the total energy consumption in the data center to execute all the jobs without any data replication, which is the sum of energy consumption of each PM accessing each data file from its source PM. Denote it as $\mathcal{E}_{init}$, we have:

$$\mathcal{E}_{init} = \sum_{i=1}^{|V_P|} \sum_{j=1}^{l} a_{ij} \cdot e_{iS_j}. \qquad (1)$$

3.3 Problem Formulation

The objective of the FRP is to minimize the total energy consumption of data access in the data center by replicating data files into different PMs while satisfying the storage capacity of each PM. Let's give the following definitions and notations.

3.3.1 File Sets and Set of File Sets

Define file set of a PM as the set of data files that this PM stores (including the initial files it stores as a source PM). For PM i, let $F_i \subseteq F$ denote its file set, and let:

$$s(F_i) = \sum_{f_j \in F_i} s(f_j)$$

denote the total size of data files in F_i. Let the sets of the file sets be represneted as:

$$\mathcal{F} = \{F_1, F_2, ..., F_{|V_p|}\}$$

Initially, F_i is the set of files that have PM i as source PMs. That is,

$$F_i = \bigcup_{1<j<l} x_i,\ 1 \leq i \leq |V_p|$$

where:

$$x_i = \begin{cases} \{f_j\} & \text{if } (i == S_j), \\ \phi \text{ (empty set)} & \text{otherwise.} \end{cases}$$

We denote the above initial file set of each PM and the set of file sets as F_i^{init} ($1 \leq i \leq |V_p|$) and

$F_{init} = \{F_1^{init}, F_2^{init}, \ldots, F_{|V_p|}^{init}\}$.

3.3.2 Energy Consumption of Data Access in Data Center

With replication, multiple copies of the same data file can exist in the data center. For energy saving, each PM accesses the copy that incurs the smallest amount of energy. Given any F and any PM i, we refer to the PM that stores a copy of f_j that i can access f_j with smallest amount of energy as i's access PM for f_j , and denote it as $A_{ij}(F)$. That is,

$$A_{ij}(\mathcal{F}) = \arg\max_k(e_{ik}, \text{where } j \in F_k).$$

Given any $\mathcal{F} = \{F_1, F_2, ..., F_{|V_p|}\}$,the minimum energy consumption of data access in data center is therefore

$$\mathcal{E}(\mathcal{F}) = \sum_{i=1}^{|V_p|} \sum_{j=1}^{l} a_{ij} \cdot e_{iA_{ij}(\mathcal{F})}. \qquad (2)$$

Note that equation 1 from section (3.1) can be represented as $\mathcal{E}(\mathcal{F}^{init})$.

3.4 Objective of FRP

The objective of FRP is to select a set of $|V_p|$ file sets $\mathcal{F} = \{F_1, F_2, ..., F_{|V_p|}\}$, such that the *minimum total energy* consumption of data access in data center. It also can be represented as:

$$\mathcal{E}_{min} = \min_{\mathcal{F}} \mathcal{E}(\mathcal{F}) \qquad (3)$$

The FRP is NP-hard [7], [13]. Below we design time efficient approximation algorithm as well as heuristic algorithms to solve it.

Chapter 4

Algorithms for FRP

4. Algorithms for FRP

4.1 An Approximation Algorithm

Our approximation algorithm delivers a solution whose total energy consumption reduction is at least one half of the optimal total energy consumption. We first give below definition.

Definition 1: (Profit of Replicating file f_j at PM i under F): The profit of replicating file f_j at PM i under $\mathcal{F} = \{F_1, F_2, ..., F_{|V_p|}\}$, denoted as $\Delta\mathcal{E}(\mathcal{F}, f_j, i)$, is the reduction of total energy cost in the data center when placing a copy of f_j at PM i divided by $s(f_j)$, given that the current set of file sets is F. Let F' =

$$\{F_1, F_2, ..., F_{i-1}, F_i \cup \{f_j\}, F_{i+1}, ..., F_{|V_p|}\}$$

then we can have,

$$\Delta\mathcal{E}(\mathcal{F}, f_j, i) = \left(\mathcal{E}(\mathcal{F}) - \mathcal{E}(\mathcal{F}^{'})\right)/s(f_j).$$

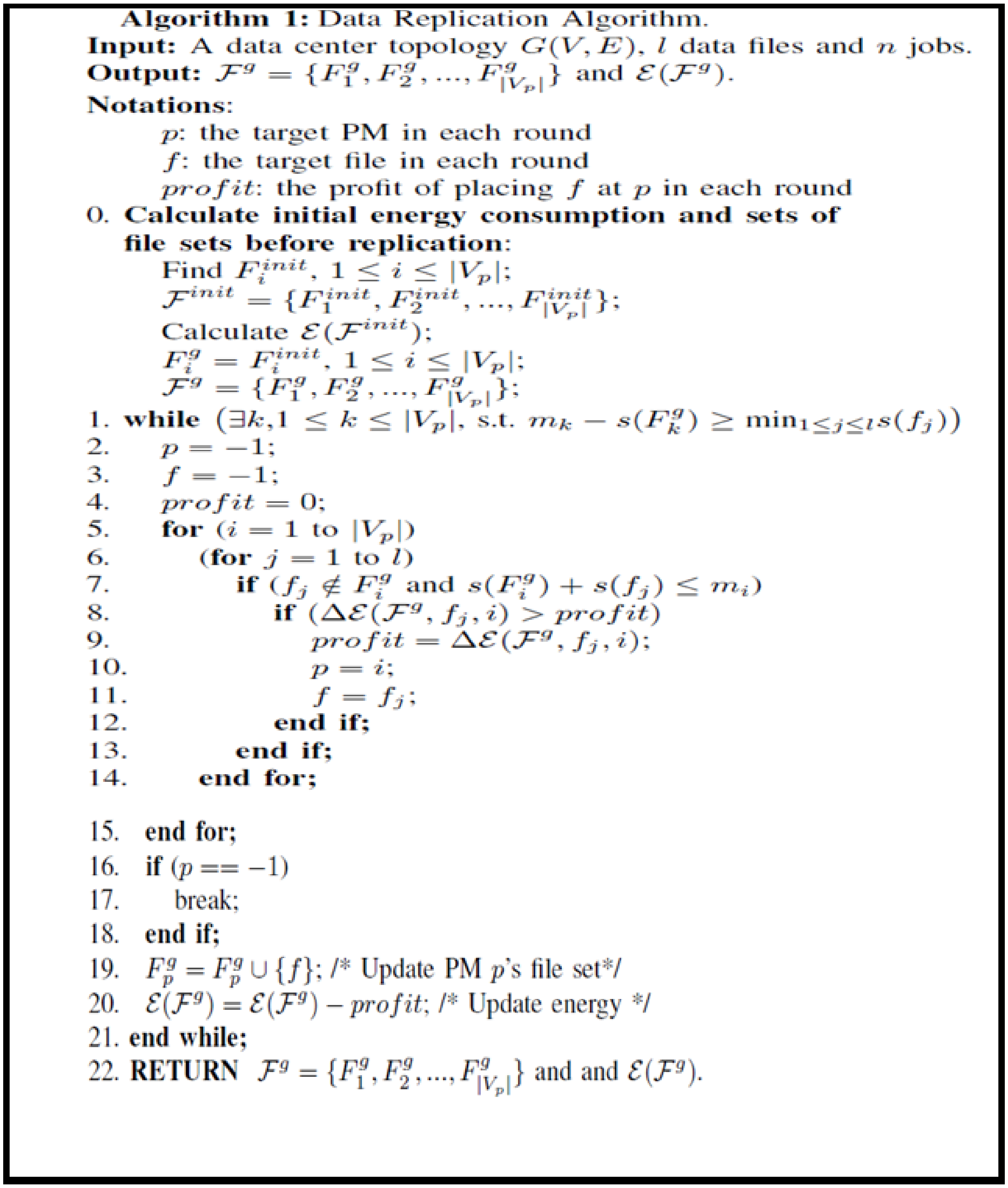
Obviously, in above definition, if $f_j \in F_i$, i.e., a copy of f_j is already located at PM i, then $\Delta\mathcal{E}(\mathcal{F}, f_j, i)$ = 0. The intuition behind the "profit" is that replicating a file into a PM is more profitable if this reduces more energy consumption of file access in the data center as well as the file has a smaller size (so that less storage space of a PM it occupies). We therefore should choose a file-PM pair for replication that achieves the maximum reduction of energy consumption while costing least amount of storage space for the replicated file. Algorithm 1 (Figure 2) below is a "profit"-based greedy algorithm that takes place in rounds. In each round, it decides that by replicating which file at which PM, it can reduce the total energy of data access the most. Here we refer to such a *file* and *PM* in that round as *target file* and *target PM*, respectively. This continues until either there is no storage space available at any PMs for file replication, or it can no longer reduce the total energy by replication (Line1). Let's denote the set of file sets produced by Algorithm 1 as:

