Computer Networks 55 (2011) 4112–4130

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Utility-directed resource allocation in virtual desktop clouds ☆

Prasad Calyam [a,*], Rohit Patali [a,c], Alex Berryman [a], Albert M. Lai [b], Rajiv Ramnath [c]

[a] *Ohio Supercomputer Center/OARnet, The Ohio State University, United States*
[b] *Dept. of Biomedical Informatics, The Ohio State University, United States*
[c] *Dept. of Computer Science and Engineering, The Ohio State University, United States*

ARTICLE INFO

Article history:
Received 16 December 2010
Received in revised form 8 June 2011
Accepted 8 July 2011
Available online 31 July 2011

Keywords:
Virtual desktop clouds
Desktop content delivery
Thin-client benchmarking
Utility-directed optimization

ABSTRACT

User communities are rapidly transitioning their "traditional desktops" that have dedicated hardware and software installations into "virtual desktop clouds" (VDCs) that are accessible via thin-clients. To allocate and manage VDC resources for Internet-scale desktop delivery, existing works focus mainly on managing server-side resources based on utility functions of CPU and memory loads, and do not consider network health and thin-client user experience. Resource allocations without combined utility-directed information of system loads, network health and thin-client user experience in VDC platforms inevitably results in costly guesswork and over-provisioning of resources. In this paper, we develop an analytical model viz., "Utility-Directed Resource Allocation Model (U-RAM)" to solve the combined utility-directed resource allocation problem within VDCs. Our solution involves an iterative algorithm that leverages utility functions of system, network and human components obtained using a novel virtual desktop performance benchmarking toolkit viz., "VDBench" that we developed. The combined utility functions are used to direct decision schemes based on Kuhn–Tucker optimality conditions for creating user desktop pools and determining optimal resource allocation size/location. We deploy VDBench in a VDC testbed featuring: (a) popular user applications (Spreadsheet Calculator, Internet Browser, Media Player, Interactive Visualization), and (b) TCP/UDP based thin-client protocols (RDP, RGS, PCoIP) under a variety of user load and network health conditions. Simulation results based on the utility functions obtained from the testbed demonstrate that our solution maximizes VDC scalability i.e., 'VDs per core density', and 'user connections quantity', while delivering satisfactory thin-client user experience.

1. 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

☆ 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, VMware and Dell.

* Corresponding author.

E-mail addresses: pcalyam@oar.net (P. Calyam), rpatali@oar.net (R. Patali), berryman@oar.net (A. Berryman), albert.lai@osumc.edu (A.M. Lai), ramnath@cse.ohio-state.edu (R. Ramnath).

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" [1] (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).

Fig. 1 shows the various system and network components in a 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 hosted desktops (VDs) that host popular applications (e.g., Excel, Internet Explorer, Media Player) as well as advanced scientific computing applications (e.g., Matlab, Moldflow). The VDs share common physical hardware and attached storage drives. At the thin-client side, users connect to a connection broker via the Internet using TCP or UDP based remote display devices. Examples of thin-client protocols used by the remote display devices include the open-source Virtual Network Computing (VNC via TCP), Microsoft Remote Desktop Protocol (RDP via TCP), HP Remote Graphics Software (RGS via TCP), and Teradici PC over IP (PCoIP via UDP). The connection broker authenticates users using technologies such as active directory, and allows users to access their entitled desktops. The VDC hypervisors are instrumented to not only modify and manage operating system/applications within VDs, but are also tasked with the appropriate VD resource sizing and location determination while servicing user desktop requests. The considerations that influence mapping of user desktop requests to distributed VDs include: (a) load balancing to improve VDC scalability, (b) energy awareness to reduce VD operation cost, and (c) express migration (e.g., during disaster recovery) for continued VDC availability.

To allocate and manage VDC resources for Internet-scale user workloads of desktop delivery, "virtual desktop cloud service providers" (CSPs) are faced with several challenges. One major challenge for a CSP will be to large-scale provision and adapt the cloud platform CPU and memory resources to deliver satisfactory user-perceived 'interactive response times' (a.k.a. *timeliness*) as negotiated in Service

