Resource Allocation and Request Handling for User-Aware Content Retrieval in the Cloud

Boyang Yu and Jianping Pan
Department of Computer Science
University of Victoria
Victoria, BC, Canada
Email: {boyangyu, pan}@uvic.ca

***Abstract*—The user-aware content retrieval services are always data-intensive and require much resource to satisfy the user demand, which incurs the high cost of implementation. Considering that they could largely exploit the pay-as-you-go paradigm and the almost unlimited resource pool of the cloud, we investigate the design issues of deploying the services to the cloud in a cost-effective way. We formulate the resource allocation and request handling problem which aims at lowering the deployment cost and guaranteeing the service quality simultaneously. Due to the hardness of obtaining an optimal solution, we design two approximate algorithms with different points of emphasis and analyze their approximation ratios as well. In addition, we discuss the implementation issues in applying the proposed algorithms to the practical systems. Finally, the algorithms are evaluated and validated through both trace-based and synthesized simulations where they show a large improvement in terms of the total system cost.**

***Index Terms*—resource allocation, data placement, request handling, optimization problem, approximate algorithms**

I. Introduction

The demand for content retrieval contributes to a large portion of the Internet traffic [1], and the user-awareness or personalized functions become important features in most of the network services today, since they make a large improvement of the user experience. We list some examples of the user-aware functions as follows: 1) profiling: the showing of personalized contents such as user icons or the personalized recommendation results of videos/posts/advertisement when the user opens a webpage of video-website/OSN/etc.; 2) accounting: for some professional websites, such as e-learning, the user access to webpages should be authenticated and accounted for the purpose of access control and billing. In each transaction of such user-aware functions, the access to system-level-shared contents is always done along with the access to the user-specific data. In the traditional design, the types of data involved in each user transaction are not differentiated which degrades the system performance to some extent. In this paper, we investigate the design issues of user-aware content retrieval services in the cloud to overcome it.

When the content retrieval services are in a large scale, we have to distribute both the incoming requests and stored data objects to a large number of serving machines, which makes the cost-effective design necessary. Meanwhile, the innovative cloud infrastructures [2] can provide computational and storage resources to the services and even offer a high flexibility in the allocation. Service providers may benefit from the pay-as-you-go cost model and the elastic resources in the cloud, if they can reconcile the needs from both resource allocation and utilization.

The *resource allocation* should be optimized, in order to minimize the monetary cost of the content retrieval service provider. For example, the number of Virtual Machines (VM) to support the whole service should be minimized and each one of them has a limited maximum service rate. In this paper, the costs of VMs and storage in the system are both considered. In terms of *resource utilization*, the replication of data objects, the data object location, and the handling of requests are some important issues that need to be addressed properly.

An important application-specific characteristic in the discussed services is: to validate a user, update its profile and provide the personalized data is the necessary and first job in the request handling. This results in that each single request would involve both a user-type object and a content-type data object. It makes the cost-effective design on resource allocation and utilization complicated. The co-location of the two involved objects is favored in terms of the access efficiency (and consequently the lower VM cost), but this would inevitably yield a larger extent of data replication, which conflicts with the favoring on a lower storage cost. Note that although user profiles are less intensive in traffic volumes, the frequent access and update necessitate a careful consideration similar to that for the requested contents.

To these aspects, we model the costs and constraints in the cloud as well as the two-stage jobs in handling a content retrieval request: user profiling and content fetching. Then the *Resource Allocation and Request Handling (RARH)* problem is formulated. It can be summarized as: given the set of users, set of contents and predicted content request rates from each user to each content, how to efficiently allocate the cloud-based resources, i.e., storage and VM nodes, and how to make these allocated resources connected as a system to handle requests. In the formulated problem, any content retrieval request would be fulfilled in one or two hops after it gets into the data center, which results in a quite low response time and avoids the congestion in the data center networks.

Solving the *RARH* problem helps to reduce the cost of the system while ensuring the service quality. However, the problem is NP-Hard, so two approximate algorithms are proposed.

For a higher access efficiency, the *One-Layer Serving (OLS)* method is devised which lowers the system traffic and response time by fulfilling the requests in only one hop and achieves a much lower storage cost than the common Decreasing First-Fit algorithm [3] by the proposed partition-based packing method. For the higher storage efficiency, we devise the *Two-Layer Serving (TLS)* method, which adopts a two-layer serving structure and significantly lowers the storage cost through the request redirection. Besides, the scheme to help the algorithms implemented in practice is presented, where how to predict the request rate matrix, the input of the algorithm, and how to provision the service based on the algorithms are discussed.

In the literature, the methods in building the user-aware content retrieval services on the cloud were still elusive, although some other applications in the cloud have attracted certain attentions, such as the interactive applications [4], Online Social Network (OSN) [2], etc. In this paper, with the joint considerations of user-awareness, efficient resource utilization, and cloud-based cost model, we justify the design issues in a user-aware content retrieval service and propose some novel methods in lowering the system cost while maintaining the service quality, which are further analyzed and evaluated through analytical approaches and simulations.

The rest of the paper is organized as follows. Section II gives the modeling framework and problem formulation. Section III presents the approximate algorithms. Section IV discusses the implementation issues and Section V makes the evaluation through simulations. Then we present the related work and conclusions in Section VI and Section VII, respectively.

II. Modeling Framework

A. Preliminaries

In this paper, the issues in building a user-aware content retrieval service based on the Infrastructure as a Service (IaaS) clouds are discussed from the perspective of the *service provider*, who rents the resources provided by IaaS clouds in a data center to build the service. The *end users* of the system are clustered beforehand into groups, under a certain criterion (such as by the geo-location of the user), or randomly (such as by hashing each user ID to an integer in a small range as the group ID). With this coarsening of granularity, multiple users are treated as one user. It lowers the computational overhead and the amount of information to be maintained in the proposed scheme, so that the number of individual users could be much large. Below we abbreviate a group of users as a *user*, and the request rate to be discussed is also at the group level. From the perspective of database system, the user discussed here can be considered as a shard or horizontal portion of the user table. The main traffic to be handled by the system is the retrieval requests for different *contents* from different users. The *contents* to be retrieved are also clustered into groups and accounted at the group level for the same coarsening purpose.

