Defragmentation of Resources in Virtual Desktop Clouds for Cost-Aware Utility-Optimal Allocation

Mukundan Sridharan, Prasad Calyam, Aishwarya Venkataraman, Alex Berryman
Ohio Supercomputer Center/OARnet, The Ohio State University, USA; Email: {*sridhara*, *pcalyam*, *venkatar*, *berryman*}*@oar.net*

***Abstract*—Cloud Service Providers (CSPs) make virtual desktop cloud (VDC) resource *provisioning* decisions within desktop pools based on user groups and their application profiles. Such provisioning is aimed to satisfy acceptable user quality of experience (QoE) levels and is coupled with subsequent *placement* of VDs across distributed data centers. The placement decisions are influenced by session latency, load balancing and operation cost constraints. In this paper, we identify the resource fragmentation problem that occurs when placement is done opportunistically to minimize provisioning time and deliver satisfactory user QoE. To solve this problem, which inherently is an NP-Hard problem, we propose a defragmentation scheme that has fast convergence time and has three levels of complexity: (i) "utility fair provisioning" (UFP) to optimize resource provisioning within a data center - to achieve relative fairness between desktop pools, (ii) "static migration-free utility optimal placement and provisioning" (MUPP) to optimize resource provisioning between multiple data centers - to improve performance, and (iii) "dynamic global utility optimal placement and provisioning" (GUPP) to optimize resource provisioning using cost-aware and utility-maximal VD re-allocations and migrations - to increase scalability. We evaluate our defragmentation scheme against 'least latency', 'least load', and 'least cost' schemes using a novel "VDC-Sim" simulator that we have developed in this study. Our simulations leverage profiles of user groups and their applications within desktop pools, obtained from a real VDC testbed. Our simulation results demonstrate that defragmentation is an important optimization step that can enable CSPs to achieve fairness, substantially improve user QoE and increase VDC scalability.**

***Keywords*-Virtual Desktop Clouds, Resource Defragmentation, Optimal Resource Allocation, Greedy Heuristic**

I. Introduction

Common user applications such as email, photos, videos and file storage are already being supported at Internet-scale by 'cloud' platforms (e.g., Amazon S3, Google Mail, and Microsoft Azure). Even academia is increasingly adopting cloud infrastructures and related research themes (e.g., NSF CluE, DOE Magellan) to support desktop delivery for various science communities. The next frontier for these user communities will be to transition 'traditional desktops' that have dedicated hardware and software installations into 'virtual desktop clouds' (VDCs) that are accessible via thin-clients. The drivers for this transition are obvious and include: (i) desktop support in terms of operating system, application and security upgrades will be easier to manage centrally, (ii) the number of underutilized distributed desktops unnecessarily consuming power will be reduced, (iii) mobile users will have wider access to their applications and data, and (iv) data security will be improved because confidential user data does not physically reside on thin-clients. VDC platform providers are beginning to host virtual desktops by augmenting domain-specific community clouds (e.g., education clouds), and are customizing desktops for these communities through pay-as-you-go services such as: Desktop-as-a-Service (DaaS), Infrastructure-as-a-Service (IaaS), and Software-as-a-Service (SaaS).

This material is based upon work supported by the National Science Foundation under award number CNS-1050225, VMware, and Dell. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

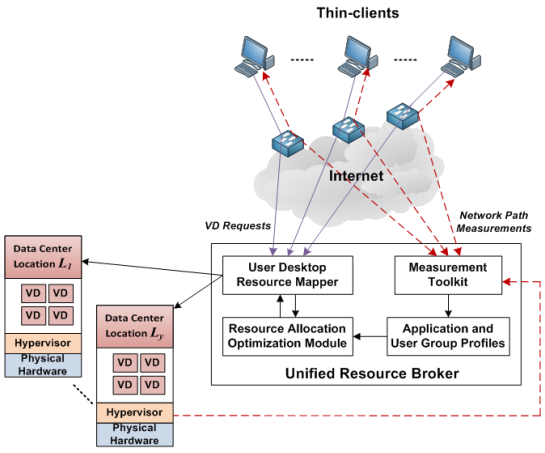

Fig. 1. Components of Virtual Desktop Cloud Infrastructure

Figure 1 shows the various system and network components in a typical VDC, comprising of two or more inter-connected data centers that handle thin-client connections from several user sites on the Internet. At each data center, a hypervisor framework (e.g., VMware ESXi, OpenVZ, Xen) is used to create virtual desktops (VDs) that host popular applications (e.g., Excel, Internet Explorer, Media Player) as well as advanced scientific computing applications (e.g., Matlab, Moldflow). A Unified Resource Broker (URB) receives the VD requests and handles them by making decisions regarding the *placement* (i.e., resource mapping) and *provisioning* (i.e., resource sizing) of resources.

The objective of the URB is to ensure user QoE is satisfied while maintaining scalability of the VDC in terms of the overall number of VDs handled. Obviously, increased scalability leads to benefits from economies of scale such as increased revenue for the CSP, and potentially lower cost per VD. To perform provisioning of VDs, the VDC relies on information regarding user groups and their application profiles that are obtained via offline profiling using measurement toolkits such as VDBench [2]. The profiles allow 'desktop pooling' corresponding to the resource requirements of applications in different user groups. They also indicate the amount of minimum-and-maximum resources to be provisioned to satisfy Service Level Agreements (SLAs) between CSP and users; an SLA for instance can specify the 'least tolerable' interaction response times (with minimum resource amount) when accessing a document editor via a thin-client, and the corresponding 'best' interaction response time (with maximum resource amount

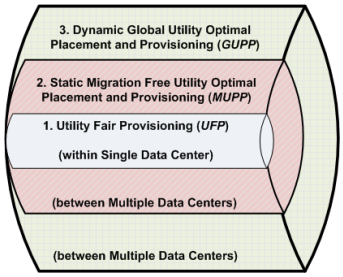

Fig. 2. Defragmentation Scheme Complexity Layers

beyond which user does not perceive any more improved QoE). Although Figure 1 shows the URB as a single entity, its implementation could infact be a set of distributed entities via secure state-sharing protocols commonly used in content-delivery networks.