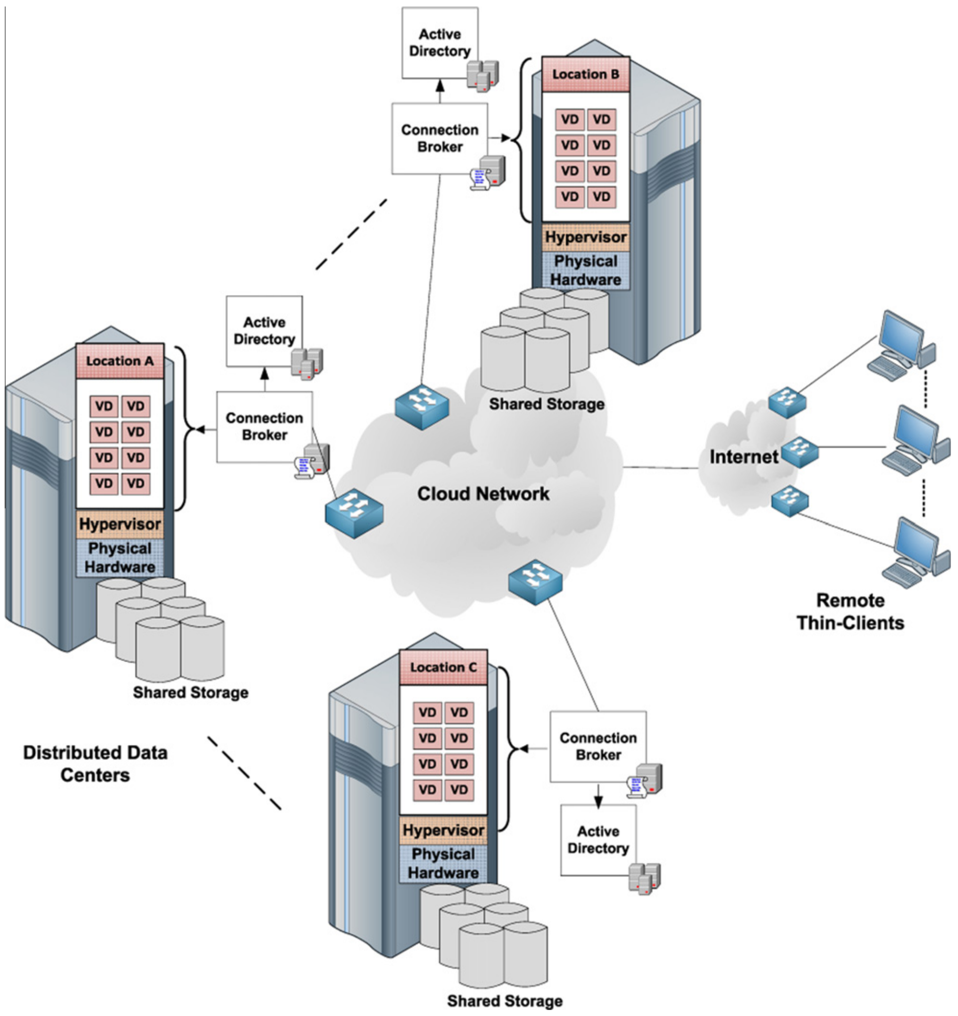

Fig. 1. Virtual desktop cloud system and network components.

Level Agreements (SLAs). Given the fact that memory is the most expensive and possibly the most contended resource in VDCs (i.e., users will idle their CPUs but will keep their applications always open on a desktop), it is a challenge for CSPs to suitably "overcommit" memory sizing for virtual desktops based on user activity profiling. Moreover, virtual desktop user activity when interacting with applications has intermittent spikes and bursts that further complicates the overcommitment estimation for CPU and memory sizing to satisfy timeliness. Hence, overcommitment in VDCs is more complex than in the case of server-virtualization, where user workloads can be modeled as transactions that have predictable resource consumption characteristics [4]. From the thin-client perspective, remote display protocols differ in terms of network bandwidth consumption from encoding (a.k.a. *coding efficiency*) based on how they crisply display different content such as text, images, and video. Further, since they employ different underlying TCP or UDP based protocols, they exhibit varying levels of user experience robustness under degraded network conditions. Hence, another challenge for CSPs is to suitably select thin-client display protocols based on application context such that coding efficiency is addressed while delivering satisfactory user Quality of Experience (QoE) even under degraded network conditions.

Our work is motivated by the fact that CSPs need frameworks and tools today that can enable them to handle the above challenges to build and manage VDCs at Internet-scale. To cope with increasing user workloads, extensive work has been done to efficiently manage server-side resources based on utility functions[1] of CPU and memory loads [2–5]. However, there is surprisingly sparse work on resource adaptation within VDCs that is coupled with utility functions of network health and thin-client user QoE. Works such as [16,17] highlight the need to incorporate network and user QoE factors into VDC resource allocation decisions. It is self-evident that any VDC platform's capability to successfully support virtual desktops is a function of both the server-side desktop performance as well as the remote user-perceived QoE. In other words, a CSP can provision adequate CPU and memory resources to a VD in the cloud, but if the thin-client protocol configuration does not account for network health degradations and application context, the virtual desktop will be unusable for the user. Hence, lack of proper "human-and-network awareness" in VDC platforms inevitably results in costly guesswork and over-provisioning while managing physical resources, which consequently annoys users due to high service cost and unreliable QoE.

In this paper, we develop an analytical model viz., *Utility-Directed Resource Allocation Model* (U-RAM) to solve the combined utility-directed resource allocation problem within VDCs along *timeliness* and *coding efficiency* quality dimensions. We show how this problem basically approximates to a binary integer programming problem whose solution is NP-hard. To solve this problem, we propose an iterative algorithm that has fast convergence. The iterative algorithm uses combined utility-directed decision schemes based on Kuhn–Tucker optimality conditions [11], and the previously mentioned considerations that influence mapping of user desktop requests to distributed VDs. The ultimate optimization objective is to allocate resources (i.e., CPU, memory, network bandwidth) such that the global utility (i.e., combined utility of all VDs at a data center) is maximized under the constraint that each VD at least meets it minimum quality requirement along *timeliness* and *coding efficiency* dimensions.