In the IaaS clouds considered, there are two types of entities, *VM nodes* and *storage volumes*. The VM nodes provide the computation resource in serving the user requests. The storage volume provides the persistent storage of data objects, and each of them is mounted to a specific VM node and can be directly accessed only through that node. So if a data object is stored in a storage volume mounted to a VM, we can simply say that the object is stored in the VM. As multiple storage volumes can be mounted to the same VM, although each has a size limit, it can be equally considered as we can allocate and mount a storage volume of any size to a VM.

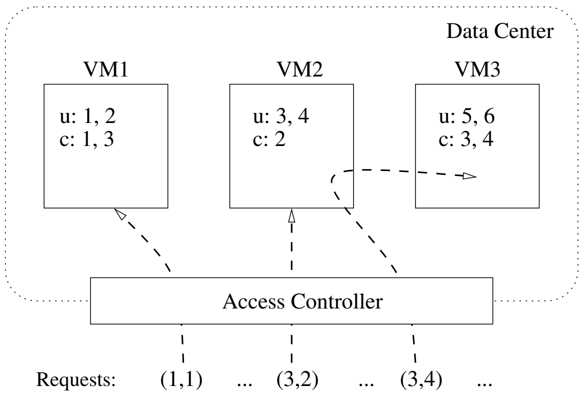

Fig. 1. Example of a user-aware content retrieval system

The retrieval requests are satisfied with the help of VMs and storage. Denote the available node set by V with cardinality N and use n to represent a specific node. In the system only a portion of VMs in V is allocated to reduce the cost. A storage volume can be allocated and mounted to an allocated VM, whose size is determined by the data objects stored.

B. System Model

Denote U the set of users directed to a data center with the cardinality I and use i to denote a specific user where $i = 1, .., I$. Denote F the set of content objects to be retrieved by the users with the cardinality J and use j to denote a specific content where $j = 1, .., J$. We use (i, j) to denote the request from user i to content j and use λ_{ij} to denote its average arrival rate. Then we obtain the set of requests, such as $R = U \times F = \{(i, j) | i = 1, ..., I, j = 1, ..., J\}$, and the load matrix containing the average arrival rates of requests, such as $A = \{\lambda_{ij} | i = 1, ..., I, j = 1, ..., J\}$. Given the predicted load matrix A for a future interval t, denoted by $A(t)$, an optimal solution for that interval t is to be obtained. Each $\lambda_{ij}(t)$ in $A(t)$ can be predicted based on its history and more details on the prediction method will be given in Section IV.

We model the fulfillment of a request as two stages: 1) the request should be checked and accounted based on its user profile i in the request (i, j), and then 2) the content would be fetched based on the requested content j in (i, j). The first stage is necessary for a user-aware system for the reasons such as user profiling, user accounting and personalization. As a result, some data related to the specific users need to be stored and updated. Since the data objects necessary in the two stages are different, they are differentiated as *user-type* and *content-type*, respectively. In the example illustrated in Fig. 1, there are three VMs allocated and the set of content-type and user-type objects stored in each VM are listed with the prefix `c:` or `u:`, respectively.

We assume each request should be fulfilled within two hops after it enters the data center, which ensures the access efficiency. A larger hop count implies a larger response time due to the queueing at more machines, and also yields more traffic overhead in the local network. Then a request is fulfilled in the procedures as follows: 1) in serving a request (i, j), the request should firstly be directed or mapped to an access node n_a and it is necessary that the user-type object about user i is stored at n_a; 2) then the request to the content can either be satisfied at n_a or a second-hop node n_r through the relaying of n_a. But in either case, the requirement is that the content j should be ready at the final serving node. In Fig. 1, request $(1, 1)$ and $(3, 2)$ are satisfied locally at VM1 and VM2, respectively. But request $(3, 4)$ is satisfied through the relay from VM2 (to obtain the user-type data `u:3`) to VM3 (to obtain the content-type data `c:4`). Also note that the same data object can appear in more than one VMs, such as `c:3`.

C. Costs, QoS Constraint and Objective

Costs: In the paper, the costs considered include: the unit cost per VM, denoted by C_v, and the cost of each user-type and content-type data object, denoted by C_u and C_c, respectively. We assume a homogeneous size of data objects within each type for the ease of discussion, which can also be considered as the allocation is based on the maximum allowed size of each object. We expect to lower the total cost of VMs and storage, but have to face the conflicts in lowering them together. So a tradeoff is made by having different points of emphasis in the respective algorithms designed below.

QoS Constraint: Denote the request arrival rate and service rate at each VM by λ and μ, respectively. To ensure the queue stability at each VM, $\lambda \leq \mu$ is necessary. We can further model the service at each VM as an M/M/1 queue or even some other types of queues, based on the actual arrival and service rate distribution. With such a queueing model, the constraint can be further tightened as $\lambda \leq \mu_{qos}$, where $\mu_{qos} \leq \mu$. This smaller μ_{qos} can ensure a certain system time. For example, in an M/M/1 queue the system time is $1/(\mu - \lambda)$ [5]. To achieve a system time T, it is necessary $\lambda \leq \mu_{qos} = \mu - 1/T$. Based on the exact queuing type in a real system, μ_{qos} can be obtained. In the following formulation, without loss of generality, we apply $\lambda \leq \mu$ as the constraint.

Objective: The objective is to build a system supporting the predicted traffic load based on the cloud infrastructure. We aim at minimizing the monetary costs on the allocation of VMs and storage while ensuring that the arrival rate (or system time) at each VM is below the threshold μ (or T).

D. Resource Allocation and Request Handling Problem

Remind that we have modeled the request rate matrix $A = \{\lambda_{ij} | i = 1...I, j = 1...J\}$ which gives the average request rate for all the items in R, the request set. For each request (i, j), it should be directed to an access node. The expected maximum rate at a single node is defined as μ, which means although different requests can be mapped to the same node, the feasible mapping is still limited by the servicing capacity of VM. The mapping function can be described as

$$f : (i, j) \rightarrow n \ , \tag{1}$$

and the binary variable $x^{[m]}_{ijn}$ is used to denote whether the request (i, j) is mapped to the access node n, where $[m]$ indicates the variable category (here $[m]$ is for the mapping, others categories are $[v]$ for VM, $[u]$ for user-type objects, $[c]$ for content-type objects and $[r]$ for the relaying relationship).