$$\mathcal{F}^g = \{F_1^g, F_2^g, ..., F_{|V_p|}^g\}.$$

4.1.1 Time Complexity of algorithm 1

The initialization stage (Line 0 in Figure 2) takes $O(|V_p|^3 + |V_p| \cdot l)$, as finding minimum energy consumption between any two PMs takes $O(|V_p|^3)$, and calculating the total energy consumption without replication (Equation 1 in section 3.1) takes $|V_p| \cdot l$. The while loop (Line 1) takes about $\frac{\sum_{1 \leq i \leq |V_p|} m_i}{\sum_{1<j<l} s(f_j)/l}$ rounds, which can be upper-bounded by $|V_p| \cdot m'$ with m' being the average storage capacity of a PM. Each round takes at most $(|V_p|^2 \cdot l)$, since it iterates over all PM-file pairs to decide which file is replicated into which PM, and it takes $O(|V_p|)$ to calculate $\Delta\mathcal{E}$. Therefore, the time complexity of Algorithm 1(the profit algorithm is:

$$O(|V_p|^3 + |V_p| \cdot l + |V_p| \cdot \bar{m} \cdot |V_p|^2 \cdot l) = O(|V_p|^3 \cdot \bar{m} \cdot l).$$



```
Algorithm 1: Data Replication Algorithm.
Input: A data center topology $G(V, E)$, $l$ data files and $n$ jobs.
Output: $\mathcal{F}^g = \{F_1^g, F_2^g, \ldots, F_{|V_p|}^g\}$ and $\mathcal{E}(\mathcal{F}^g)$.
Notations:
    $p$: the target PM in each round
    $f$: the target file in each round
    $profit$: the profit of placing $f$ at $p$ in each round
0. Calculate initial energy consumption and sets of
   file sets before replication:
    Find $F_i^{init}$, $1 \leq i \leq |V_p|$;
    $\mathcal{F}^{init} = \{F_1^{init}, F_2^{init}, \ldots, F_{|V_p|}^{init}\}$;
    Calculate $\mathcal{E}(\mathcal{F}^{init})$;
    $F_i^g = F_i^{init}$, $1 \leq i \leq |V_p|$;
    $\mathcal{F}^g = \{F_1^g, F_2^g, \ldots, F_{|V_p|}^g\}$;
1. while $(\exists k, 1 \leq k \leq |V_p|$, s.t. $m_k - s(F_k^g) \geq \min_{1 \leq j \leq l} s(f_j))$
2.     $p = -1$;
3.     $f = -1$;
4.     $profit = 0$;
5.     for $(i = 1$ to $|V_p|)$
6.         (for $j = 1$ to $l$)
7.             if $(f_j \notin F_i^g$ and $s(F_i^g) + s(f_j) \leq m_i)$
8.                 if $(\Delta\mathcal{E}(\mathcal{F}^g, f_j, i) > profit)$
9.                     $profit = \Delta\mathcal{E}(\mathcal{F}^g, f_j, i)$;
10.                    $p = i$;
11.                    $f = f_j$;
12.                end if;
13.            end if;
14.        end for;
15.    end for;
16.    if $(p == -1)$
17.        break;
18.    end if;
19.    $F_p^g = F_p^g \cup \{f\}$; /* Update PM $p$'s file set*/
20.    $\mathcal{E}(\mathcal{F}^g) = \mathcal{E}(\mathcal{F}^g) - profit$; /* Update energy */
21. end while;
22. RETURN $\mathcal{F}^g = \{F_1^g, F_2^g, \ldots, F_{|V_p|}^g\}$ and and $\mathcal{E}(\mathcal{F}^g)$.
```

Figure 2 the Profit Algorithm.

4.1.2 Submodularity

A set function $\phi : 2^U \rightarrow N$ is called submodular if for every A ⊆ B ⊆ U and $e \in U - B$ it holds that:

$$\phi(A \cup \{e\}) - \phi(A) \geq \phi(B \cup \{e\}) - \phi(B).$$

Next we prove that $\mathcal{E}(\mathcal{F})$ is submodular when all the files have the same unit size.