For every VD request to be handled in the VDC, the URB has to maintain "fairness" i.e., it has to maintain the relatively same utility [1] between desktop pools to avoid unfair overprovisioning of any particular user group based on its application profile. In addition, given the fact that VD resource reallocations and VD migrations are expensive operations in a VDC, the URB may not always able to place a VD in a 'globally optimal' manner. In such cases, the URB opportunistically assigns a new VD request to a data center that has atleast the minimum resources for SLA compliance, and that satisfies the thin-client session latency, load balancing and operation cost constraints. Over time, such opportunistic placements by the URB results in a 'resource fragmentation' problem that is analogous to the 'disk fragmentation' problem that limits performance in personal computers. In the VDC context, the resource fragmentation affects both user QoE and scalability as can be seen in the following two example cases: i) users may be connected to VDs that are allocated only the minimum resources when additional resources are available at a different data center; (ii) VDs may be rejected if the only data center that satisfies the placement constraints is overloaded and there are other lightly loaded data centers that can handle the already placed VDs in the overloaded data center. In both these cases, 'defragmentation' using a 'global' optimization involving VD resource reallocations and VD migrations can improve user QoE and increase scalability respectively. However, it is important to note that the global optimization needs to be triggered at time points where, the net benefit from improved user QoE and increased scalability exceed the cost of 'global' optimization that involves VD resource reallocations and VD migrations.

In this paper, we provide an optimal solution for the above resource fragmentation problem, which inherently is an NP-Hard problem, by developing a defragmentation scheme that has fast convergence time and has three levels of complexity shown in Figure 2. The UFP maintains relatively same utility between desktop pools with different application profiles of user groups. The MUPP is an online algorithm that opportunistically maximizes the utility between multiple data centers and allows quick provisioning times for VDs upon new user request arrivals. In contrast, the GUPP is an offline algorithm that globally optimizes the utility between multiple data centers by minimizing 'utility loss', when the 'net benefit' from improved user QoE and increased scalability exceeds the operational cost of 'global' optimization. We evaluate our defragmentation scheme against 'least latency', 'least load', and 'least cost' schemes using a novel "VDC-Sim" simulator that we have developed in this study. The simulator enables visual analysis of VD placements across distributed data centers and allows analysis of the achieved "utility loss" minimization and "net benefit" maximization levels for various solution configurations in fragmented VDCs. Our simulations leverage profiles of user groups and their applications obtained from a real VDC testbed that we developed in our recent prior study [3]. Our simulation results demonstrate that defragmentation is an important optimization step that can enable CSPs to substantially improve user QoE and VDC scalability.

The remainder of this paper is organized as follows: In Section II, we describe prior literature related to our work. In Section III, we formally describe the defragmentation solution objective. In Section IV, we describe our UFP and MUPP algorithms. In Section V, we describe our GUPP algorithm. In Section VI, we first present our simulation setup in VDC-Sim and then present validation results of our algorithms. Section VII concludes the paper.

II. Related Work

Resource fragmentation problems in storage and memory systems have been extensively studied in prior literature. Given that the basic problem is combinatorial in nature, earlier works have developed heuristics that iteratively decrease the number of fragments. We extend such an approach to the resource fragmentation problem in VDCs. The work in [6] is closely related to our work; the authors illustrate the importance of matching homogeneous workloads of high-performance computing tasks, while placing them at distributed server locations. They claim that the problem solution is similar to that in a tetris game, where objects of different shapes need to be packed in a box. A careless packing of workloads on servers will create huge capacity holes, resulting in sub-optimal resource usage—which they call the *tetris effect*. In comparison, our workloads are heterogeneous i.e., VDs have varying application profiles and the provisioning as well as placement constraints are more involved due to the nature of thin-client protocols that are sensitive to latency, loss and available bandwidth [2]. Further, we actually solve the fragmentation problem using optimization techniques, whereas authors in [6] merely illuminate the problem. In [5], the authors recommend a unified broker architecture for resource allocations similar to our approach. They also describe how unified resource brokers can perform functions such as performance isolation, live migration and suspend/resume capabilities.

In terms of comparison of optimization algorithms in prior literature for related problems, authors in [7] makes resource allocations based on the network topology and application requirements for data-intensive workloads using genetic search algorithms. In [4], the authors develop a fair resource allocation scheme, which is round-robin across data centers based on resource dominance levels. In [11], the authors consider the problem of SLA-based resource allocation for cloud data centers. While they consider placement directed by utility of applications, they do not factor in the cost of migration in the optimization algorithms. In our previous work on utility-directed resource allocation model (U-RAM) [3], we developed an iterative algorithm that addresses *only resource provisioning* issues in VDCs based on the theoretical underpinnings

[1] A utility function indicates how much of application performance can be increased with larger resource allocation. Beyond a certain point, application utility saturates and any additional resource allocation fails to further increase application performance.