We formulate the *Resource Allocation and Request Handling (RARH)* problem, in the form of binary programming. It is presented in (2)–(8), where (2) is the objective function, specifying that the total cost on nodes, user-type and content-type storage should be minimized. In (2), whether node n is allocated is represented by the binary variable $x^{[v]}_n$. The cost on the VMs is determined by the number of VMs allocated. The binary variable $x^{[u]}_{in}$ tells the availability of the user-type object related to user i at VM n. The binary variable $x^{[c]}_{jn}$ indicates whether content-type object j is stored at VM n. The total cost to minimize is expressed as a linear function of the number of VMs, the sum of the number of different content-type objects in each VM, and the sum of the number of different user-type objects in each VM.

$$\textbf{min} \quad C_c \sum_{j \in F} \sum_{n \in V} x^{[c]}_{jn} + C_u \sum_{i \in U} \sum_{n \in V} x^{[u]}_{in} + C_v \sum_{n \in V} x^{[v]}_n \tag{2}$$

s.t.

$$\sum_{i \in U} \sum_{j \in F} \lambda_{ij} (x^{[m]}_{ijn} + \sum_{n_a \in V} x^{[m]}_{ijn_a} x^{[r]}_{ijn_a n}) \leq \mu x^{[v]}_n, \forall n \in V \tag{3}$$

$$x^{[r]}_{ijn_a n} \leq x^{[c]}_{jn}, \forall i \in U, j \in F, n_a \in V, n \in V \tag{4}$$

$$x^{[m]}_{ijn} \leq x^{[u]}_{in}, \forall i \in U, j \in F, n \in V \tag{5}$$

$$\sum_{n \in V} x^{[m]}_{ijn} = 1, \forall i \in U, j \in F \tag{6}$$

$$x^{[m]}_{ijn} \leq x^{[c]}_{jn} + \sum_{n_r \in V} x^{[r]}_{ijnn_r}, \forall i \in U, j \in F, n \in V \tag{7}$$

$$\begin{aligned} x^{[m]}_{ijn}, x^{[v]}_n, x^{[u]}_{in}, x^{[c]}_{jn}, x^{[r]}_{ijn_a n_r} \in \{0,1\}, \forall i \in U, j \in F,& \\ n_a \in V, n_r \in V.& \end{aligned} \tag{8}$$

About the constraints, (3) represents the QoS constraint at each VM n, such as, if any node n is allocated, the sum of the access traffic (first hop) and the redirected traffic (second hop) to that node should be lower than the maximal load μ. In (3), the variable $x^{[r]}_{ijn_a n}$ is used to represent whether the request (i, j), which was initially mapped to access node n_a, will (or will not) be relayed to another node n by 1 (or 0). For those requests to content j, their redirection from access node n_a to another node n is possible to be enabled only if there is a copy of j at the node n, which is shown through (4). (5) defines the user-type object requirement at the access node, which ensures request (i, j) can be mapped to n only if there is a copy of user-type object related to i in node n and (6) defines each request should be mapped to one and only one access node. It is also necessary that if a request is mapped to a node, it should be either fulfilled at the node, or redirected to another node n_r to retrieve the content, which is shown in (7). (8) gives the binary variables used in the formulation.

In fact, the formulated problem applies the constraint of at most *two-hop* in the request fulfillment to the formulation. However, to improve the access efficiency and avoid network congestion, we could change it to the *one-hop* constraint, such that the request to the content should be completely fulfilled at the access node. Then the relaying is not needed and the necessary condition for $x_{ijn}^{[m]} = 1$ is that the data objects related to user i and content j are both available at the node n. Moreover, the one-hop case is equivalent to that we set all relaying variables, i.e., $x_{ijn_an_r}^{[r]}$, to 0 in the formulation. Below we would propose two algorithms oriented to the one-hop or two-hop case, respectively.

III. Algorithms

A. Access-Efficient Method

The *RARH* problem is NP-hard [6] and the number of variables in the problem formulation is $O(IJN^2)$, so it is meaningful to design an approximate algorithm that solves the problem in the polynomial time. Note that we use N_{opt} to represent the result of the optimal solution and use N_{apx} to represent that of the approximate solution. An algorithm is with approximation ratio r if N_{apx} based on its solution can always ensure $N_{apx} \leq rN_{opt} + \delta$, where δ is a small constant.

Here we will solve the problem with the one-hop fulfillment constraint for a better access efficiency. In the one-hop case, the traffic of the second hop is not needed which lowers the traffic inside the data center and only one VM access is enough for the request fulfillment which improves the response time.

It is noticed that if the storage cost is ignored such that $C_u = C_c = 0$, the problem is the same as the traditional bin-packing problem [3], and it is equivalent to packing requests (with size λ_{ij}) as items into VMs (with capacity μ) as bins. In the literature of bin-packing [3], it is proved that First-Fit gives a 2-approximation ratio and its general idea is to pack each item into the first bin that can fit it. An improved version, called Decreasing First-Fit, which sorts the items in a descending order by the size before the packing, gives a $3/2$-approximation ratio. Another approach is Next-Fit, which tries to pack an item into the last created bin without any search on the previous ones. However, since the storage cost is not negligible in our formulation, we need to devise a new solution that aims at minimizing the VM cost and storage cost simultaneously.

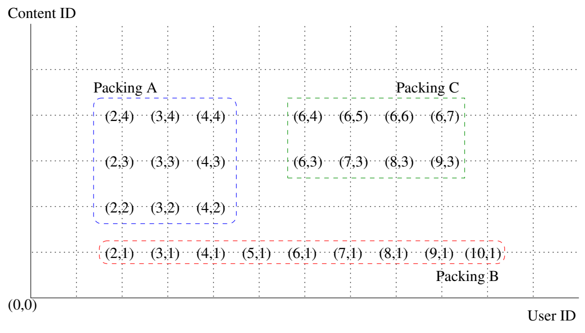

Fig. 2. Different cases of packing in the OLS algorithm

We propose an algorithm as shown in Alg. 1, termed as *One-Layer Serving (OLS)*. It tries to lower the storage cost through packing the requests from the same partition to the same VM. The partitions are specially designed intending to achieve a lower storage cost. An example will be presented to illustrate our idea, where all the request rates to be packed are in the same size 1, and the unit cost of the user and content object is the same. Fig. 2 shows a portion of the request rate matrix A and each shown point (i, j) represents a kind of request. From Fig. 2, Packing A is better than Packing B in terms of storage cost, because both of them pack 9 requests, however, the former packs the 3×3 square, introducing $3 + 3 = 6$ storage cost (3 in user-type and 3 in content-type), while the latter packs the 9×1 row, introducing $9 + 1 = 10$ storage cost. If the bin size is 9, we can use 3 and 3 as the width and height of partitions, which can achieve a lower storage cost than any other schemes to pack 9 items. In OLS, we use a similar rectangle to divide A into partitions. The width and height of each partition can be considered as the maximum allowed number of user-type and content-type objects in it, denoted by N_u and N_c, respectively.