4.1.3 Profit Algorithm Thermos and proofs

Theorem 1: $\mathcal{E}(\mathcal{F})$ is submodular when $s_{j_1} = s_{j_2} = 1 \; \forall \; 1 \leq j_1 \neq j_2 \leq l.$

Proof: In each round of Algorithm 1, it selects a data file, say f_j, and places a copy of it into the storage of PM i. It is equivalent to say that a variable D_{ijk} is selected in this round, where $1 \leq i \leq |V_p|$, $1 \leq j \leq l$, and $1 \leq k \leq m_i$. Dijk indicates that fj is placed in the kth storage slot of PM i. Algorithm 1 essentially selects a sequence of such variables. Then we can rewrite $\mathcal{E}(\mathcal{F})$ as $\mathcal{E}(A)$, where A is the set of variables selected so far. Next we show that $\mathcal{E}(A)$ is

submodular. Let U be the entire set of variables selected after the algorithm, and let A ⊆ B ⊆ U. Let $D_{ijk} \in U - B$. Since $\mathcal{E}(\mathcal{F})$ is a minimization function, we need to show that:

$$\mathcal{E}(A) - \mathcal{E}(A \cup \{D_{ijk}\}) \geq \mathcal{E}(B) - \mathcal{E}(B \cup \{D_{ijk}\}).$$

Let $\mathcal{E}_j(A)$ denote the total energy consumption accessing f_j after A is selected. Since D_{ijk} can only possibly affect the energy consumption accessing f_j, we only need to show that

$$\mathcal{E}_j(A) - \mathcal{E}_j(A \cup \{D_{ijk}\}) \geq \mathcal{E}_j(B) - \mathcal{E}_j(B \cup \{D_{ijk}\}).$$

This is indeed true since in each round of Algorithm 1, it finds the PM-file pair that reduces the access energy consumption the most.

Next we show that Algorithm 1 delivers a solution whose total energy cost reduction is at least one half of the optimal total access cost reduction. The proof technique used below is similar to that used in [11] for a closely related problem of data replication in data grid scientific applications.

Theorem 2: Given any instance of FRP, let $\mathcal{E}_{init}$ be the total energy consumption of data access without replication, $\mathcal{E}_{min}$ be the optimal total energy consumption of data access with replication, and $\mathcal{E}_g$ be the total energy consumption of data access given by Algorithm 1. We have:

$$\frac{\mathcal{E}_{init} - \mathcal{E}_g}{\mathcal{E}_{init} - \mathcal{E}_{min}} > \frac{1}{2}$$

when all the files have the same unit size.

Proof: Let L is the total number of rounds in Algorithm1. And let the sequence of selections in Algorithm 1 is $\{n_1^g f_1^g, n_2^g f_2^g, ..., n_L^g f_L^g\}$, with $n_i^g f_i^g$ indicating that at round i, data file f_i^g is replicated at PM n_i^g . Let the optimal sequence of selections be $\{n_1^o f_1^o, n_2^o f_2^o, ..., n_L^o f_L^o\}$, with $n_i^o f_i^o$ indicating that at round i, data file f_i^o is replicated at site n_i^o. Let $O = \mathcal{E}_{init} - \mathcal{E}_{min}$ and $C = \mathcal{E}_{init} - \mathcal{E}_g$ be the profit from optimal algorithm and Algorithm 1 respectively. Consider a new data center graph G', where the storage capacity of each PM i is changed from m_i to $2m_i$. For each PM i, let its first m_i storage slots store the data files obtained in Algorithm 1, and its second mi storage slots store the data files selected in optimal algorithm. Now we calculate the profit O' for G'. O' ≥ O, because each site in G0 stores extra data files beyond the data files stored in the same PM in G.

Let the sequence of selections in G' be $\{n_1^g f_1^g, n_2^g f_2^g, ..., n_L^g f_L^g, n_1^o f_1^o, n_2^o f_2^o, ..., n_L^o f_L^o\}$

The profit after the first L selections is C. For the second L selections, we need to calculate the profit when adding $n_i^o f_i^o$ on $\{n_1^g f_1^g, n_2^g f_2^g, \ldots, n_{i-1}^g f_{i-1}^g\}$. Thus, the sum of the profits due to selection of $\{n_1^o f_1^o, n_2^o f_2^o, ..., n_L^o f_L^o\}$ is less than or equal to C too.

Therefore, O is less than or equal to O', which is less than or equal to two times of C.

4.2 Heuristic Algorithms

We further propose two other time-efficient heuristic file replication algorithms, and compare them with the approximation algorithm via simulations.

4.2.1 Local Greedy Algorithm

In Local Greedy, it replicates each PM's most frequently requested data files in its local storage. That is, for PM i with mi storage capacity, it places the mi data files (out of the l files) that have the highest request frequencies by PM i. Using a heap, finding the top m_i files from l files take $O(l + m_i \cdot \log l)$. Therefore, it takes $O(|V_p| \cdot (l + \bar{m} \cdot \log l))$ for all the $|V_p|$ PMs, where $\overline{m}$ is the average storage capacity of a PM. After this replication, calculating the total energy cost is $|V_p| \cdot l \cdot |V_p| \cdot \overline{m}$ since for each PM-file pair, finding a copy of this file that is closest to the PM takes $|V_p| \cdot \overline{m}$ time. Therefore, the time complexity for the Local Greedy is $O(|V_p|^2 \cdot \overline{m} \cdot l)$.

4.2.2 Pod-Based Greedy Algorithm

In this algorithm, it first finds the aggregate request frequency of each file in each pod (i.e., the sum of the request frequencies of all the PMs in this pod for that data file). Then in each pod, it replicates the data files with the highest aggregate request frequency that are allowed by the total storage capacity of that pod. Specifically, we start with the file with the highest aggregate frequency, and place a copy of it to the PM that has the highest request frequency to it. If this PM is full, it tries the one with the second highest request frequency, etc. This finishes until all those data files are placed into the pod. Finding the aggregate request frequencies take $O(|V_p| \cdot l)$, placing replica copies of data files into all the pods takes $O(|V_p| \cdot l)$, and calculating the total energy cost is $|V_p| \cdot l \cdot |V_p| \cdot \overline{m}$. Therefore, the time complexity for Pod-Based algorithm is $O(|V_p|^2 \cdot \overline{m} \cdot l)$.

Chapter 5

Simulation and Performance Evaluation

5. Simulation and Performance Evaluation

5.1 Simulation Setting

In this section, we compare the performances of the three file replication algorithms. We refer to our approximation algorithm as Profit, the pod-based greedy algorithm as Pod, and the local greedy algorithm as Local. We generate fat-tree data centers of different sizes: k = 8, a small data center with 128 PMs; and k = 16, a large data center with 1024 PMs. The size of each data file and its replica copies is 2 GB. The storage capacity of each PM is varied from 100GB to 500GB. There are 1000 data files that are either located in the central database of the cloud data center (referred to as Central DB), or are randomly placed on the PMs (referred to as Random Placement).