The implementation of our solution involves two-phases shown in Fig. 2. The first phase involves characterizing utility functions of system, network and human components for *offline* simulated user loads. The utility functions correspond to application (e.g., Internet Explorer, Media Player) profiles and user group (e.g., Engineering Industry, Campus Computer Lab) profiles. To collect measurements for characterizing the utility functions, we leverage a virtual desktop performance benchmarking toolkit viz., "VDBench" that we developed in [8]. 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' that leverage and extend our earlier research on slow-motion benchmarking of thin-clients [14]. In this paper, the novelty of using our VDBench toolkit is to obtain user application and group profiles in an offline manner, which can then be leveraged *online* by our iterative algorithm to: (a) create user desktop pools, and (b) optimize mapping of resource allocation size/location in the VDC under real user loads.

We deploy VDBench in a VDC testbed featuring: (a) popular user applications (Spreadsheet Calculator, Internet Browser, Media Player, Interactive Visualization), and (b) TCP/UDP based thin-client protocols (RDP, RGS, PCoIP) under a variety of user load and network health conditions. Simulation results based on the utility functions obtained from the testbed demonstrate that our solution maximizes VDC scalability i.e., 'VDs per core density', and 'user connections quantity', while delivering satisfactory thin-client user experience.

The remainder of this paper is organized as follows: Section 2 formulates the U-RAM problem. Section 3 details our optimization methodology and iterative algorithm. Section 4 presents the VDBench toolkit. Section 5 explains performance results of our proposed U-RAM solution. Section 6 describes related work. Section 7 concludes the paper.

2. Terminology and problem formulation

In this section, we first describe the quality dimensions and their related metrics. Next, we lay out our U-RAM problem scope by listing the involved parameters, requirements, and assumptions.

2.1. Quality dimensions and metrics

The *timeliness* quality dimension corresponds to the perceived interactive response times when a user is interacting with an application. Metrics that fall under this

[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.

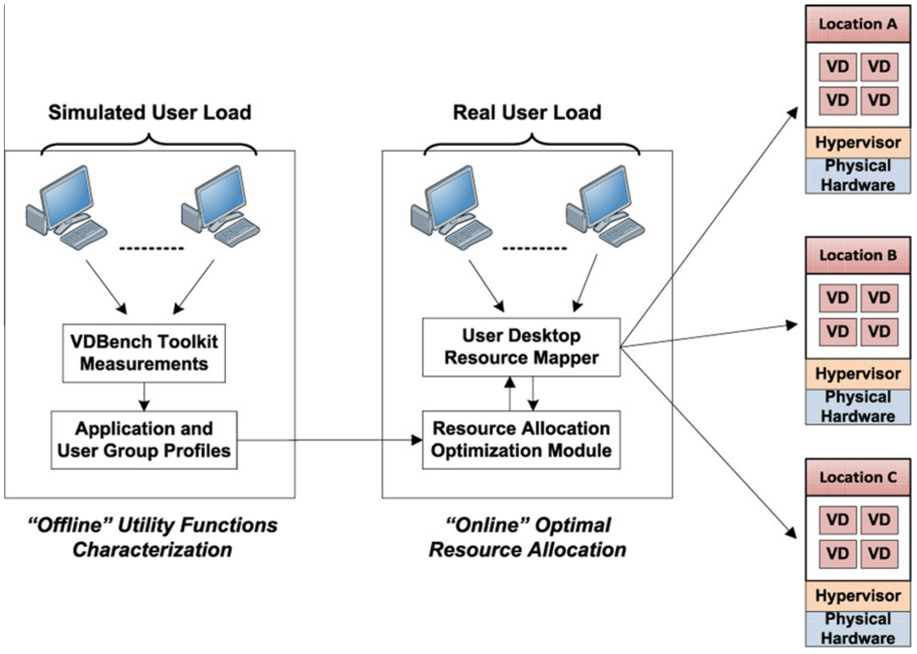
Simulated User Load
Real User Load
VDBench Toolkit Measurements
Application and User Group Profiles
User Desktop Resource Mapper
Resource Allocation Optimization Module
"Offline" Utility Functions Characterization
"Online" Optimal Resource Allocation
Location A
Location B
Location C
VD
Hypervisor
Physical Hardware

Fig. 2. Schematic for utility-directed resource allocation in a VDC.

quality dimension can be grouped under two categories: (i) atomic task time, and (ii) aggregate task time. Referring to Fig. 3, atomic task time is measured as the time taken for an intermediate task to complete while using an application. Intermediate tasks could include for e.g., Matlab application open time, "Save As" task time in Microsoft Excel or web-page download time in Internet Explorer. Aggregate task time refers to the overall execution time of several atomic tasks. Examples of aggregate response time can be inferred from Fig. 3 by calculating the time difference between t_3 and t_0 that corresponds to the time to complete a sequence of atomic tasks: opening an application, performing application tasks and closing the application.