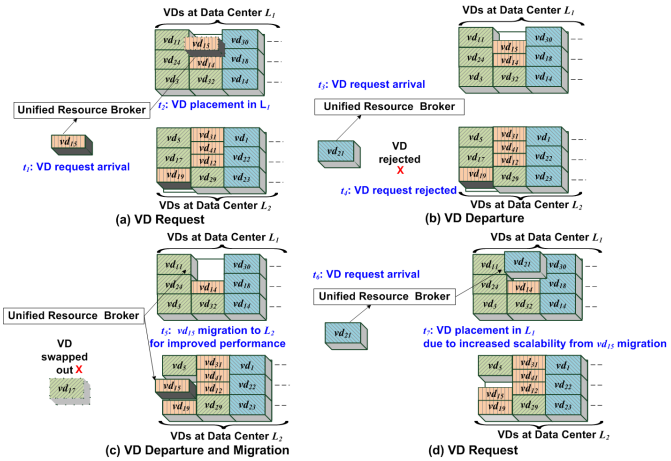

Fig. 3. Resource fragmentation in VDC

in works such as [8], [9]. In this work, we extend our earlier U-RAM work and propose greedy heuristics that addresses *both placement and provisioning* of VDs within VDCs. The novelty is in the fact that complexity of the algorithms in this paper increase in steps to improve fairness, performance and scalability in VDCs and also take into consideration a cost-benefit analysis to determine reallocations and migrations of VDs between multiple data centers.

III. Resource Fragmentation Problem in VDCs

A. Problem Overview

Given the fact that CSPs manage VDs at multiple data center sites, they need to consider the following factors as new VD requests arrive: (a) session latency, (b) operation cost, and (c) resource availability, all while delivering satisfactory user-perceived QoE. Based up on session latency and operation cost factors, VDs are placed at a location that will yield maximum benefit under the constraint of resource availability. Figure 3(a) shows a scenario when a VD request best suited for maximizing QoE at Location L_2 is allocated to Location L_1 due resource unavailability at L_2. This creates a situation where the placement is sub-optimal. Also, Figure 3(b) shows a scenario where a VD request is rejected due to lack of resources at both L_1 and L_2. Figure 3(c) shows a scenario where one of the VDs already placed at L_2 departs, creating a resource 'hole' or 'fragment'. Several fragments created by such departures lead to the *resource fragmentation problem* that affects VDC scalability and user QoE. In the above example scenario, we solve this problem by migrating v_{12} that was originally placed at L_1. In Figure 3(d), since vd_{12} migrated to L_2, defragmented resources are now available at L_1 to accept the earlier rejected request vd_{21}. The four scenarios together illustrate the resource fragmentation problem and our solution that involves migrating VDs to new locations in order to optimize the system utility. We remark that migration of a VD is an expensive process, and we consider the cost and benefits before making migration decisions in our solution.

The solution to the above broad problem is the goal of this paper. In general, this is a combinatorial problem where optimal allocation depends on the order of the VD arrival, and is a directed instance of the Generalized Assignment Problem [10]. GAP problems typically require complex solutions, and even generic or greedy solutions are computationally intensive. Hence, we are interested in finding 'near-optimal' solutions that are practical and quick to implement, rather than an optimal computationally intensive solution.

B. Utility Definition

For a VD vd_i that is allocated resources $R^i = \{r_{i1}\ ,\ r_{i\ 2},\ ...,\ r_{iJ}\}$ at data center L_l, the utility generated is U_i^l, the composite quality generated is Qc_i and the network latency of the VD client to data center L_l is vl_{il}. U_i^l is a function of the composite quality Qc_i and the latency vl_{il} of the VD to the data center L_l i.e., $U_i^l = Qc_i * f(vl_{il})$; Qc_i is a weighted function of the qualities generated along D quality dimensions and can be given as - $Qc_i = \sum_{d=1}^{D} w_{id} * q_d$.

C. Defragmentation Solution Objective

Formally, the defragmentation i.e., solution objective can be stated as follows: Given a list of N VDs, L data centers each of which has J type of resources, where $R_1, .., R_J$ is the total resource of each type, given a provisioning PR_k and a placement P_k for the VDs and the total system utility U_k, find an optimal provisioning PR_o and placement P_o, such that "net benefit" generated is maximum. Net benefit is defined as the difference between the gain in total system utility and the cost involved in migrating VD from the current placement and provisioning configuration to a new configuration. The maximization of net benefit is subject to: (i) *resource constraints* i.e., at any location, total resource of any type allocated to VDs does not exceed the resource capacity at that location, (ii) *quality constraints* i.e., every VD request is ensured a quality that at least satisfies the SLA, and (iii) *fairness constraints* i.e., the composite qualities of all user groups at any location are equal.

$$
\begin{aligned}
&Given: \\
&\quad P_k, PR_k \text{ and } U_k \\
&Maximize: \\
&\quad NetBenefit_{ko} = (U_k - U_o) - \ MC_{ko} \qquad (1) \\
&Subject\ to: \\
&\quad Resource\ Constraint: \sum_{\{i:v_i \epsilon S_l\}} r_{ij} \leq R_{jl},\ \forall\ \{j,l\} \\
&\quad Quality\ Constraint: \ Qc_i^{min} \leq Qc_i \\
&\quad Fairness\ Constraint: \ \forall\ \{i,k\}\ Qc_i = Qc_k
\end{aligned}
$$

D. Defragmentation Solution Methodology

The defragmentation solution characteristics is combinatorial in nature and an exhaustive search approach would be intractable. In order to solve the resource fragmentation problem in an efficient and practical way, we develop a defragmentation scheme that has fast convergence time and has three levels of complexity shown in Figure 2: (i) "utility fair provisioning" (UFP) to optimize resource provisioning within a data center - to achieve relative fairness between desktop pools, (ii) "static migration-free utility optimal placement and provisioning" (MUPP) algorithm to optimize resource provisioning between multiple data centers without affecting previous placements - to improve performance, and (iii) "dynamic global utility optimal placement and provisioning" (GUPP) algorithm to optimize resource provisioning using cost-aware and utility-maximal VD re-allocations and migrations - to increase scalability.