5.2 Energy Consumption Models

We use r_e, r_a, and r_c to denote the power consumption of transmitting one data file copy on the edge, aggregate, and core switches respectively. We consider two energy consumption models that are currently adopted in cloud data center research:

- In uniform energy model, the energy consumption of data access is measured as number of switches the data traverses [10]. We set $r_e = r_a = r_c = 1$.
- In skewed energy model, the core switches handle huge amount of traffic across the

entire data center, therefore consuming more energy power than aggregate switches, which consume more energy power than edge switches. We set $r_e = 1$, $r_a = 5$, and $r_c = 10$.

5.3 Data File Access Pattern

We adopt two data file access patterns to characterize the request frequencies of data files.

- In Zipf distribution, the request frequency to access the $j^{th} (1 \leq j \leq l)$ popular data file is represented by $p_j = \frac{1}{j^{\theta} \sum_{k=1}^{l} \frac{1}{k^{\theta}}}$ We choose θ to be 0.6 based on the real trace studies collected at Facebook data center [6], [15].
- In random access, the request frequency of each file by each PM is a random number between 0 and 100.

5.4 Performance Comparison Under Uniform and Skewed Energy Models

Fig. 3 and Fig. 4 show the total energy consumption of the three algorithms by varying the storage capacity of each PM, under uniform and skewed energy models, respectively. It shows that all three replication algorithms effectively reduce the total energy consumption of file access in the data center. Profit outperforms Local and Pod in the entire parameter range under both energy models. We also observe that all three algorithms perform better under skewed energy model than under uniform energy model by reducing more energy consumptions. This is because in skewed energy model, core switches cost more energy than aggregation and edge switches. By storing the replica copies at local PMs, access traffic does not go through core switches often, therefore reducing energy consumption more in skewed energy model than in uniform energy model. Finally, we observe that under each energy model, the energy consumption by all three algorithms decrease with the increase of storage capacity in most cases, except for Pod when storage capacity exceeds 200 GB. Under Pod, each pod continues storing only one copy of each data file with the increasing of storage, therefore keeping the energy consumption the same.

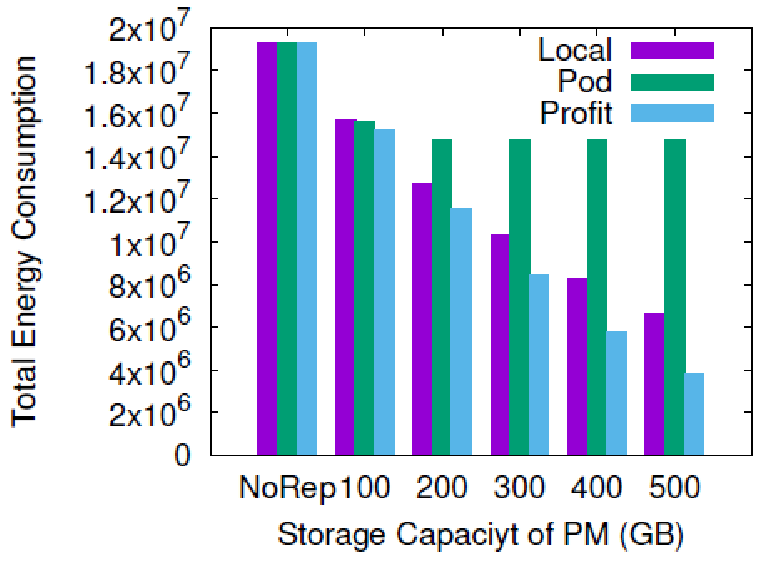


Fig. 3. Performance comparison for data center with 128 PMs under uniform energy model. "NoRep" indicates the total energy consunmption without any replication. Here, we adopt random access pattern and central DB.

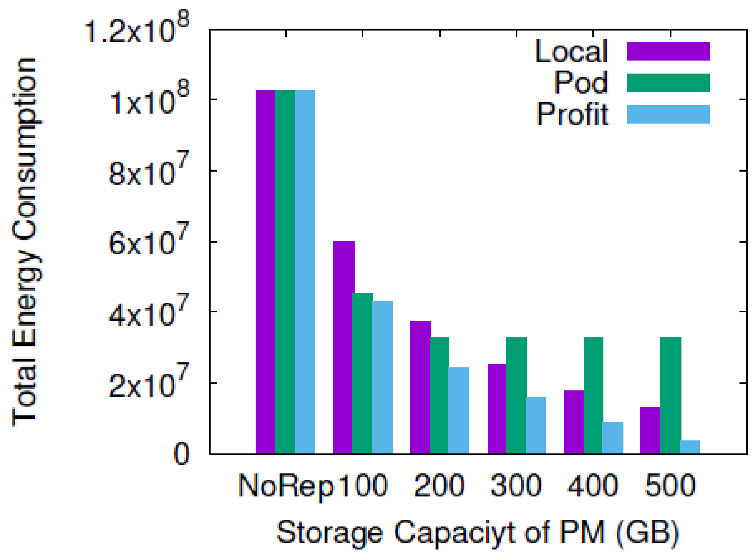
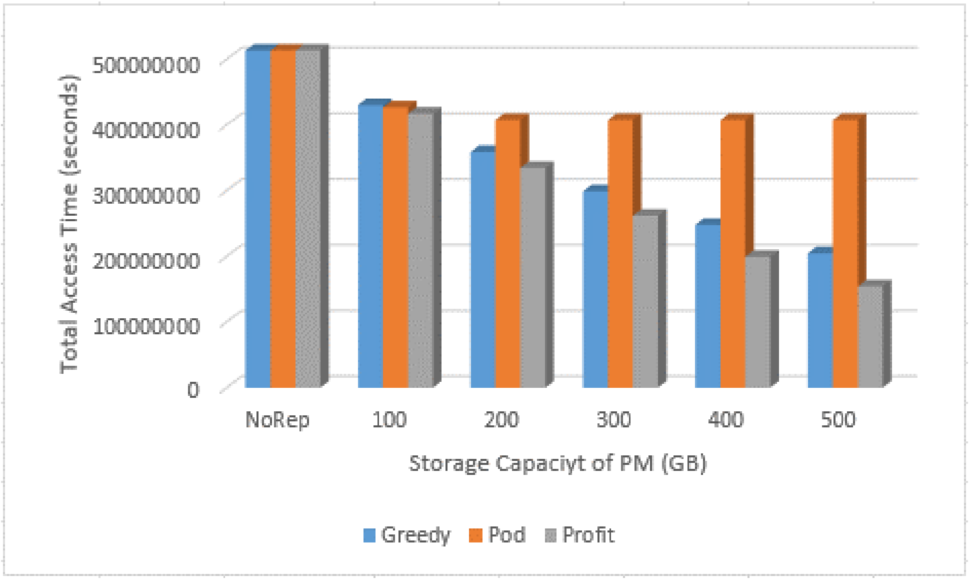


Fig. 4 . Performance comparison for data center with 128 PMs under skewed energy model. Here, we adopt random access pattern and central DB.