Algorithm 1 *One-Layer Serving (OLS)*

```
1. Initialize bin set $S = \emptyset$; Obtain $N_u$ and $N_c$ through (10).
2. Remove an item $(i, j)$ from $R$. Calculate its partition ID through (11).
3. Obtain the list of existing bins in that partition and search the first-fit bin in the list to put $(i, j)$ into.
4. If failed in 3, create a new bin for $(i, j)$, set the partition ID of the bin to the same as $(i, j)$, and put it into $S$.
5. Repeat 2–4, until $R$ is empty.
6. Merge the bins in $S$. The $S$ after merging is the solution.
```

When $C_u = C_c$, to pack items in a square is always a good choice as shown from the example, such that $N_u = N_c$. When $C_u \neq C_c$, we can tradeoff the width and height in the rectangle by making $C_u/C_c \approx N_c/N_u$. For example, when $C_u : C_c = 1 : 2$, Packing C, where $N_u = 4$ and $N_c = 2$, is the best way to pack at least 8 items in terms of achieving the lowest storage cost. In such a case, the storage cost for the rectangle 4×2 is $4 \times 1 + 2 \times 2 = 8$, which is better than other cases, such as, 3×3 costs 9 and 2×4 costs 10. Through this idea, the area of the rectangle used in partitioning is set to the expected number of items in a bin, denoted by M, such as

$$M = \frac{\mu}{E[\lambda_{ij}]} = \frac{\mu}{\sum_i \sum_j \lambda_{ij}/(IJ)} , \tag{9}$$

and then N_u and N_c can be obtained as

$$N_u = \lfloor \sqrt{C_u M / C_c} \rfloor, N_c = \lfloor \sqrt{C_c M / C_u} \rfloor . \tag{10}$$

The matrix A is virtually divided into rectangle partitions with the width N_u and height N_c. For each item (or request) (i,j) to be packed into a bin (or VM), we calculate its designated partition ID, such as

$$partition(i,j) = (\lfloor i/N_u \rfloor, \lfloor j/N_c \rfloor) \ . \tag{11}$$

Next we try to pack that item into the first-fit bin with the same partition ID, which implies that a list of related bins should be maintained for each partition in the algorithm. If no one is found, create a new bin with that partition ID and put the item into it. In the resultant bin set after all the items are packed, some of the bins might be almost empty, so in Step 6, merge the bins by iterating all the created bins in S and checking whether there exist any two bins that the size after they are merged will not violate the bin capacity.

From the bin set S after merging, we can obtain the number of allocated VMs and the storage volume size for each VM, which guide the resource allocation. The data object location and the mapping function f are also obtained from S, which guide the resource utilization. The worst-case time complexity of *OLS* is in $O((IJ)^2)$.

Theorem 1 (Approx. Ratio of VM Cost): OLS gives a 2-approximation ratio in the part of VM cost.

Proof: This proof refers to that for First-Fit in [5]. In the resultant bin set, there are no two bins that can be merged together. Then if the bin number is even, for any two bins, the sum of their occupied size should be larger than μ, and we obtain $\mu N_{apx}^{[v]}/2 \le \sum_i \sum_j \lambda_{ij}$. It is obvious that $N_{opt}^{[v]} \ge \sum_i \sum_j \lambda_{ij}/\mu$, so we obtain $N_{apx}^{[v]}/N_{opt}^{[v]} \le 2$. Similarly, if the bin number is odd, except for one bin, all the others could be merged in pairs with a size larger than μ, so $\mu(N_{apx}^{[v]} - 1)/2 \le \sum_i \sum_j \lambda_{ij}$. Therefore $N_{apx}^{[v]} \le 2N_{opt}^{[v]} + 1$. ■

Theorem 2 (Approx. Ratio of Storage Cost): It is ensured that *OLS* at least gives a $J/N_c + N_u N_c$ (or $I/N_u + N_u N_c$) approximation for the user-type (or content-type) storage, if $I \mid N_u$ and $J \mid N_c$, where $I \mid N_u$ means I is divisible by N_u.

Proof: We prove the approximation ratio for the user-type objects here and that for the content-type is similar. For the use-type storage, the times of any user-type object i appearing in the resultant bin set are no more than $\lceil \sum_j \lambda_{ij}/\mu \rceil$, so $N_{opt}^{[u]} \ge \sum_i \lceil \sum_j \lambda_{ij}/\mu \rceil \ge I$. In *OLS*, $N_u \times N_c$ is used to partition matrix A. Only consider those whole-size partitions, and denote P_f (P_s) the number of partitions with a total rate larger (not larger) than μ. Since $P_s + P_f = \lfloor \frac{I}{N_u} \rfloor \lfloor \frac{J}{N_c} \rfloor$ and $\mu P_f \le \sum_{i,j} \lambda_{ij}$, $P_s \ge \lfloor \frac{I}{N_u} \rfloor \lfloor \frac{J}{N_c} \rfloor - \sum_{i,j} \lambda_{ij}/\mu$. Obviously $N_{apx}^{[u]} \le IJ$, the total number of requests. In the P_s partitions, each has a saving of $N_u N_c - N_u$ from IJ, so we obtain $N_{apx}^{[u]} \le IJ - (N_u N_c - N_u)P_s \le IJ - (N_u N_c - N_u)(\lfloor \frac{I}{N_u} \rfloor \lfloor \frac{J}{N_c} \rfloor - \sum_{i,j} \lambda_{ij}/\mu)$. Then by setting $\delta = IJ - \lfloor \frac{I}{N_u} \rfloor \lfloor \frac{J}{N_c} \rfloor N_u N_c$, we can obtain $N_{apx}^{[u]} \le IJ/N_c + N_u N_c \sum_{i,j} \lambda_{ij}/\mu + \delta$. Because $N_{opt}^{[u]} \ge \sum_i \lceil \sum_j \lambda_{ij}/\mu \rceil \ge I$, $N_{apx}^{[u]} \le (J/N_c + N_u N_c + \delta/I) N_{opt}^{[u]}$. When I and J can be divided by or much larger than N_u and N_c respectively, δ can be ignored. ■

From Theorem 2, we observe that the approximation ratio obtained is related to the chosen value of N_u and N_c. In the special case that $N_u = N_c = 1$, the ratio is $\max(I, J)$. The ratio proved here is still not very tight, and the performance is further investigated through simulations in Section V.