The *coding efficiency* quality dimension corresponds to the perceived data throughput when an application has sizeable data flows that are transferred over a network path and consumed in real-time by a user. Metrics that fall under this quality dimension include: (i) render time, (ii) data transmission rate (a.k.a. bandwidth consumption), and (iii) video quality. Render time can be defined as the time taken for a screen update of an atomic task to

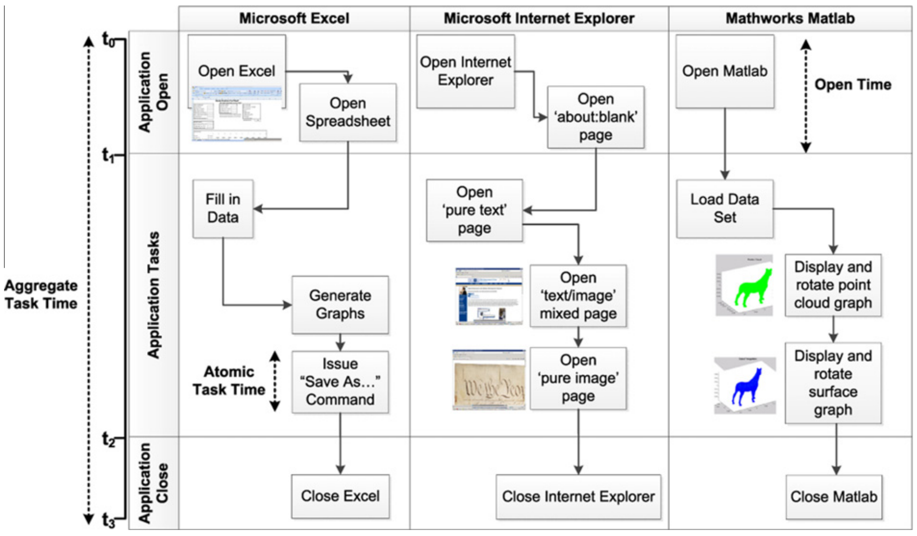

Fig. 3. Timeliness metrics in virtual desktop user tasks.

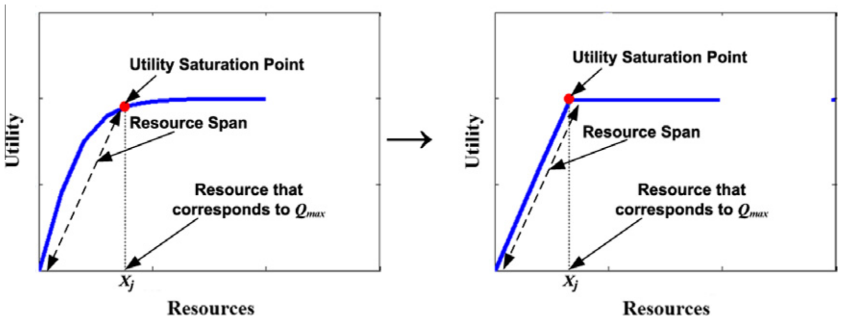

Fig. 4. Utility of non-linear function approximated to linear function.

complete, as observed in a network trace. An example is the time between the first and last packet of network activity (characterized by increase in packets per second) to go from-and-to an idle state after an activity burst during a image page download in a web-browser. Data transmission rate is the ratio of the amount of data transmitted for an atomic task screen update or an aggregate task's screen updates (i.e., consider the case of video playback in a Media Player), and the corresponding render time. Video quality in the context of a remote display device using thin-client protocols such as VNC, RDP, RGS or PCoIP can be defined as shown in Eq. (1). This definition is based on the video quality metric developed in [14] that normalizes the performance differences when displaying video frames with UDP and TCP based thin-client protocols. The normalization accounts for fast completion times of video frames playback with image impairments (e.g., tiling) in UDP based thin-clients, in comparison to relatively long completion times with no impairments (e.g., frame freezes) in TCP based thin-clients.

$$Video\ Quality = \frac{\frac{\frac{Data\ \ Transferred\ \ (aggregate\ \ fps)}{Render\ \ Time\ \ (aggregate\ \ fps)}}{IdealTransfer(aggregatefps)}}{\frac{\frac{Data\ \ Transferred\ \ (atomic\ \ fps)}{Render\ \ Time\ \ (atomic\ \ fps)}}{IdealTransfer(atomicfps)}}. \quad (1)$$

All of the above metrics are affected by various factors such as the: (a) concurrent user load on the VDC system, (b) level of rapid interactivity in the user application, (c) network health conditions (e.g., available bandwidth, latency, loss), and (d) the robustness of the thin-client protocol employed under degraded network conditions. Their utility functions in the larger context affect the global utility of a VD in a VDC. Obviously, determining actual values of above metrics that can be tolerated by users is a subjective process and involves extensive human subject experimentation [20]. In practice, the acceptable minimum-to-maximum ranges of these metrics for specific user applications are negotiated within SLAs. Nevertheless, the ranges can be set by intuition of what general users perceive as an interactive response time 'lag' (e.g., a user will perceive an undesired lag if a VD does not respond to a mouse click or does not open an application within a couple of seconds) or a degraded data transmission 'throughput' (e.g., a user will perceive an undesired throughput if a web-page of an image in a VD does not load within a couple of seconds).

2.2. The U-RAM problem

The U-RAM enables resource allocations in VDCs such that SLAs are met while resource overprovisioning is minimized. The terms used in U-RAM are defined as follows:

- Let there be n virtual desktops $\{v_1, v_2, \ldots, v_n\}$ and m resources $\{R_1, R_2, \ldots, R_m\}$. Note that the resource types we consider in U-RAM are CPU, memory and network bandwidth. The considerations that influence the mapping of user desktop requests to VDs include: (a) load balancing to improve VDC scalability (b) energy awareness to reduce the VD operating cost and (c) express migration for continued VDC availability. These considerations are described in detail in Section 3.3. If there are l data centers, the VDC's resource mapper shown in Fig. 2 maintains a preference list $\{L_1, L_2, \ldots, L_l\}$ which