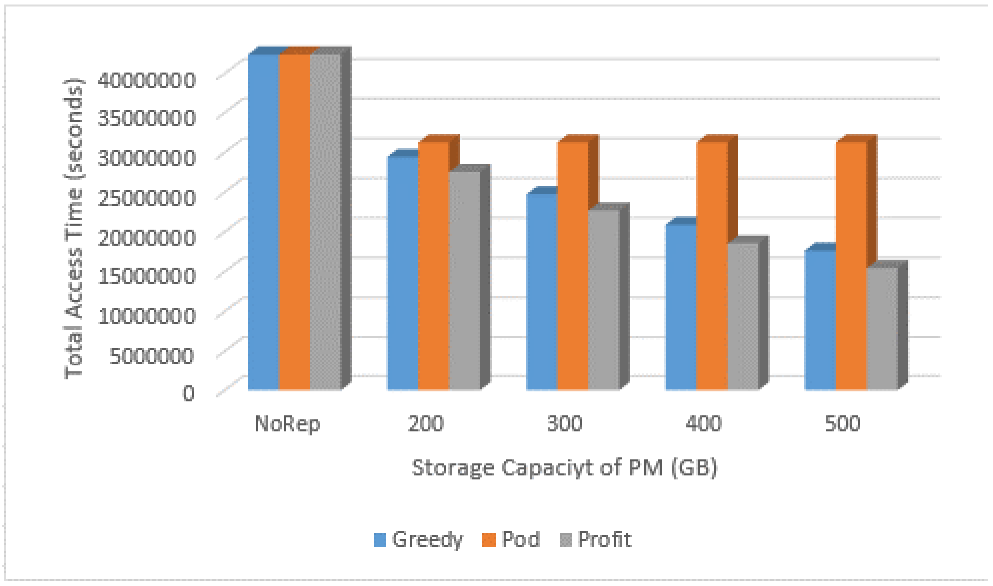
5.5 Performance Comparison Under CloudSim

CloudSim is one of the most popular open source cloud simulators in the research and academia. We set the link bandwidth as 100MB/s in CloudSim, and measure the total access time of data files yielded by the three algorithms, as shown in Fig 5. We observe that Profit performs better than Local, which outperforms Pod. In particular, when the storage is large (500 GB), Profit can reduce the total access time of the data center by roughly half via replication. Note that under CloudSim, both uniform and skewed energy models perform the same, since the access time only depends on file sizes and link bandwidth, which is fixed in this case.



5.5.1 Random Placement of Data Files

All simulations so far assume the availability of a central DB, and all the data files are initially stored in this central DB. Next we study the effects of the random initial placement of data files. We set the bandwidth of the links under edge switches as 1GB, under aggregation switches as 2GB, and under core switches as 5GB. Fig. 6 shows that the total access cost of different algorithms. The performance comparison of the three algorithms stay the same as in the central DB. However, the costs are much smaller than those in Fig. 5, since the links have much higher bandwidth.



5.5.2 Study of Scalability

We study the performances of the three algorithms in larger data center of 1024 PMs, in order to understand their scalability. All the set up is the same as in Fig. 6, except for the size of the data center. We compare Profit and Local, since both outperforms Pod. Table II shows the total energy consumptions of both algorithms in a small 128-PM data center and a large 1024 PMs. We set the storage capacity of each PM as 300GB, the medium in the storage parameter range. The last column, Improvement Percentage, is calculated as the energy consumption difference between Profit and Local divided by energy consumption of Local. It shows that in small data center, Profit improves upon Greedy by 6:69% while in large data center, it is 13:78% improvement. This shows that our approximation algorithm performs better than Local in large data centers therefore is more scalable.

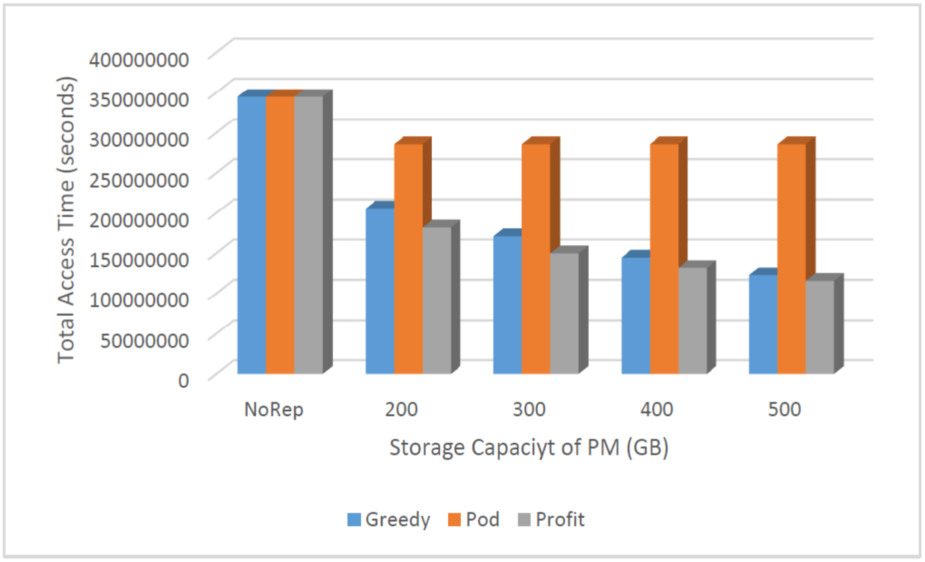


Fig. 7. Performance comparison for data center with 1024 PMs, with random initial placement.

TABLE II
STUDY OF SCALABILITY. STORAGE CAPACITY = 300GB.

Data Center Size	Optimal	Greedy	Improvement Percentage (%)
128	537600	576183.99	6.69
1024	3481600	4038467.19	13.78

5.5.3 Zipf Distribution

We study the performance of the proposed algorithms under Zipf distribution access pattern. Fig. 8 shows that under Zipf distribution, the performance difference between Profit and Local is even larger. This shows that Profit works particularly well for data files with distinct popularity levels. When data files have distinct popularity levels, the popular data is always be replicated to more PMs according to our approximation algorithm, therefore reducing the energy consumption the most.