IV. Migration-free Utility-Optimal Placement & Provisioning Algorithm

We first explain how we make utility-optimal fair Resource provisioning (UFP) at a single data center, such that - given a set of VDs to be placed at the data center, we make optimal resource provisioning to achieve fairness. Next, we explain how we use the UFP in the MUPP algorithm to make optimal placement and provisioning across multiple data centers without reallocations and migrations. In MUPP, we focus on finding an optimal placement for the new arriving VDs, without disturbing the locations of any of the previously placed VDs in the VDC.

A. Utility-optimal Fair resource Provisioning Algorithm (UFP) for a Single Data Center

The UFP algorithm consists of: i) "offline" utility function characterization for 'user groups', and ii) "online" optimal and fair resource provisioning. We group VDs of similar usage profiles into distinct 'user groups' and provision the same amount of resources to all VDs in a user group. To collect measurements for characterizing the utility functions, we leverage a virtual desktop performance benchmarking toolkit viz., "VDBench" that we developed in [2]. The VDBench uses a novel methodology that allows correlation of thin-client user events with server-side resource performance events by virtue of 'marker packets'. In this paper, we use our VDBench toolkit to obtain user application and user group profiles in an offline manner, which can then be leveraged online to optimize resource mapping and sizing in a VDC under real user loads.

UFP handles resource allocation fairness based on user's perceived QoE considerations. More specifically, resources are allocated such that all VDs in a data center have the same relative user QoE. Maintaining same quality level across user groups may not necessarily require equal allocation of resources, as applications in one user group may require more/less resources than in the other user groups. We define a "fairness index" metric that quantifies the quality difference between two VDs at a data center site. If $\delta(x,\ y)$ defines difference in quality level of VD x and y and $x\ \neq\ y$, then -

$$f = 1 - Max(\delta(i,j)) \qquad \forall i,j\ and\ i \neq j \tag{2}$$

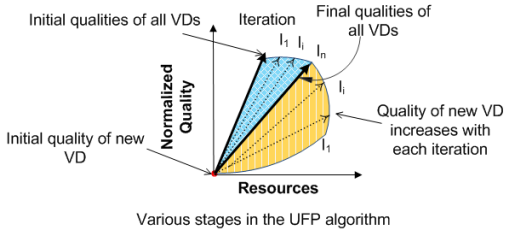

Fig. 4. Illustration of Iterative UFP algorithm

UFP aims at maximizing utility which is modeled upon user QoE. The utility-optimal resource provision problem presented in this paper is an instance of *Bounded Knap-Sack problem*, and is known to be NP-hard in number of VDs. However, since we profiled the VDs in to a smaller set of 'user groups', our resource provisioning is simplified, and the running time decreases significantly. Using this observation, we now describe the design an iterative algorithm that has fast convergence: When a new VD arrives, we allocate resources to the new VD from the resources already allocated to the VDs of the same user group, which decreases the user group's R_{set} value and decreases its quality represented as Qc_g. Next, we iteratively adjust the resources of the user groups by removing resources from the user group with maximum quality and adding it to the user group with minimum quality, until all of them have the same quality. By adjusting the δr, the amount of resource exchanged between the maximum and minimum quality user groups, based on the difference in quality levels, we make the algorithm achieve fast convergence. Figure 4 explains this provisioning scheme visually.

While resource provisioning on a single resource can be done as explained above, provisioning for multiple resources is much more complex. It is possible to solve the optimality problem across multiple quality dimensions, because we consider quality dimensions that are independent. Quality dimensions are called dependent if increase in one dimension, results in a decrease in the quality along another dimension [8], [9]. The multiple-quality dimensions can be solved if the utility function is modeled as a linear combination of the individual qualities. The resource allocation is solved for individual qualities and then resources are combined using the same linear function (as used to combine the qualities) to come up with the final resource allocation. The problem for multiple resource dimensions can be solved, if the *maximal majorant* of quality versus resource mapping are known. The maximal majorant is a bounding curve along each quality dimension. We measure the maximal majorant for each resource, by setting the other resources at their maximum values, and by measuring the impact of change in quality due to the change in the resources. Measuring the quality this way, decreases the space to be searched for the optimum values, since we know that the optimum has to lie on this maximal curve. Thus, in order to arrive at an optimal fair provisioning, the UFP algorithm is run for each quality and resource separately to get their allocations, and then they are combined linearly to obtain the final resource allocation.

B. Migration-free Utility-optimal Placement and Provisioning Algorithm

The UFP algorithm discussed above makes fair resource provisioning for a given set of VDs at a single data center. We use the UFP algorithm as a building block to solve our MUPP algorithm. In the MUPP problem, we need to place a new VD request at a data center location, such that it maximizes the overall system utility at that instant without any reallocations or migrations. We do this by, running the UFP algorithm for each of the data centers L_l assuming the VD will be placed at L_l, and we pick the data center which maximizes the *change in system utility* for each data center. By maximizing the change in utility, we ensure that the overall system utility is also maximized. The MUPP algorithm is optimal since, the UFP is optimal. In addition, for each VD we pick a data center such that the overall system utility is maximized.