Lemma 1 (Approx. Ratio of Total Cost): With the result of Theorem 1 and 2, the approximation ratio of the total cost through OLS is no more than $\max(J/N_c, I/N_u) + N_u N_c$, if $I \mid N_u$ and $J \mid N_c$.

B. Storage-Efficient Method

With the two-hop request fulfillment allowed, it is not necessary to put the user-type and content-type object related to a request in the same node, which removes the dependence between the two types and potentially gives a much higher reduction in the storage cost. In the two-hop case, a method is devised aiming at a much higher storage efficiency although the access-efficiency has to be sacrificed to some extent.

We propose the *Two-Layer Serving (TLS)* method, as shown in Alg. 2. It sets two layers of nodes in the system: the first layer is for user accessing and only provides the user-type objects, while the second layer only consists of relay nodes with the requested contents. The sorting before the request packing for each layer significantly lowers the storage cost. In the request handling, any request is firstly mapped to an access node in the first layer, and later it is redirected to the node in the second layer and the content request is fulfilled there.

Algorithm 2 *Two-Layer Serving (TLS)*

```
1. Initialize bin set $S^{[i]} = \emptyset$, $S^{[j]} = \emptyset$, $S = \emptyset$.
2. Sort $R$ by index $i$, and if two with the same index, sort
   them by the $\lambda_{ij}$ in the decreasing order, denoted by $R_i$.
3. Insert items in $R_i$ into bin set $S^{[i]}$ using Next-Fit.
4. Sort $R$ by index $j$, and if two with the same index, sort
   them by the $\lambda_{ij}$ in the decreasing order, denoted by $R_j$.
5. Insert items in $R_j$ into bin set $S^{[j]}$ using Next-Fit.
6. Set up the redirection route between each request in $S^{[i]}$
   to the same request in $S^{[j]}$.
7. Obtain the set $S = S^{[i]} \cup S^{[j]}$. Merge the bins in $S$ when
   possible and output $S$.
```

Here we give more explanations. In the first layer, we need to pack R into a set of bins. Since only the user-type objects are needed at this layer, we can pack the requests with the same user-type i in the same bin as much as possible. First, all the requests are sorted according to the index i, and if some are with the same index, they will be sorted according to the object size. After this sorting, the objects are packed into bins using an approach similar to Next-Fit, which means each item is put into the current bin if it satisfies the capacity constraint. Otherwise, create a new bin as the current bin and put the item into the bin. Similarly, we pack R into bins again in the second layer, and the approach is similar to the first layer. After the two steps, we would have two layers of bins. In Step 6, we set up the redirection path between these two layers. Since for each item in R, it appears in both of the layers, we could easily build the connection between the layers by linking the same item in these two layers together. Obviously, the worst-case time complexity of *TLS* is in $O(IJ)$.

Theorem 3 (Approx. Ratio of VM Cost): TLS gives 4 approximation ratio in the part of VM cost.

Proof: The resultant bin set built in each layer through *TLS* still satisfies the initial condition in the proof of Theorem 1, so a similar approach applies. Therefore the cost of the first layer on VM satisfies $N_{apx}^{[i]} \leq 2N_{opt}^{[v]} + 1$. So does the second-layer cost $N_{apx}^{[j]}$. Therefore the total cost of the two layers on VM satisfies $N_{apx}^{[v]} \leq N_{apx}^{[i]} + N_{apx}^{[j]} \leq 4N_{opt}^{[v]} + 2$. ■

Theorem 4 (Approx. Ratio of Storage Cost): For each type of storage, *TLS* ensures an approximation ratio of 2.

Proof: Here we prove the ratio of the user-type storage, and the method for the content-type is similar. The total number of any user-type object i in the optimal solution should be no less than the minimum number of bins to only pack requests related to i, such as $N_{opt}^{[u]} \geq \sum_i \lceil \sum_j \lambda_{ij}/\mu \rceil$, where $\lceil \sum_j \lambda_{ij}/\mu \rceil$ specifies the minimum number of bins related to i. In *TLS*, the actual number of bins storing user-type object i is no more than the optimal number by 1, because of the sorting by i before packing. So the cost of our approach satisfies $N_{apx}^{[u]} \leq \sum_i (\lceil \sum_j \lambda_{ij}/\mu \rceil + 1)$. Therefore, the approximation ratio is $N_{apx}^{[u]}/N_{opt}^{[u]} \leq 1 + \frac{\sum_i 1}{\sum_i \lceil \sum_j \lambda_{ij}/\mu \rceil} \leq 2$. ■

Lemma 2 (Approx. Ratio of Total Cost): From Theorem 3 and 4, *TLS* gives a 4-approximation ratio in the total cost.

IV. Implementation Discussions

Two assumptions are made in applying the proposed algorithms to the real systems. We assume the request rates would show certain steadiness in a short interval, although they might change drastically after a long time-period. This ensures the solution for that short interval meaningful. We also assume the traffic pattern between two adjacent short intervals should be strongly correlated, so we can use the logged statistics to accurately predict the request pattern of the next interval. These two assumptions are often found in real application scenarios and existing research work [7].

With the assumptions, we propose an implementation scheme where the time is discretized into intervals. The statistics about the request rates of each interval is measured and collected. We use $\lambda_{ij}(t)$ and $\lambda_{ij}^{[e]}(t)$ to represent the *measured* and *predicted* arrival rate of request (i, j) at time interval t, respectively. We can easily measure each $\lambda_{ij}(t)$ at the access controller as shown in Fig. 1, by simply counting the number of requests to (i, j) at the interval t. When the system scale is large, it is necessary to have a cluster of homogeneous access controllers to equally distribute the incoming traffic. In such a case, $\lambda_{ij}(t)$ is the sum of the measured results at every access controllers. With the method of Exponentially Weighted Moving Averaging (EWMA) [8], the rate for the next interval $t+1$ can be predicted given the actual arrival rate $\lambda_{ij}(t)$ and the predicted rate $\lambda_{ij}^{[e]}(t)$ of the last time interval t, such as,

$$\lambda_{ij}^{[e]}(t+1) = \alpha\lambda_{ij}(t) + (1-\alpha)\lambda_{ij}^{[e]}(t) \ , \qquad (12)$$

where $\alpha \in (0, 1)$ is a parameter determining the tradeoff between the most recent information and history information. If α is larger, it means we give more weight to the most recent load in the prediction; otherwise, the impact of the most recent load is less emphasized. The prediction accuracy can impact the service quality at each VM in the next interval, i.e, average response time. The problem exists in any other schemes relying on the prediction. A common solution is to allocate resources more than necessary, which introduces the tradeoff between the cost of the extra resources allocated and the possible sacrificing of user experience due to the unpredicted sharp traffic surge. In our scheme, we can adjust μ in (3) to make this tradeoff.