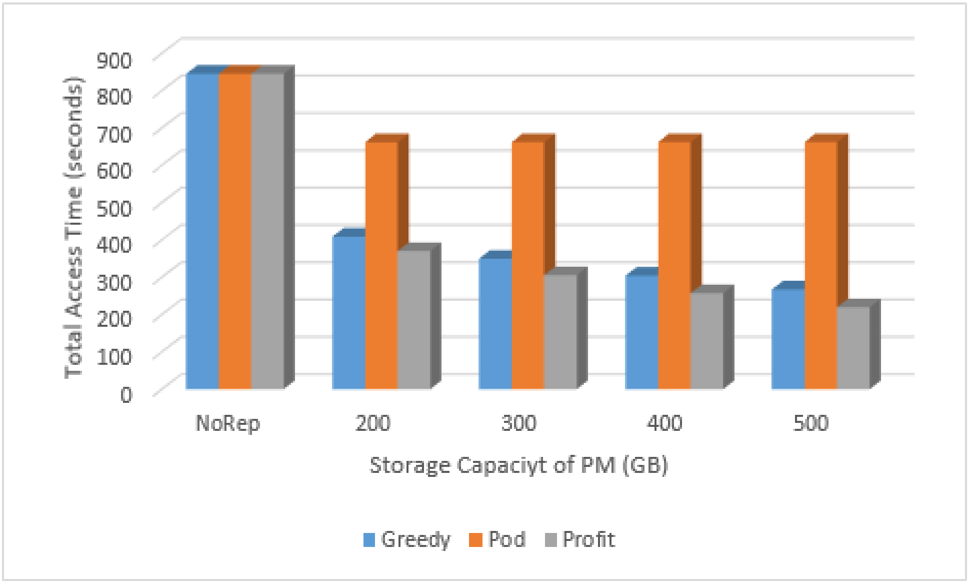


Fig. 8. Performance comparison for data center with 128 PMs with Zipf distribution access pattern.

Chapter 6

Conclusion and Future Work

6. Conclusion and Future Work

We studied file replication problem in data intensive cloud data centers, and designed a time-efficient approximation algorithm with performance guarantee. It was based on a novel concept called “profit”, and optimizes over a submodular function that can be computed efficiently. Our algorithm reduced the total energy consumption of data access by at least half of what is achieved by an optimal replication solution. We also designed two energy- and time-efficient heuristic file replication algorithms. Currently, we assume that the VMs that execute user jobs stay in a particular PM for its entire lifetime. As future work, it would be interesting to investigate how dynamics of VM migration can interplay with the data replication, to better achieve the energy efficiency in cloud data centers.

REFERENCES

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

[2] D. Boru, D. Kliazovich, F. Granelli, P. Bouvry, and A. Y. Zomaya. Energy-efficient data replication in cloud computing datacenters. Cluste Computing, 18(1):385–402, 2015.

[3] D. Boru, D. Kliazovich, F. Granelli, P. Bouvry, and A. Y. Zomaya. Models for efficient data replication in cloud computing datacenters. In Proc. of the IEEE ICC, 2015.

[4] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience (SPE), 41(1):23–50, 2011.

[5] X. Dong, T. El-Gorashi, and J. M. H. Elmirghani. Green ip over wdm networks with data centers. JOURNAL OF LIGHTWAVE TECHNOLOGY, 29(12):1861–1880, 2011.

[6] L. Durbeck, N. Macias, and J. Tront. Energy efficiency of zipf traffic distributions within facebook's data center fabric architecture. In Proc. 25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2015.

[7] M. Karlsson and C. Karamanolis. Choosing replica placement heuristics for wide-area systems. In Proc. of the IEEE ICDCS, 2004.

[8] W. Li, Y. Yang, and D. Yuan. A novel cost-effective dynamic data replication strategy for reliability in cloud data centers. In Proc. Of the International Conference on Dependable, Autonomic and Secure Computing (DASC), 2011.

[9] B. Lin, S. Li, X. Liao, Q. Wu, and S. Yang. estor: energy efficient and resilient data center storage. In Proc. of the International Conference on Cloud and Service Computing (CSC), 2011.

[10] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In Proc. of IEEE INFOCOM, 2010.

[11] D. T. Nukarapu, B. Tang, L. Wang, and S. Lu. Data replication in data intensive scientific applications with performance guarantee. IEEE Transactions on Parallel and Distributed Systems, 22(8):1299–1306, 2011.

[12] F. Ping, X. Li, C. McConnell, R. Vabbalareddy, and J. H. Hwang. Towards optimal data replication across data centers. In Proc. of the International Conference on Distributed Computing Systems Workshops (ICDCSW), 2011.

[13] L. Qiu, V. Padmanabhan, and G. Voelker. On the placement of web server replicas. In Proc. of the IEEE INFOCOM, 2001.

[14] V.K. Mohan Raj and R. Shriram. Power management in virtualized datacenter - a survey. Journal of Network and Computer Applications 69:117–133, 2016.

[15] N. Sharma, S. Barker, D. Irwin, and P. Shenoy. Blink: Managing server clusters on intermittent power. SIGARCH Comput. Archit. News, 39(1):185–198, 2011.

[16] B. Wang, Z. Qi, R. Ma, H. Guan, and A. V. Vasilakos. A survey on data center networking for cloud computing. Computer Networks, 91:528–547, 2015.

Appendix A
Selected Parts of the Code

A.1 Calculating the Total Access Time

There are two ways to calculate the total energy consumption first one is by considering each host and then search for the files in the host itself, hosts in the same edge, hosts in the same pod, hosts outside of the pod (or in the central database).

```
  public double
totalAccessTime(ArrayList<org.cloudbus.cloudsim.checkpoint.centralexample.Fil
e> FileList,double [][] accessrates, ArrayList<NetworkHost> hosts){
              double energyconsumption =0;
          double totalBandwidth= 0;


          boolean found= false;


          for (int j=0 ; j < hosts.size(); j++){

              for (int i = 0 ; i < FileList.size() ; i++ ){

                 if (hosts.get(j).localDataBase.contains(i) == true){  // The
file is in the same machine COST= 0

                     found = true;
                         energyconsumption += 0d * (accessrates[j][i]);
                         totalBandwidth+= 0d *  (accessrates[j][i]); //in
second
                         countInHost++;
                        if(j == host){
                               System.out.println("The File : " + i + " In PM
1 :"+"  The Accessrate is :" + accessrates[j][i]);
                               System.out.println("The one time access is  "
+ 0 );

                               PM1agrr+= 0;
                               PM1agrrZiph+= 0 * accessrates[j][i];
                               System.out.println("The multiplication is :" +
0 * accessrates[j][i] );
                         }

                 }
                 if (found != true){ //file is not in the same machine;
                 for(int ii = 0 ; ii < hosts.get(j).edge.size(); ii++){ //
for the cost from other machines in the same edge Cost 3;
                         if (
hosts.get(hosts.get(j).edge.get(ii)).localDataBase.contains(i) )
                         {

                            found= true;//the file is in the edge
                            countInEdge++;
                            energyconsumption += 1d * (accessrates[j][i]);
```

```
                              totalBandwidth+= 4d * (accessrates[j][i]) ; //in
seconds

                         break;
                            }}}
                    if (found != true){
                            for(int ii = 0 ; ii < hosts.get(j).pod.size();
ii++){ //for the cost from machines in the same pod COST 7
                                if
(hosts.get(hosts.get(j).pod.get(ii)).localDataBase.contains(i)){
                                    countInPod++;

                                   found = true; // file in pod

                                    energyconsumption += 7d *
(accessrates[j][i]);

                                    totalBandwidth+= 6d *
(accessrates[j][i]) ;  //in seconds
                                    break;
                            }
                        }
                    }
                    if (found != true){// not in the pod at all take it from
database with COST 22
                            {


                              found = true; // file out of pod (or Central
database)
                                energyconsumption += 22d *
(accessrates[j][i]);
                                totalBandwidth+= 6.8d * (accessrates[j][i]);
//in seconds

                             }
                        }
                    found = false;
                    }
                        // machine j has been checked against file i
                   }// file i is done let's move to second file

          return energyconsumption; // if I want the total access time I
manually change this line to return totalBandwidth.
        }
```