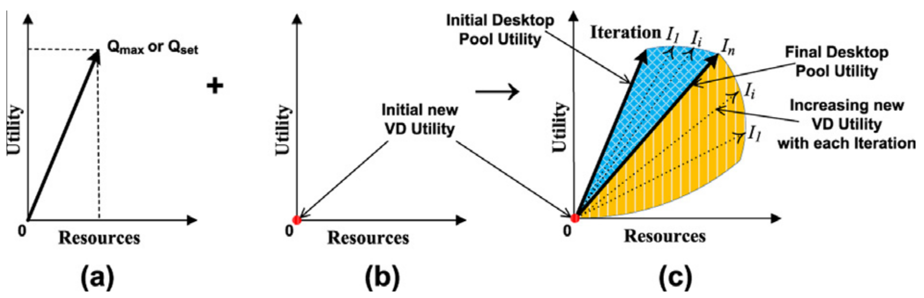
Q_{max} or Q_{set}
Utility
Resources
0
+
Initial new VD Utility
Initial Desktop Pool Utility
Iteration I_1 I_i I_n
Final Desktop Pool Utility
I_i
I_1
Increasing new VD Utility with each Iteration
(a)
(b)
(c)

Fig. 5. U-RAM for maximizing U; (a) Desktop pool utility when there are n provisioned VDs, (b) initial utility of new n + 1th VD, (c) U-RAM iterations to achieve final desktop pool utility.

is looked-up online to identify the preferred location of data center resources for a new user desktop request such that the resource sizing determined by U-RAM can be provisioned. A VD v_i placed at preferred location l is given by $v_{l,i}$.

- Resource R_j allocated to v_i can be represented as $R_{i,j}$. Utility U_i for virtual desktop v_i is the utility value generated by the system when v_i is allocated $R^i = \{R_{i,1}, R_{i,2}, \ldots, R_{i,n}\}$. Global utility is given by $U(R^1, R^2, \ldots, R^n) = \sum_{i=1}^{n} U_i(R^i)$.
- Each VD needs to satisfy quality constraints along d quality dimensions, $\{Q_1, Q_2, \ldots, Q_d\}$. In our study, we consider SLAs with 2 quality dimensions viz., *timeliness* and *coding efficiency*. Each VD has to be provisioned with at least minimal resources to satisfy minimum requirements along each of the quality dimensions Q_{min_k}. If abundant resources are available in the VDC, resources can be provisioned such that each VD has Q_{max_k} as indicated by the utility functions. Per the definition of utility, any additional resources provisioned will not each increase VD's utility beyond Q_{max_k}. In cases where there are more than minimal resources available but resources are not abundant, the resource provisioning is such that each VD has Q_{set_k} whose value is $Q_{min_k} < Q_{set_k} < Q_{max_k}$. The minimal resource requirement is expressed as $R_i^{min_k} = \left\{R_{i,1}^{min_k} R_{i,2}^{min_k}, \ldots, R_{i,m}^{min_k}\right\}$ where $R_{i,j}^{min_k} \geqslant 0;\ 0 \leqslant j \leqslant m$. A VD is feasible if it is allocated a minimum set of resources on every quality dimension in the SLA.

The *assumptions* for U-RAM are as follows:

- System and network resources are limited and maximum demand of all user VDs cannot be satisfied simultaneously. This assumption is valid because, there otherwise will not be any resource contention and all the user VDs can be allocated maximum resources.
- Each VD requires a certain minimum resource allocation to perform satisfactorily for a user; any additional allocation of resources adds utility to the VD that is perceived by the user as improved VD performance along *timeliness* and/or *coding efficiency* quality dimensions measured using the metrics described in Section 2.1.
- Utility of a VD increases monotonically with increase in allocated resources until a point after which, there is no user perceivable improvement in the quality with any additional allocation. Empirical results from our VDC testbed presented in Section 5.1.4 validate this assumption as well as the previous assumption. Moreover, this assumption is generally true in any computer system.
- Each provisioned VD has sufficient resources to meet the minimum resource requirements $R_i^{min_k}$ along each dimension Q_k; a new user VD request is rejected when its handling could potentially violate the SLA compliance along each dimension Q_k of an already provisioned VD. This assumption is valid because accommodating the new request will result in SLA violations in both the previously provisioned VD and the new VD.
- We assume that the utility functions are non-decreasing and concave.

Based on the above description, we are required to make resource allocations to each VD such that the global utility $U(R^1, R^2, \ldots, R^n)$ is maximized under the constraint that every VD is feasible with respect to each of the quality dimension Q_k. This problem lends itself to a binary integer programming problem that can be shown as follows:

$$\textit{Maximize} \quad f(U) = \sum_{i=1}^{n} U_i(R^i)$$

$$\textit{Subject to}: \sum_{i=i}^{n} R_{i,j} \leqslant R_j^{max} \quad \forall j = \{1, \ldots, m\}$$

$$R_j^{min} \leqslant \sum_{j=1}^{m} \sum_{i=1}^{n} R_{i,j} v_i \leqslant R_j^{max}$$

R_j^{max} is a resource allocation point for resource j beyond which there is no user perceptible improvement in the performance for any further allocation and hence, if allocated will waste resources and overprovision a VD. Although the utility function of a VD is a monotonically increasing nonlinear concave function, we adopt constraint programming to model our problem and approximate it to a piece-wise linear function as shown in Fig. 4. Such an approximation is shown to be reasonable as evidenced in works such as [5,6,9] for similar utility-driven resource allocation problems.