V. Cost-Aware Global Utility-Optimal Placement and Provisioning Algorithm

We divide cost-aware global utility-optimal allocation into two sub-problems: (i) finding the Global Utility-optimal Placement and Provisioning of all VDs such that the global utility is optimized, and (ii) identifying the set of VDs to migrate from the current configuration to the optimal configuration such that the net benefit is maximized. We use a *greedy* heuristic to solve the GUPP problem, since optimal algorithm

Algorithm 1 Migration-free Utility-optimal Placement and Provisioning Algorithm for Multiple Data Centers

Input: Utility functions $U_x(R_x)$ of user groups from offline profiling, list of currently available resources $\{R_1,\ R_2\ ...\ R_m\}$ at l data centers
Output: Resource allocations $R_{i,j}$ for VDs v_i where $i \in \{1\ ...\ n\}$ that will maximize global utility U
begin procedure
for a VD request vd_i belonging to group g **do**
  for each data center L_l **do**
    Cache the current data center utility as $U^{l,old}$
    Calculate R_i^l using the UFP algorithm
    Calculate $U^{l,new}$, the new system utility for L_l
    $U^{l,diff} = U^{l,new} - U^{l,old}$
  end for
  Place = vd_i at dc_k such that,
  $U^{k,diff} = max\ \{\ U^{l,diff}\ \}$
  Allocate R_i^k to vd_i
end for
end procedure

is exponential time in number of VDs N and locations L. Given a set of VDs, there exists at least one ordering that will yield an optimal solution. However, since the combinations of resources, user groups and locations are very large in number, it is difficult to determine the optimal ordering with a brute force approach. Hence, we calculate the maximum system utility possible under ideal conditions i.e., every VD is placed at a location which generates maximum utility and gets resource provisioning that generates best quality in the given configuration. We call this the "upperbound" of the total system utility or just "maximum total system utility". We then find a VD ordering that generates a total system utility that is as close as possible to the maximum total system utility. Since we are trying to maximize system utility by moving as close as possible to the upperbound, we call this as "minimizing utility loss" from the upperbound system utility.

For minimizing utility loss, we first calculate the upperbound of utility for each VD by giving it an optimal resource provisioning and placement. To calculate the optimal resource provisioning, we consider the ideal configuration where resources are not fragmented across different data centers. We pool resources across all the data centers and determine upperbounds for resources allocation for VDs based on their user group profiles. We call these as "resource upperbounds". Assuming VDs are allocated upperbound resource values at each location, we find out the maximum utility that could be generated for each VD. The sum of these values gives the maximum total system utility. Since we take the maximum utility for every VD for the calculation of the upperbound, there cannot be any VD ordering which could generate a total system utility more than the upperbound of the total system utility. Since system resources are limited, this utility upperbound is almost never achieved for all the VDs at once. Details of this algorithm are elaborated as a 3-step process in the following:

1) Determining Resource Upperbounds: We determine the upperbound of resource allocation to each VD R_i^{ub} by first making an allocation using the UFP algorithm by pooling resources across all data centers. This is the optimal allocation if resources are not fragmented across multiple data centers, and thus serves as an upperbound of allocation.

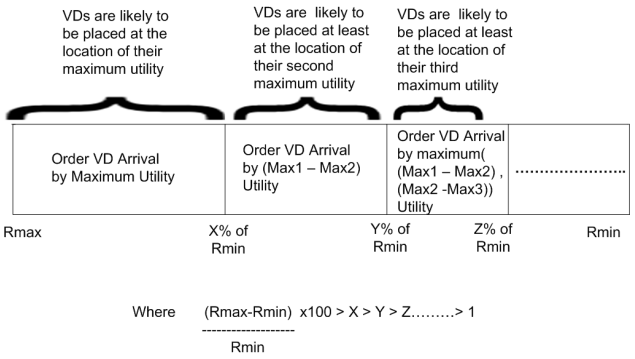

Fig. 5. Generic formulation of the GUPP algorithm for N data centers

2) Determining Utility Upperbounds: Next, we determine the utility that would be generated by each VD at each of the data centers if the upperbound resource allocation R_i^{ub} were to be made available at each data center. This helps us determine the upperbound utility that could be generated by each VD.

3) Finding the right VD arrival order: If we can place all the VDs at locations that generate maximum utility, then that would be the optimum placement which yields maximum total system utility. But, that is not possible in most cases. Hence, our greedy algorithm starts placing VDs one-by-one, at data centers that generate maximum utility. The key insight here is that, there is at least one combination of VD arrival order that can generate the optimum utility. So, instead of dividing VDs into non-overlapping subsets and assigning them in data centers–which is what a number of greedy algorithms do for the GAP problem–we try to find one of the 'optimal' VD arriving orders. We use a metric in our greedy algorithm to minimize the utility loss due to sub-optimal placement. We use the utility upperbounds for each VD at each location to sort the VDs in decreasing order of utility loss due to sub-optimal placement. Figure 5 illustrates this idea. However, the utility loss for each VD depends on where the VD is eventually placed with respect to its maximum utility. Since it is tough to foresee, we make an approximation based on the amount of resources available in the system. Figure 5 shows the progression of the 'utility loss metric calculation' with resource load. Initially we sort VDs based on the maximum utility generated, thereafter we give more importance to the VDs that generate the higher relative system utility. However, when the system is at a point where the 'upperbound' on allocation is x% of the R_{min}, we switch the metric to a 'utility loss' metric, captured by $U_i^{max^1} - U_i^{max^2}$, i.e., the difference between a VD's utility at its maximum location and the second maximum location. Similarly, as we move towards higher and higher resource load regions, we progressively move the utility loss metric to capture more sub optimal locations. Once the VDs are sorted using 'utility-loss' calculations, we use our MUPP algorithm to place them one-by-one.

A. Cost-Aware Migration

The GUPP gives a new set of placement and provisioning for all the VDs in the system. However, all VDs do not necessarily generate a positive benefit upon a migration process, which inherently is an expensive process. We model the cost of migration and normalize it to the utility of the VDs and then select only the VDs that generate positive net-benefit.