The other way is to calculate the Total Access Time and the energy consumption using the distances. We have 0 distance when the file is in the host, 1 distance when the file is in the same edge, 3 distance when the file is in the same pod, 5 distance if the file is not in the same pod or in the Central Database.

```
public int TotalAccessTime2(ArrayList<FileCopy> p){

        double energyConsumption = 0;
         double TotalAccessTime = 0;
         for( int i = 0 ; i < p.size(); i++){
                if ( p.get(i).distance == 0){
                      energyConsumption += 0 * rates[p.get(i).hostID][p.get(i).fileID];
                       TotalAccessTime += 0;
                }

                if ( p.get(i).distance == 1){

                      energyConsumption += 1 * rates[p.get(i).hostID][p.get(i).fileID];
                       TotalAccessTime += 4 * rates[p.get(i).hostID][p.get(i).fileID];
                }

                if(p.get(i).distance == 3){


                energyConsumption +=  7 * rates[p.get(i).hostID][p.get(i).fileID];
                      TotalAccessTime += 6 * rates[p.get(i).hostID][p.get(i).fileID];

                }
               if(p.get(i).distance == 5){

               energyConsumption += 22 * rates[p.get(i).hostID][p.get(i).fileID];
               TotalAccessTime += 6.8 * rates[p.get(i).hostID][p.get(i).fileID];

                }

        }
        System.out.println("The total accesstime from TotalAT :" + totalAccessTime);
        System.out.println("The total bandwith  from TotalAT :" + totaltime);
        return 0 ;
            }
```

The helper function to allocate each file and the distance between this file and the hosts.

```
public void Disallocator(){
        int id;
        int FileID;
        for(int i = 0 ; i < hostListg.size() ; i++) {

                for(int j = 0 ; j < hostListg.get(i).localDataBase.size(); j++){

                       FileID= hostListg.get(i).localDataBase.get(j);

                         pairs.get(i*1000 + FileID).isThere = true;
                   pairs.get((i*1000) + FileID).distance  = 0;

                      for(int t = 0 ; t < hostListg.get(i).edge.size(); t++){
                             id = hostListg.get(i).edge.get(t) *1000;
                        if(pairs.get(id+ FileID ).distance > 1){

                                   pairs.get(id+ FileID).distance = 1;

                             }
                      }

                    for(int x = 0 ; x < hostListg.get(i).pod.size(); x++){
                       id = hostListg.get(i).pod.get(x) *1000;
                   if(pairs.get(id + FileID).distance > 3){

                                   pairs.get(id + FileID).distance = 3;
```

```
                           }

                       }


               }


           }
                  }
```

A.2 Profit Algorithm

A.2.1 Find the Replica Effect for a Host and a File

```
public double replicaEffect (int hostID , int
fileID,ArrayList<org.cloudbus.cloudsim.checkpoint.centralexample.File> FileList,
ArrayList<NetworkHost> hosts  ){
        double replicaE=0;  // to calculate the aggregated replica effect
         int id ;

         id = hostID*1000;
         if(pairs.get(id + fileID).distance == 5){ //File is Not in Pod
         replicaE+= (6.8) * rates[hostID][fileID];

         for(int i = 0 ; i < hosts.get(hostID).edge.size(); i++){

         id = hostListg.get(hostID).edge.get(i) * 1000;

         if(pairs.get(id + fileID ).distance > 1){

           replicaE+= (6.8 - 4) * rates[hosts.get(hostID).edge.get(i)][fileID];

                             }

                  }


           for(int i = 0 ; i < hosts.get(hostID).pod.size(); i++){

           id = hostListg.get(hostID).pod.get(i) * 1000;
           if(pairs.get(id+ fileID ).distance > 3){
           replicaE+= (6.8 - 6) * rates[hosts.get(hostID).pod.get(i)][fileID];
                           }
                           }
                  }

            if(pairs.get(id + fileID).distance == 3){ // File is in the Same pod /not
edge

               replicaE += (6 - 0) * rates[hostID][fileID];

               for(int i = 0 ; i < hosts.get(hostID).edge.size(); i++){
               id = hostListg.get(hostID).edge.get(i) * 1000;
```

```
                id = hosts.get(hostID).edge.get(i) *1000;
                if(pairs.get(id + fileID ).distance > 1){

                replicaE +=  (6 - 4) *  rates[hosts.get(hostID).edge.get(i)][fileID];

             }

                  }
                 }

                 if(pairs.get(id + fileID).distance == 1){ // File in the same edge


                    replicaE += 4 * rates[hostID][fileID];



              return   replicaE; // return the aggregated effect for this replica
        }
```

A.2.2 Code for Checking All the replica effects

After we check all the replicas effects we then choose the winners and make the actual replication

```
public int minAccessTime
(ArrayList<org.cloudbus.cloudsim.checkpoint.centralexample.File> FileList,
ArrayList<NetworkHost> hosts){


        double maxReduction = Double.MIN_VALUE;
        double value= 0;
        int winHost=0 ;
        int winFile= 0;
        int id;
       for(int d = 0 ; d <  (hosts.get(0).m * hosts.size()); d++){  //Number of round to
fill the datacenter (Note some of these rounds are not counted in case there is no Central DataBase)

              maxReduction = Double.MIN_VALUE;


             for (int i = 0 ; i < FileList.size(); i ++){

               for (int j= 0 ; j < hosts.size(); j++){

 if(hosts.get(j).localDataBase.size() >= hosts.get(j).m
hosts.get(j).localDataBase.contains(i)) // if Host is full or contain the file already
                              {
                               value = Double.MIN_VALUE;
                              }
                              else{

              value= replicaEffect(j, i, FileListg, hostListg); // calculate the
replica effect for this round
                             if ( value > maxReduction){
                                    maxReduction = value;
                                    winHost = j;
                                    winFile= i;

                               }
                                }
                   }
                   }

                   System.out.println("The winners are host " + winHost + " and File :" +
winFile);

//The actual replication starts here
if(hostListg.get(winHost).localDataBase.size() < hostListg.get(winHost).m
&& !hostListg.get(winHost).localDataBase.contains(winFile)){
                 hostListg.get(winHost).localDataBase.add(winFile);

            pairs.get(winHost*1000 + winFile).isThere = true;
           pairs.get((winHost*1000) + winFile).distance  = 0;



              for(int i = 0 ; i < hosts.get(winHost).edge.size(); i++){
                     id = hosts.get(winHost).edge.get(i) *1000;
               if(pairs.get(id+ winFile ).distance > 1){

                          pairs.get(id+ winFile).distance = 1;

                     }
              }

             for(int i = 0 ; i < hosts.get(winHost).pod.size(); i++){
                id = hosts.get(winHost).pod.get(i) *1000;
           if(pairs.get(id + winFile).distance > 3){

                          pairs.get(id + winFile).distance = 3;

                     }


               }
               }
    //The actual replication Ends here

        }

          totalAccessTime(FileListg, rates, hostListg); Calculate the Total access Time
Or the Total energy consumption based on the user desire
         return 0;
      }
```