3. U-RAM optimization methodology

In this section, we first explain the Kuhn–Tucker optimality conditions [11] that we leverage to allocate resources in VDCs based on utility functions. Next, we describe our iterative algorithm to maximize VDC global utility U based on the Kuhn–Tucker optimality conditions. Lastly, we discuss the considerations that influence mapping of real users' desktop requests to distributed VDs in U-RAM implemented VDCs.

3.1. Kuhn–Tucker optimality conditions

Given that different resource allocation choices yield different global utility U values, our resource allocation problem is a 0–1 knapsack problem instance given by

$$\textit{Maximize} \quad f(x) = \sum_{i=1}^{n} v_i x_i$$

$$\textit{Subject to}: \sum_{i=1}^{n} W_i X_i \leqslant W, \quad x \in \{0, 1\}$$

Hence, the solution to our problem is NP-hard [19]. In order to solve the problem, we develop an iterative algorithm explained in Section 3.2 that makes resource allocation decisions with fast convergence. We derive conditions for optimality in our iterative algorithm with the following corollary.

Corollary. *Since, a VD's utility function is concave, it is twice continuously differentiable, i.e.,* $\frac{d^2U}{dR^2} = U_i'' \leqslant 0$ *for* $R > R_i^{min}$. *Hence, an optimum way to perform resource allocation can be obtained from Kuhn–Tucker optimality conditions [11] for such functions.*

A necessary and sufficient condition to perform optimal resource allocation in VDCs is for {i,j}, $i \neq j$ with $R_i > 0$ and $R_j > 0$, $U_i'(R_i) = U_j'(R_j)$.

Proof. The result is a conclusion of the Kuhn–Tucker Theorem. Let us suppose two allocations given by $R_i > 0$, $R_j > 0$ and $U_i'(R_i) > U_j'(R_j)$ and $i \neq j$, result in an optimal solution. Since $R_j > 0$, subtracting an infinitesimally small value from R_j and adding it to R_i will increase the global utility, since $U_i'(R_i) > U_j'(R_j)$. This contradicts our earlier assumption that it was an optimal allocation as the new allocation increased the global utility. Hence, for a resource allocation to be optimal, $U_i'(R) = U_j'(R_j)$ for $i \neq j$ with $R_i > 0$ and $R_j > 0$ at all times. □

From the above result, we can conclude that the optimal way to allocate resources in a VDC will be to provision resources to VDs such that their slopes of utility functions are equal at a given resource allocation decision point. Empirical results from our VDC testbed presented in Section 5.1.4 clearly show that the resource requirements are vastly different for different applications. For example, the CPU requirement for satisfactory Matlab application's user experience is much greater than that for Excel application. As another example, the network bandwidth requirement for satisfactory Media Player application's user experience is much greater than that of Internet Explorer application. Consequently, it is not reasonable to allocate identical amount of resources for different user groups who use different applications or subsets of applications. For argument sake, we can create a desktop pool for an 'Engineering Site' whose users mainly use Matlab, Internet Explorer and Excel, and a separate pool for a 'Distance Learning Site' whose users mainly use Media Player, Internet Explorer and Excel. Obviously, application grouping within a desktop pool depends on the actual desktop requirements of user groups. Our goal now is to achieve maximized utility of VDs within a desktop pool given any application grouping, which then ensures that the global utility is optimal as deduced from the Kuhn–Tucker optimality conditions.

3.2. Algorithm to maximize U

We now explain our iterative algorithm's inner workings to maximize VDC global utility U. We remark that this algorithm (pseudocode listed in Algorithm 1) resides inside the *Resource Allocation Optimization* module shown in Fig. 2. We presume that the algorithm is aware of the amount of resources needed to achieve Q_{min} and Q_{max} for each user group through the profiling step with offline benchmarking measurements in Fig. 2. To convey how our algorithm maximizes the utility of each desktop pool, we use the illustration in Fig. 5.

Let us start with a scenario where n VDs belonging to a desktop pool are already provisioned all the resources within a VDC with a utility $U_x(Q_x)$ that results in Q_{max} or Q_{set} as marked in Fig. 5(a). When a new n + 1th VD request arrives, the VDC is aware of the request's $R_i^{min/max}$ for $Q_{min/max}$ based on the user group it is originating from, and the corresponding profile obtained through offline benchmarking measurements. Obviously, the new VD's utility is zero before any resources are allocated to it as shown in Fig. 5(b). In each iteration $I_1 \ldots I_n$, the VDC adds resources to the new VD to increase its slope of utility by subtracting the corresponding amount of resources from the previously provisioned VDs. This in turn causes a decrease in the utility of the previously provisioned VDs in each iteration. The iterations continue as shown in Fig. 5(c) until the final desktop pool utility is such that the new VD's utility slope is same as the (decreased) utility slope of the previously provisioned VDs.