Algorithm 2 Global Utility Optimal Placement and Provisioning Algorithm

Input: set of VDs in the system $\{vd_1, ..., vd_n\}$
Output: Resource placement and allocations $R_{i,j}$ for VDs v_i where $i \in \{1 \ ... \ n\}$ that will maximize global utility U
begin procedure
Pool the total resources available at all data centers into a single set of resources $R_1, ..R_J$
Calculate R_i for each vd_i using the UFP algorithm
$RI = \sum_g (R_g^{set} - R_g^{min})/R_g^{min} * 100$
switch (RI)
    case($RI > x$) :
        LU_i^{max} = $U_i^{max^1}$
        break;
    case($RI > y$) :
        LU_i^{max} = ($U_i^{max^1}$-$U_i^{max^2}$)
        break;
    .....
    case($RI > z$) :
        LU_i^{max}= Max$\{U_i^{max^1} - U_i^{max^2}, U_i^{max^2} - U_i^{max^3}, .., U_i^{max^{L-1}} - U_i^{max^L}\}$
end switch
List $vdsToPlace$ =Sort vd_i based on LU_i^{max} in decreasing order
Use MUPP algorithm to make resource allocation for VDs in the order in $vdsToPlace$
end procedure

1) Modeling cost of migration: We model the cost of migration in such a way that the cost of migrating individual VDs can be computed and added together. The cost of migration Mc_i of a VD is split in two three parts as shown in Equation (2): (i) a snapshot cost ($Sc_i(dc_c)$) incurred at the current data center, (ii) a deployment cost ($Dc_i(dc_f)$) incurred at the (future) data center to which a VD is expected to be moved, and (iii) a network cost of moving a VD which might depend on the time of the day. Separating the cost into these three components gives us more flexibility in computing the cost of migration, and performing the cost-benefit analysis to maximize net benefit.

$$Mc_i = Sc_i(dc_c) + Dc_i(dc_f) + Nc_i(t) \quad (3)$$

2) Determining the Maximum Net-Benefit Migration Set: The GUPP algorithm makes optimal allocations assuming all VDs will be migrated to new locations. Since we want to migrate only VDs that increase the utility, we now need to select the VDs carefully such that the resource provisioning made by GUPP does not change. To achieve this, we use an incremental greedy algorithm where we identify pairs of VDs that belong to the same user group whose locations are swapped in the MUPP and GUPP allocations. If we find such pairs of VDs, then these VDs can be migrated (i.e., swapped) without worrying about their resource provisioning, since user groups across data centers more or less receive the same provisioning in GUPP. We call these pairs of VDs *Positive VD Pairs*.

VI. Performance Evaluation

In this section, we evaluate our MUPP and GUPP algorithms using a novel "VDC-Sim" simulator that we have developed in this study. Our simulations leverage profiles of user groups and their applications obtained from a real VDC testbed that we developed in our recent prior study [3].

A. Simulation Setup

User group	CPU(GHz)		Memory(GB)		Bandwidth(Mbps)	
	R_{min}	R_{max}	R_{min}	R_{max}	R_{min}	R_{max}
Engineering Site	0.7	1.5	0.35	1.2	0.2	1.5
Distance Learning	0.3	1.5	0.2	1.1	2	4
Campus Computer Lab	0.5	1.4	0.2	0.8	0.1	1.2

TABLE I
User Group Parameters

In order to evaluate our defragmentation algorithms, we developed an event-driven distributed resource allocation simulator viz., "VDC-Sim" for visually analyzing performance in VDCs. The simulation setup consist of a number of data centers, where each data center consists of a number of servers of fixed capacity. The number of data centers, number of servers per data center and the capacity of each server can be specified using a configuration file. An initialization script generates a 'VD-request-sequence', which is a sequence of allocation and de-allocation requests, belonging to different user groups, generated randomly based on allocation-to-deallocation ratio. Each data center and the thin-client of a VD have physical locations which can be specified as $\{x, y\}$ co-ordinates in a 2D-plane. The VD requests are allocated a physical location randomly in a 2D-plane. The physical locations are used to calculate the distance between data centers and thin-clients and these distances are used to approximate the network latency in the simulations. The main simulation loop takes the VD-request-sequence and handles the events one-by-one. For all the results in this section, we used the following configuration: 3 data centers with 2 servers, where each server has capacity of 16 GHz CPU, 32 GB RAM and 50 Mbps bandwidth. Various placement and provisioning algorithms can be plugged-in to the simulator to study their performance, for the same VD-request-sequence and data center settings. The simulator calculates the utility of each VD and the total system utility and also the keeps track of the resources allocated, the cost of resources, the marginal utility of each VD and also the fairness index.

We simulate VDs belonging to different user groups, with three different resource profiles. Table I shows the R_{min} and R_{max} values of the 3 resources considered for the 3 different user groups. The R_{min} and R_{max} values were obtained in a real-world testbed [1] as explained in Section IV-A. We measure and use two qualities – the *normalized application open times* and the *normalized application response times*, the two quality dimensions that we consider while calculating the VD utilities.

As explained in Section III, we also consider the network latency while calculating the utility of individual VDs. Network latency is an important factor for VD clients performance as shown in [2]. Although, the quality of all user groups decreases as latency increases, the rate of decrease need not be the same for each user group.