At the end of each time interval, the request rate matrix from the prediction is obtained and then the proposed algorithm *OLS* or *TLS* can be applied. Based on its solution, the amount of the VM and storage resource allocated, and the data location will be adjusted accordingly. The deployment is feasible in the IaaS clouds, since users of IaaS clouds can decide how to connect the rented node logically and to place a data object in which node by their own choice. In our work, the primary objective is to reduce the monetary cost of the system and the main constraint is to match the service capacity and throughput of each VM node. That could be well achieved through the proposed scheme.

During each time interval, the access controller (or the cluster of homogenous access controllers) is responsible for routing each request (i, j) to the proper VM. It is necessary that each access controller maintains a hash table with the size of $I \times J$. Using (i, j) as a key, the resultant hashing value through that table indicates the routing destination. The table is built based on the solution of the algorithm. Its size is controllable because of the granularity coarsening process introduced in Section II.A, where we can adjust the number of accounted user groups and content groups to an acceptable level. This avoids the expensive overhead to obtain the optimal solution and provides a good support to the large scale system.

V. Performance Evaluation

A. Data Preparation

NASA-HTTP Trace [9]: This is the user request trace of a NASA website, where 1,891,714 HTTP requests are captured in a period of 28 days. For each request, the client address (IP or domain name), content object to fetch, and timestamp are given. According to our design, in the first step, we cluster all the client addresses into user groups and all the requested contents into content groups. The grouping is by a simple hash function and results in 101 grouped users and 101 grouped contents. Besides, the arrival rate of each request in a given interval can be counted from the request log. Because the traffic intensity in the dataset is still too low to use multiple serving machines, we change the granularity of timestamps in the trace from 1 to $1/10,000$, for the purpose of having a large enough traffic intensity in the simulation.

Generated Data: To give more insights of the algorithms to be evaluated, we analyze the distribution of requests to different contents in the NASA-HTTP dataset, and use it to flexibily generate different experiment data under a given traffic intensity but still following the same distribution. The distribution from the trace is shown in Fig. 3. We found that it fits the Zipf distribution with index H = 0.65, so we use the Zipf distribution to generate the random data. It was justified that the overall user demand distribution to a large set always shows the long-tail behaviour and Zipf can be used to approximate it [10]. A recent measurement also justifies that the user requests to YouTube satisfy the Zipf distribution [11]. Note that although the NASA-HTTP trace for webpages is used in the experiments, the methods still apply in other scenarios. We introduce the trace only to show the validity of synthesizing the request pattern based on the Zipf distribution.

In the synthesized dataset, the numbers of user and groups considered are set to 200 and 500, respectively. Note that they are the number of groups or shards after granularity coarsening and the number of actual individuals could be much larger. The request rate matrix A, modeled in Section II.B, is generated with the following steps: 1) the average request rate of all the requests is given at first, denoted by $\overline{\lambda}$; 2) we create 10 user patterns, and for each pattern, the request rate to each content follows the Zipf distribution; for each pattern, the content set is shuffled into a random sequence, and then the request probability to the c-th content is determined by the Zipf with $H = 0.65$, such that $P(c) = (1/c^H)/\sum_{j\in F}(1/j^H)$; then the request rate is obtained as $P(c)\overline{\lambda}J$; 3) each user is randomly assigned a pattern, and adopts the request rate of that pattern.

B. Evaluation Settings

The simulations are based on a program written in JAVA, implementing the algorithms proposed. The inputs of the algorithm are the extracted or synthesized request rate. The outputs are the decisions on the resource allocation, data location and request handling. The system costs are used as the metrics in the evaluation, where the costs per unit of resources (VM or storage) are estimated based on the prices of Amazon Web Services [12]. The numbers of allocated units of resources are from the output of the algorithm. For the verification of the simulation, we reviewed the outputs and validated that the deployment based on them satisfies the constraints in the modeling.

The two algorithms proposed in the paper are implemented and evaluated, i.e., `OLS` and `TLS`. Besides, a baseline approach is implemented and compared, which is the Decreasing First-Fit (`DFF`). In this method, the requests are first sorted decreasingly by request rate and then each request type (i, j) is greedily inserted into the first-fit bin in the order after sorting. It can be considered as a basic improvement to the standard practice of distributed storage systems (e.g., Cassandra), where key-value pairs are randomly distributed to the available nodes. The algorithms are first evaluated through the NASA dataset, and then extensively checked and compared under different parameters.

The service rate of each VM node is set to 500 requests/sec. The unit monetary cost in the evaluation is based on the price of Amazon Web Services [12]. For the VM part, the unit price of a VM is estimated as \$0.060/hour×720hours/month = \$43.2/month. For the storage part, we assume that the size of each storage shard is in the range of $(1 \sim 10)$GB, and then the unit price of storage is estimated as (\$0.1/month/GB + \$0.1/million IO×1 million IO/month/GB) × $(1 \sim 10)$GB= (\$0.2 $\sim$ \$2)/month. Base on our estimation on the average size of storage units and the average I/O operation number related to each storage unit, we set the ratio of cost components $C_v : C_u : C_c$ to $50 : 1 : 1$ by default in the simulations. The exact ratio varies depending on the actual shard size, I/O operations, and cloud pricing policy, so the discussions on different ratios are made to validate the performance of algorithms in a broader range of cost ratio.