Algorithm 1. Given a set of VD requests, find resource allocation with maximized global utility U

```
1: Input: Utility functions $U_x(R_x)$ of user groups from offline profiling, list of currently available resources $\{R_1, R_2 \ldots R_m\}$ at $l$ data centers
2: Output: Resource allocations $R_{i,j}$ for VDs $v_i$ where $i \in \{1 \ldots n\}$ that will maximize global utility $U$
3: begin procedure
4: for each new VD $v_i$ where $i \in \{1 \ldots x \ldots n\}$ do
5:     Refer to preference list $\{L_1, \ldots L_x, \ldots L_l\}$ and identify the data center location $L_x$ for placement
6:     Calculate current $U_x(R^x)$ for the selected desktop pool $g \in \{1 \ldots p\}$ at $L_x$ where $v_{l,i}$ is being provisioned, and $R_{p,j}^{SLACK} = R_{p,j}^{set} - R_{p,j}^{min}$
7:     if $\left(R_j + \sum_{g=1}^{p} R_{g,j}^{slack}\right) \geqslant R_{i,j}^{min}\Big)$ then
8:       if $(R_j \geqslant 0)$ then /* If resources are available in the desktop pool*/
9:         if $\left(R_j \geqslant R_{i,j}^{max}\right)$ then
10:          Subtract $R_{i,j}^{max}$ from $R_j$
11:          Add $R_{i,j}^{max}$ to $R_{i,j}$
12:        else if $\left(R_j \geqslant R_{i,j}^{min}\right)$ then
13:          $R_j = 0$
14:          Add $R_j$ to $R_{i,j}$
15:        end if
16:      end if
17:      while $\left(\frac{dR_{i,j}}{dR} < \frac{dR_{c,j}}{dR}\right)$ {
18:        if $(R_j \geqslant 0)$ then /*Allocate resources to VD from available resources at a location till slopes match*/
19:          Subtract $R'$ from $R_j$ /* $R'$ is the available resource decrement step size*/
20:          Add $R'$ to $R_{i,j}$
21:        else /* If there are no free resources available in the desktop pool*/
22:          if $\left(R_{g,j}^{slack} \geqslant R'\right)$ then
23:            Subtract $R'$ from $R_{g,j}$
24:            Add $R'$ to $R_{i,j}$
25:          end if
26:        end if
27:    else
28:      exit notifying VDC capacity reached!
29:    end if
30: end for
31: end procedure
```

Herein, we explain our U-RAM iterative algorithm with a case study featuring 3 desktop pools at a data center as shown in Figs. 6 and 7. Fig. 6 shows overprovisioned VDs when handled by a "fixed resource allocation model" (F-RAM), which is a commonly used strategy in VDCs. Since the CSPs do not have any utility-directed knowledge of the resource requirements, each VD when using F-RAM is provisioned with excess (wasteful) resources that produce utility Q_{excess}, which in terms of user perception is the same as Q_{max}. Fig. 7(a) shows how U-RAM handles new VD requests by allocating resources that produce utility Q_{max} when there are freely available resources. Once all the resources have been allocated, as newer VD requests arrive, the iterative algorithm handles the new VD requests by allocating resources in a manner that produce utility Q_{set} in all VDs as shown in Fig. 7(b). Lastly, Fig. 7(c) shows a case where it was determined in one of the U-RAM iterations that resource allocation to a new VD would cause the utility of one of the previously provisioned VDs to drop below Q_{min}. Such a drop results in violation of SLA compliance along *timeliness* and/or *coding efficiency* dimensions. Hence, the new VD requests will be rejected thereafter and the subtracted resources from the previously provisioned VDs will be returned back to those VDs. Using the corollary from Section 3, we can say that this is the optimum resource allocation when using U-RAM.

3.3. Considerations for resource location determination

These considerations are used in the "User Desktop Resource Mapper" in Fig. 2 to generate a resource location preference list input for our iterative algorithm. In our U-RAM, we separate out VD 'provisioning' aspects and resource location for VD 'placement' aspects in VDCs since both are equally complex NP-Hard problems. Detailed arguments that favor separation of these two aspects can be found in works such as [5,7].

In this paper, we emphasize the problem of 'provisioning' and assume that the 'placement' problem has been solved by alternate methods, which in fact have been well studied in literature [21,22]. The placement decisions are often affected by CSP-specific policies and the distributed architectures of data center sites. Hence, there can be various reasons for choosing one data center over the other to place VDs whose resource sizing has been determined by U-RAM. In the following, we briefly provide heuristics for the three most common considerations that may be used individually or in combination with each other:

3.3.1. Load balancing

System load can vary at different data centers based on the types of desktop pools and number of VDs they are hosting at any given time. The U-RAM model should be provided with intelligent inputs for VD placement such that it facilitates load balancing in the VDC. For load balancing, a decision can be made to place a VD in a data center with least resources utilization. Based on this, a preference list $\{L_1, \ldots, L_n\}$ can be generated as a set of feasible data centers to handle a particular VD request connection. For each connection, there should be a primary and one or more secondary data centers listed taking into account the network path conditions between the thin-client and the data center. If r_l is the total amount of resource available at a data center l, then load balancing heuristics is given as:

$$r'_l = r_l - \sum_{i=1}^{n} R_{i,j}, \quad \forall j \in \{1, 2 \ \ldots \ m\}; \quad \forall l \in \{1, 2 \ \ldots \ L\}$$

where $\max\{r'_l\}$ gives the data center with most available resources.

3.3.2. Energy awareness

The cost of energy can be substantially more at one data center as compared to another. Under such situations, it will be beneficial to allocate $v_{l,i}$ at a location that will minimize the CSP's operating costs for a VD. These costs can be associated with power consumption expense at the data center. Dynamic power management schemes such as the one proposed in [21] can be used to consolidate VDs to a minimum number of hosts across locations that cumulatively lower the power consumption expense for CSPs.