A.3 Greedy Algorithm

```
public double
greedyAlgo(ArrayList<org.cloudbus.cloudsim.checkpoint.centralexample.File>
FileList,double [][] accessrates, ArrayList<NetworkHost> hosts){
       int redTotalAccessTime = 0; // The reduced total access time
       boolean done = true;
```

```
        double count;


         for (int i = 0 ; i < hosts.size(); i ++){

              count = 100;
              done =true;

          while (done){

          for (int j = 0 ; j < FileList.size(); j ++){

              if (accessrates[i][j] == count &&
              !hostListg.get(i).localDataBase.contains(j)){

              hosts.get(i).localDataBase.add(Integer.parseInt(FileList.get(j).g
              etName()));



                          }

                    if(hosts.get(i).localDataBase.size() == hosts.get(i).m){

                                     break;

                                    }

                          }
              count --; // after checking all the 100 check the 99 and so on …
              if(count == 0 || hosts.get(i).localDataBase.size() ==
              hosts.get(i).m) done=false;

                    }
        }
         return totalAccessTime(FileListg, rates, hostListg); //calculate the
total access time or the total energy consumption
       }
```

A.3.1 Greedy Algorithm with Zipf

With Zipf, the Greedy Algorithm is a special case since each host will replicate the first files as his storage can take. For example, with storage of 500GB for per host, each host will replicate the first 250 files. For simplicity, a special function were coded for this purpose

```
 public double greedyAlgo2(){ //this is espical case for Ziph

    for (int i = 0 ; i < hostListg.size(); i ++){
    for( int j = 0 ; j < hostListg.get(i).m; j++){
           if(hostListg.get(i).localDataBase.size() < hostListg.get(i).m &&
      !hostListg.get(i).localDataBase.contains(j) )
            hostListg.get(i).localDataBase.add(FileListg.get(j).id);
```

```
          }
         }

         return totalAccessTime(FileListg,rates,hostListg);
          }
```

A.4 Pod-Based Algorithm

```
public int
podBasedAlgo(ArrayList<org.cloudbus.cloudsim.checkpoint.centralexample.File>
FileList,double [][] accessrates, ArrayList<NetworkHost>
hosts,ArrayList<filesData> files ){

       boolean done = true;
       int count =1;
       Collections.sort(files, new Comparator<filesData>(){
               public int compare(filesData o1, filesData o2){
                  return  Double.compare(o2.sumPerPod , o1.sumPerPod);
               }
            }); // sort the files array to have the hot files

       // sort highest servers for each file
      for (int i = 0 ; i< files.size() ; i++){
            Collections.sort(files.get(i).rest, new Comparator<PointZ>(){
               public int compare(PointZ o1, PointZ o2){
                  return Double.compare(o2.y , o1.y);
               }
            });
      }

       Collections.sort(files, new Comparator<filesData>(){
               public int compare(filesData o1, filesData o2){
                  return Double.compare(o2.sumPerPod , o1.sumPerPod);
               }
            });


       for ( int i = 0 ; i < files.size() ; i ++){

             if (hosts.get(files.get(i).MaxID).localDataBase.size() <
hosts.get(files.get(i).MaxID).m &&
!hosts.get(files.get(i).MaxID).localDataBase.contains(FileList.get(files.get(
i).column).getName())){//check if the local database if full

hosts.get(files.get(i).MaxID).localDataBase.add(Integer.parseInt(FileList.get
(files.get(i).column).getName()));//add the file in the local database of the
host with the highest accessrate
             } else { // if the host is full put the file in the second
server with the highest rate ( for example if the host with 10 is full then
put it in the host with 9...)

 for(int j= 0 ; j < files.get(i).rest.size() ; j++){
 if (hosts.get(files.get(i).rest.get(j).x).localDataBase.size() <
hosts.get(files.get(i).rest.get(j).x).m &&
!hosts.get(files.get(i).rest.get(j).x).localDataBase.contains(Integer.parseIn
t(FileList.get(files.get(i).column).getName())) ){

hosts.get(files.get(i).rest.get(j).x).localDataBase.add(Integer.parseInt(File
List.get(files.get(i).column).getName()));
                        break;
                         }





                  }




             }
             }






      return 0;
     }
```

To decide what are the hot files for each pod I used a helper function:

```
public ArrayList<filesData> podBasedC2(double [][] accessrates , int k){

       int min= k;

       ArrayList<filesData> files = new ArrayList<filesData>();
       for ( int j = 0 ; j < accessrates[0].length; j ++)
   {
             filesData file = new filesData();
             double aggregatedSum = 0 ;
             double temp= Double.MIN_VALUE;
             file.MaxID=0;

       for(int i= min ; i < (min + ((Config.portNum/2)*(Config.portNum/2))); i++){
             PointZ p = new PointZ();

                  aggregatedSum+= accessrates[i][j];
                  temp = accessrates[i][j];
                  p.x = i;
                  p.y = temp;
                  file.row =i;
                  if(temp > file.maxValue){ file.maxValue = temp;
                                                 file.MaxID= i;
                              ;}
                        file.rest.add(p);
       }
       file.column= j;
       file.sumPerPod= aggregatedSum;
       files.add(file);


       }
```

```
        return files;
      }
```

Also, to represent each file as a competitive file for being hot I used a helper class

```
public class filesData {


      public int row = -1; //which row
      public int column = -1; // which column
      public int MaxID = -1; //the Host Where this file should be stored (has
the max access rate);
      public double sumPerPod = -1; // the total accessRate for this file in
a pod
      public double maxValue= -1;
      public ArrayList<PointZ> rest = new ArrayList<PointZ>(); // the rest of
hosts that can hold this file


}
```