C. Evaluation Results

Remind that a request rate matrix was extracted from the NASA-HTTP dataset. Based on the matrix, the three algorithms are applied and their comparative results are shown in Fig. 4. In terms of the total cost, `OLS` and `TLS` both outperform `DFF`, the baseline. Then we scrutinize the cost by comparing the three types of cost respectively, which helps us to analyze the root cause of the difference in the total cost. `OLS` is similar to `DFF` in the VM cost, but incurs much lower storage cost. This is because the cost-effective data placement in the former. We also implemented the *RARH* problem in the one-hop case with the optimization tool Gurobi [13], and tried to obtain the optimal solution for the same input as in Fig. 4. But it failed to give the solution in an acceptable time, because of the scale of the optimization problem. For a further simplified input with only 11 users and 11 contents, the best solution from Gurobi in 1 hour has a total cost of $1,880$ while `OLS` can instantly give its solution with a total cost of $1,960$ for the same input.

To have more insights on the algorithms, with the generated data, a set of experiments with different average request rates is conducted, which range from 1 to 10 requests/sec. The number of user-type or content-type objects that need to be stored by each algorithm is shown in Fig. 5. It can be observed that `TLS` gives the best performance in the storage cost, since it sacrifices the cost of VM. `OLS` has a higher total storage cost than `TLS` but is still much better than `DFF`, which validates that the partition-based packing method can largely reduce the storage cost. We also observe that the costs of user-type and content-type storage are almost the same in `OLS`, which is related to the fact that $C_u : C_c$ is set to $1 : 1$ in this experiment.

Then the cost on the VM is shown in Fig. 6. We observe that `OLS` has slightly higher cost than `DFF` because the partition-based packing reduces the storage cost, but it simultaneously results in some small and wasted fragments in the VM service capacity. Besides, `TLS` costs almost twice of `OLS`. It is because the two-layer serving design introduces more traffic load to the system. In fact, under the two-hop setting, each request from

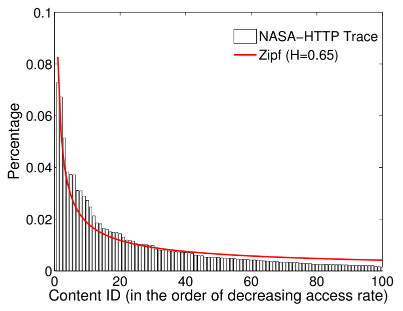

Fig. 3. Content request rate distribution in the NASA-HTTP trace

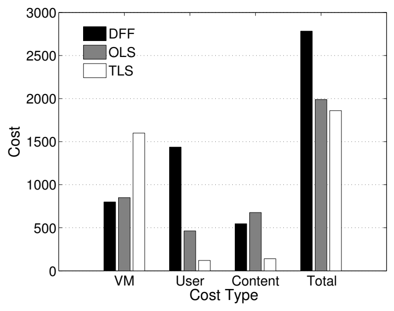

Fig. 4. Comparison results for the trace

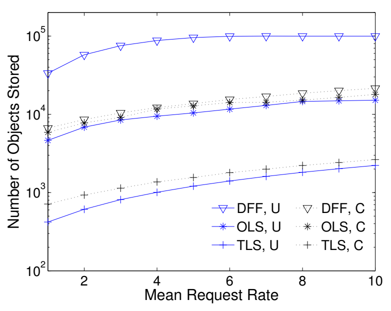

Fig. 5. Comparison results of storage cost

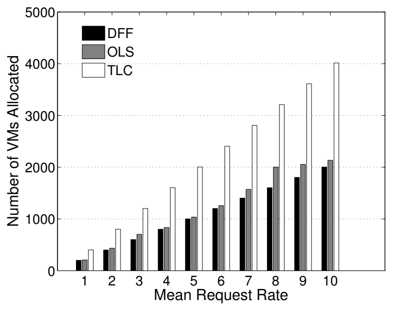

Fig. 6. Comparison results of VM cost

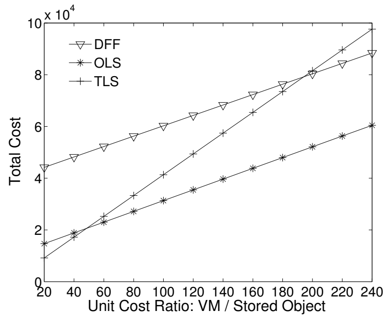

Fig. 7. Effect of cost ratio $C_v : C_u$

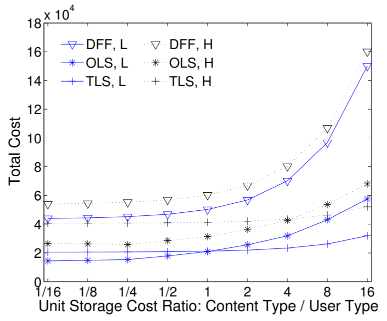

Fig. 8. Effect of cost ratio $C_u : C_c$

the user is fulfilled only after it goes through two nodes in the system, one for the user profiling and another for the requested content fetching. Comparatively, OLS or DFF shows a great advantage on the VM cost and access efficiency because they fulfill the request through only one node.

Besides, the total cost under different cost ratios is discussed. In Fig. 7, $C_u = C_c = 1$ is set and we change the value of C_v to check the resultant total cost. It could be observed when the ratio is small, TLS could outperform the others, since its aggressive allocation of VMs largely reduces the demand for data replication and therefore lowers the cost of storage. But if the unit VM cost increases, the advantage disappears soon. Then OLS is always better than DFF in all the ratios tested, since the former moderately sacrifices the VM cost to achieve in return the large reduction in the storage cost. From the observations here, we suggest that TLS is more suitable for the application scenarios where exceptionally large objects are requested, resulting in a much lower ratio of $C_v : C_u$; otherwise, OLS is better to be applied.

Then we set $C_c = 1$, and set C_v to 50 or 100 respectively, termed as L and H in Fig. 8, and then C_u is varied. The two proposed algorithms still outperform DFF in all the settings. Besides, we notice when the ratio is high, the cost of TLS can still keep stable, because it relies more on the VMs than the storage and therefore it is less affected by the change of unit storage cost. Besides, when C_u is larger, it implicitly results in a smaller ratio of $C_v : C_u$, which lets the TLS get close to or even outperform OLS, for the same reason stated above. This informs us that the ratio of C_v to the larger one of C_u and C_c should be paid enough attention, when we are to decide which is more suitable for a specific application scenario between the two proposed algorithms. We also validate that the advantage of the proposed algorithms holds in all the range of cost ratios in the experiments.