B. Algorithms Compared and Results

We compare our UFP algorithm (provisioning algorithm for MUPP) against the U-RAM algorithm from our previous work

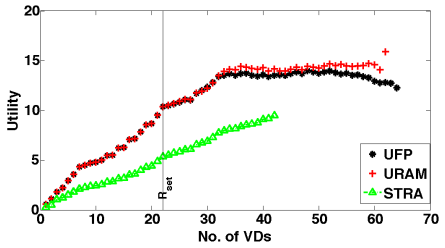

Fig. 6. Utility versus No. of VD request for a Single Data Center

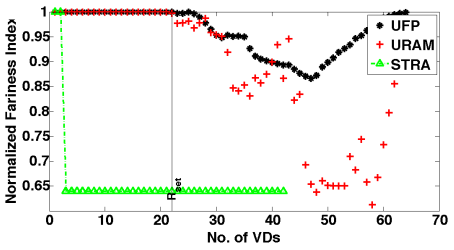

Fig. 7. Fairness Index versus No. of VD request for a Single Data Center

on provisioning [3], and with a static provisiong algorithm called the "Single-Tier Resource Allocation" (STRA). STRA has fixed pre-allocated VDs that it allocates for all user thin-clients. U-RAM is an utility-optimal algorithm for a single site and in general produces equal or more utility for a single site than the UFP algorithm when resources are available. We will use U-RAM as the provisioning algorithm while doing the placement analysis so that we can isolate and study the effect of VD placement on utility. Futher, we compare our MUPP algorithm with 'Least Cost', 'Least Load' and 'Least Latency' placement schemes.

1) Provisioning Analysis: To begin with we present the results of the STRA, U-RAM and UFP algorithms at a single site. Figures 6 and 7 show the utility and the fairness index results for STRA, U-RAM and UFP algorithms. As we can see from Figure 6, STRA either under-allocates or over-allocates resources and hence both its utility and scalability (number of VDs allocated) suffers. U-RAM makes allocations such that it maximizes utility, but does not have any control over the relative qualities between user groups and hence the fairness suffers. The UFP algorithm sacrifices an insignificant amount of utility in order to achieve fairness between the user groups as shown in Figure 7. The fairness of UFP drops slightly in the middle region because of the discrete nature of the quality profiles.

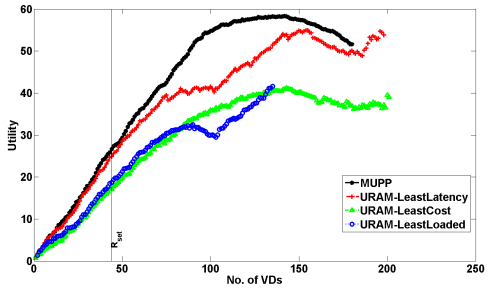

Fig. 8. Comparison of Utility of Placement Schemes for 3 Data Centers

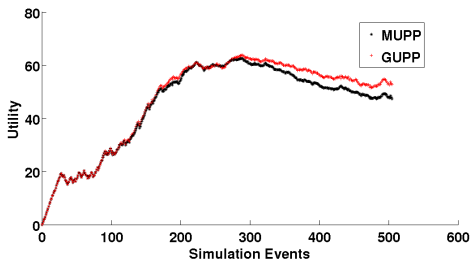

Fig. 9. Utility versus Simulation Instance, 3 Data Centers with Deallocations

2) Placement Analysis: Next we analyze the utility and fairness index of the MUPP algorithm, with different placement algorithms, when placing arriving VDs across 3 data centers, without any deallocations. Figure 8 shows the utility generated by the MUPP and the other placement algorithms. We can see that MUPP achieves the best utility possible, even while guaranteeing quality fairness between the different user groups. The Least Cost has the worst performance in terms of utility since it places all the VDs at the same data center to decrease cost during the initial phase, even though there are abundant resources at the other data centers. The most interesting aspect of the Figure 8 is that even the Least Latency algorithm performs worse than the MUPP, proving our original claim that just placing the VDs at a data center that has the least latency does not result in an optimal placement. The Least Latency algorithm performs similar to MUPP in the region where there are abundant resources, but as resources start to become scarce, it might not be possible to place a VD at a data center with the least latency. In this region, the utility of a VD is affected by both the resources available at a data center and the latency to the data center, and for optimum performance a placement algorithm should consider both these aspects, which is what MUPP does.

3) GUPP and Migration Analysis: In this section we analyze the performance our GUPP algorithm and the Cost-Aware Migration algorithm. First we compare the utility of GUPP algorithm with MUPP. Since we have already shown that the MUPP algorithm performs better the other placement heuristics in the previous discussion, we consider only the performance of MUPP while comparing with the performance of GUPP.

While MUPP is optimal for given order of VD arrival, it is not optimal for a given set of VDs. This is precisely what the GUPP is designed to address. Figure 9 shows a similar comparison for a VD-request-sequence which has both allocations and de-allocations. The difference in utility is seen due to a VD with lower utility value already allocated at a data center, results in a VD of potentially higher utility value being directed to a sub-optimal location. As can be seen in the Figure 9, the gap in the utilities increases as the simulation progresses, since the deallocations increases the 'holes' in the resource allocations and sub-optimal allocations accumulate in the MUPP, while GUPP is able to reallocate VDs to better locations to improve the overall system utility.