VI. Related Work

We investigated the optimization of a content retrieval system through exploiting the predictability of traffic load. Similar problems commonly exist in Content Distribution Networks (CDN). In [14], the network is modeled as a weighted graph to consider the distance between nodes, and the problem to minimize the cost of replica storage and retrieval is formulated and several heuristics are proposed. In [15], both the replica placement and caching are considered to improve the user experience in the HTTP services. In its proposed hybrid scheme, the storage in a node is shared by the replicator and caching. Karlsson et al. [16] stated that any specific heuristic might only ensure certain performance metrics it is designed for, and suggested the methods in determining the applicability of any specific placement heuristic. The similarity of our work and CDN is that they both consider how to serve user requests through a certain amount of distributed machines. And the difference is that our work did not consider the geo-distribution of these machines. Instead, we differentiate the types of data stored in the network, considers the impact of traffic load in making data replica decisions and investigate the cost-efficient design method under the cloud-based cost model, which is not discussed in CDN. The method proposed could be a supplement to improving the system efficiency of CDN in one data center.

With the convenience and potential cost reduction of deploying services to the cloud, the study on the data placement or service placement in the cloud emerges. Alicherry et al. [17] considered two types of entities in IaaS clouds, storage and

VM, and they focused on optimizing the total (or maximum) access latency by the proper mapping between the given set of VM nodes and set of storage nodes from the perspective of cloud provider. Our work considers how to utilize those nodes from the prospective of cloud users and solves the problems of data placement and request handling routine. Argarwal et al. [7] considered the geo-distributed clouds. By collecting user logs in different data centers, a centralized decision is made to guide each data object to move to a weighted center of user requests in order to lower the access latency. Our work aims at the optimization in one data center and considers the benefit and cost of data co-location. In [2], it is suggested that the data object placement in a cloud-based OSN system should consider the social relationship between users. Rochman et al. [1] proposed a high-level resource assignment and placement problem to satisfy certain service demands with clouds while minimizing the service cost. With a predefined assignment scheme, the placement problem was solved by transforming it into the min-cost flow problem. All such works are not designed for the user-aware content retrieval services, such as having respective models and objectives or not capturing the user demand of the specific service.

The formulated problem in the paper is related to bin-packing [3], but with some extra objectives, therefore we cannot obtain a favorable solution by the common bin-packing algorithms. In [18], a different extra objective is considered in the two-dimensional bin-packing, i.e., to maintain a balanced load of each bin. Xavier et al. [19] modeled the different items in each bin as an extra constraint and a moving-window based heuristic is proposed. However, our problem is different such that the number of different objects in a node has no upper limit, although it is expected to be low. In [20], bin-packing is used in the VM placement problem, whose motivation is how to efficiently pack the VMs into physical machines with multiple types of resource constraints. Our work can potentially be generalized as a new variant of the bin-packing problems and be more broadly applied to some different scenarios in the future.

VII. Conclusions

In this paper, we investigated and formulated the problem of optimizing the resource allocation and request handling for the user-aware content retrieval services. Two approximate algorithms with different design emphasis were proposed and the approximation ratios were derived. Besides, the issues in implementing the algorithms in practice were discussed. Through simulations, the performance of the devised algorithms was evaluated, showing a large improvement in the total cost while ensuring the service quality.

In the future, we will improve the work with the considerations of data caching and adaptability to the dynamic workload, in order to achieve a lower system cost in the whole running period. In fact, the application of the proposed methods can be extended to the scenarios which require to place data items from two different categories while aiming at a higher access efficiency or storage efficiency. Following the work, we will also try to extend the problem with a more generalized formulation for the broader spectrum of applications, such as allowing a higher number of data items involved in each transaction.

Acknowledgment

This work is supported in part by NSERC, CFI and BCKDF.

References

[1] Y. Rochman, H. Levy, and E. Brosh, "Resource placement and assignment in distributed network topologies," in *Proc. of IEEE INFOCOM*, 2013.

[2] L. Jiao, J. Li, T. Xu, and X. Fu, "Cost optimization for online social networks on geo-distributed clouds," in *IEEE ICNP*, 2012.

[3] E. G. Coffman Jr, M. R. Garey, and D. S. Johnson, "Approximation algorithms for bin packing: A survey," *Approximation algorithms for NP-hard problems*, pp. 46–93, 1996.

[4] H. Zheng and X. Tang, "On server provisioning for distributed interactive applications," in *IEEE ICDCS*, 2013.

[5] Q. T. I. Adan and J. Resing, *Queueing Theory*. Eindhoven University of Technology, 2002.

[6] G. L. Nemhauser and L. A. Wolsey, *Integer and combinatorial optimization*. Wiley New York, vol. 18, 1998.

[7] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services." in *Proc. of USENIX NSDI*, 2010.

[8] J. S. Hunter, "The exponentially weighted moving average." *Journal of Quality Technology*, vol. 18, no. 4, pp. 203–210, 1986.

[9] Online, "http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html."

[10] L. A. Adamic and B. A. Huberman, "Zipf's law and the internet," *Glottometrics*, vol. 3, no. 1, pp. 143–150, 2002.

[11] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "YouTube traffic characterization: a view from the edge," in *Proc. of ACM IMC*, 2007.

[12] Online, "http://aws.amazon.com/ec2/pricing/."

[13] Online, "http://www.gurobi.com/."

[14] X. Tang and J. Xu, "On replica placement for qos-aware content distribution," in *Proc. of IEEE INFOCOM*, 2004.

[15] S. Bakiras and T. Loukopoulos, "Combining replica placement and caching techniques in content distribution networks," *Computer Communications*, vol. 28, no. 9, pp. 1062–1073, 2005.

[16] M. Karlsson and C. Karamanolis, "Choosing replica placement heuristics for wide-area systems," in *Proc. of IEEE ICDCS*, 2004.

[17] M. Alicherry and T. Lakshman, "Optimizing data access latencies in cloud systems by intelligent virtual machine placement," in *Proc. of IEEE INFOCOM*, 2013.

[18] D. Liu, K. C. Tan, S. Huang, C. K. Goh, and W. K. Ho, "On solving multiobjective bin packing problems using evolutionary particle swarm optimization," *European Journal of Operational Research*, vol. 190, no. 2, pp. 357–382, 2008.

[19] E. C. Xavier and F. K. Miyazawa, "The class constrained bin packing problem with applications to video-on-demand," *Theoretical Computer Science*, vol. 393, no. 1, pp. 240–259, 2008.

[20] J. Xu and J. A. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Proc. of IEEE GreenCom*, 2010.