Now we explain the migration analysis which is an integral part of GUPP. Figure 10 shows the difference between the locations of VDs between the MUPP and the GUPP allocations. As we explain in Section V-A, even though a large number of VDs are sub-optimally placed in MUPP, it might not be beneficial to migrate all of these VDs to the locations

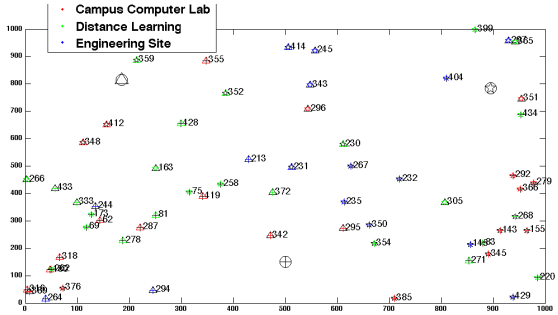

Fig. 10. VDs that are not optimally placed at the end of simulation

Parameters	Campus Lab	Distance Learning	Engineering Site
Total Allocated	185	156	159
Total Deallocated	67	42	58
VDs Suboptimally Placed	22	25	16
Unique Positive Pairs	8	5	4
MUPP Utility Sum	9.9628	9.0074	7.6427
Utility After CA Migration	12.9704	10.4426	8.6823
Cost	0.6272	0.3903	0.3142
Benefit	2.3804	1.0448	0.7254
Percent Increase	30.1889	15.9326	13.6029

TABLE II
SUMMARY OF COST-AWARE MIGRATION ALGORITHM

suggested by GUPP. We need to analyze which of these VDs should be migrated after factoring the cost-of-migration. We use a cost function given by Equation 3 for the cost-benefit analysis. We normalize the snapshot, deployment and the network costs relative to the max utility generated by a VD (which is 1.0). We use a fixed snapshot and a deployment cost of 0.005 units, and a network cost of 0.0001 per unit distance. Network cost is multiplied by the distance between the current and the new location of a VD. As the distance between the data centers increases, the network cost and hence its impact on the migration cost increases. The positive pairs identified for migration result in a net-benefit after migration as shown in Figure 11.

Table II summaries the results of the Cost-Aware Migration algorithm for the same instance as shown in Figure 11, categorized by the 3 user groups Campus Computer Lab, Distance Learning and Engineering Site. We can note that - after calculations, subtracting the cost of migration from the benefit will still have 30%, 15% and 13% increase in the utility of individual groups, and the improvement in the user QoE will

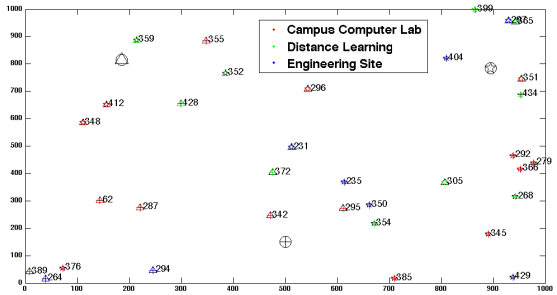

Fig. 11. Positive VD Pairs identified for Migration

be higher. Thus, the Cost-Aware Migration algorithm results in an improved QoE for the user, better resource utilization and scalability for the CSP.

VII. CONCLUSION

In this paper, we identified and provided an optimal solution for the resource fragmentation problem in VDCs. Although we study the problem in the context of VDCs, we believe that our solution can be applied to any distributed computing infrastructure.We devised a practical framework for modeling the exponential complex problem of Placement and Provisioning together, and derived an optimal solution that considers the user quality, the relative fairness of quality between user groups, the network factors such as latency, and the cost of migration. Our proposed multi-step solution comprising of migration-free placement and provisioning for quick *online* resource allocation i.e., improved performance and a global utility-optimal placement and provisioning for improving the resource utilization i.e., increased scalability - provides a pertinent approach to the combinatorial problem of resource fragmentation.

The key insight from our GUPP heuristic is the minimization of the 'utility loss' due to sub-optimal allocations. Our Cost-Aware Migration algorithm is incremental and selects only VD pairs that can be swapped such that the net benefit of every pair of VDs swapped is positive - while considering: the snapshot cost, the deployment cost and the network cost of migration. Our simulation results demonstrate that defragmentation is an important optimization step that can enable CSPs to substantially improve user QoE and VDC scalability. In addition, our proposed defragmentation scheme empowers CSPs to handle resource allocations of VDs across data centers in a manner that balances trade-offs between cost incurred and QoE delivered.

REFERENCES

[1] VMLab: Testbed for Desktop Virtualization Experiments. http://vmlab.oar.net.

[2] A. Berryman, P. Calyam, A. Lai, and M. Honigford. VDBench: A Benchmarking Toolkit for Thin-client based Virtual Desktop Environments. In *Proc. of IEEE CloudCom*, 2010.

[3] P. Calyam, R. Patali, A. Berryman, A. Lai, and R. Ramnath. Utility-directed Resource Allocation in Virtual Desktop Clouds. *Elsevier Computer Networks Journal (To Appear)*, 2011.

[4] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant Resource Fairness: Fair Allocation of Heterogeneous Resources in Datacenters. Technical Report UCB/EECS-2010-55, EECS Department, University of California, Berkeley, May 2010.

[5] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase. Virtual machine hosting for networked clusters: Building the foundations for autonomic orchestration. In *In Proc. VTDC 06*, 2006.

[6] A. Hillier. Server Capacity Defrag: Maximizing Infrastructure Efficiency through Strategic Workload Placements. http://www.cirba.com/assets/docs/CiRBA-wp-Server-Capacity-Defrag.pdf, 2011.

[7] G. Lee, N. Tolia, P. Ranganathan, and R. H. Katz. Topology-aware resource allocation for data-intensive workloads. *SIGCOMM Comput. Commun. Rev.*, 41:120–124.

[8] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A Resource Allocation Model for QoS Management. In *In Proceedings of the IEEE Real-Time Systems Symposium*, pages 298–307, 1997.

[9] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. Practical Solutions for QoS-based Resource Allocation Problems. In *In IEEE Real-Time Systems Symposium*, pages 296–306, 1998.

[10] D. B. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62:461–474, December 1993.

[11] H. N. Van, F. D. Tran, and J.-M. Menaud. SLA-Aware Virtual Resource Management for Cloud Infrastructures. In *CIT (1)*, pages 357–362, 2009.