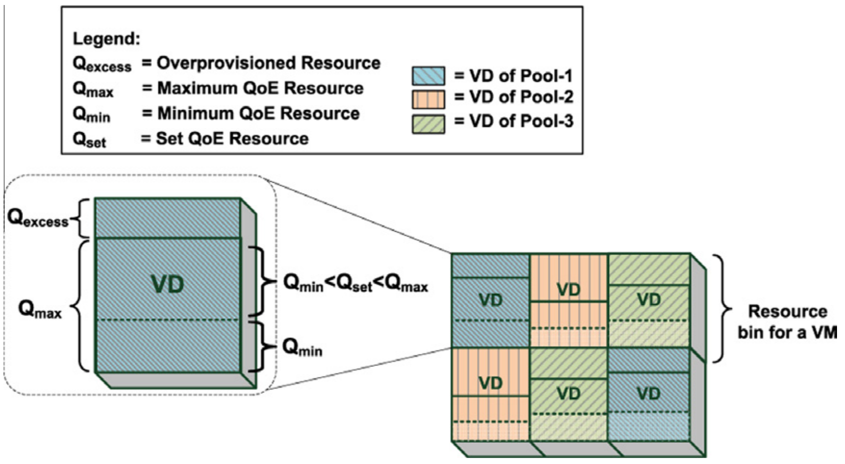
Legend:
Q_{excess} = Overprovisioned Resource
Q_{max} = Maximum QoE Resource
Q_{min} = Minimum QoE Resource
Q_{set} = Set QoE Resource
▨ = VD of Pool-1
▥ = VD of Pool-2
▨ = VD of Pool-3
Q_{excess}
Q_{max}
VD
$Q_{min}<Q_{set}<Q_{max}$
Q_{min}
VD
Resource bin for a VM

Fig. 6. Overprovisioned VDs when handled by F-RAM.

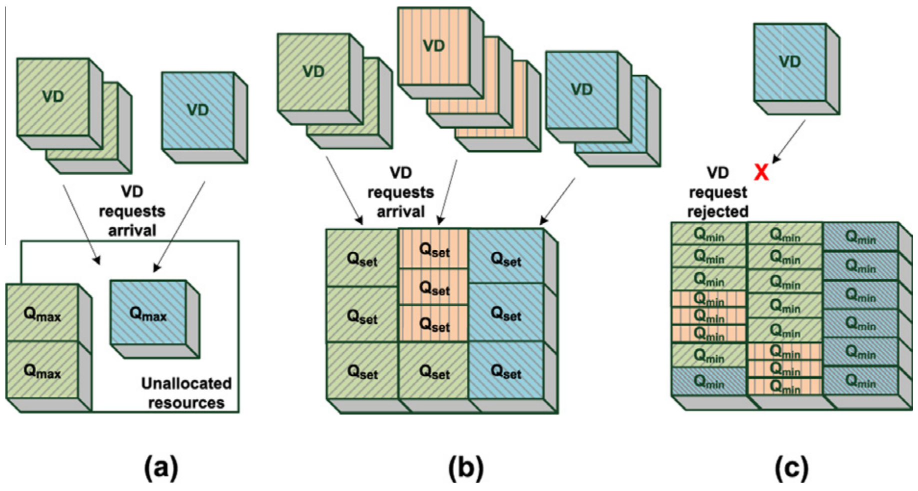

Fig. 7. Example to illustrate U-RAM iterative algorithm handling of VD requests; (a) New VD requests handling with freely available resources, (b) new VD requests handling with all available resources allocated, (c) new VD request rejected when SLA violation occurs.

$$\text{Minimize } f(C) = \sum_{i=1}^{n} v_{l,i}C_i,\ v_{l,i} \in 0,1$$

We note that our resource allocation problem can be extended to include energy awareness. For this, we can maximize the global utility U by adding a constraint in the objective function that seeks to minimize cost of operating a VD at location l.

3.3.3. Express migration

In case of a data center failure (e.g., disaster recovery case) or a planned downtime of a set of allocated resources, a CSP will need to migrate VDs quickly from one data center to another. The migration process is generally expensive, time consuming and disruptive since it requires reconstructing snapshots of system and network components from a known point before migration such that the imminent migration does not violate SLA requirements. A heuristic can be used in express migration that prioritizes VD request fulfillment during the migration process when it is not possible to satisfy SLAs of all VD requests. A VD request connection of a more business critical application is thus given higher priority and should be migrated first to a data center with more redundancy in place to prevent further outages. There are efficient approaches such as the one in [22] to handle such express migrations without loss of connectivity in interactive user applications.

4. VDBench toolkit

In this section, we briefly describe the techniques used in VDBench to benchmark system, network and human components for deriving corresponding utility functions. VDBench toolkit uses a novel methodology and metrics described in Section 2.1 to benchmark thin-client based virtual desktop environments. In the following, we first briefly explain our methodology to perform user-load simulation based benchmarking to derive utility functions of system components. Subsequently, we briefly explain our methodology to perform slow-motion application interaction based benchmarking to derive the corresponding utility functions of network and human components. For more detailed descriptions of VDBench toolkit development and how it leverages and extends earlier research on slow-motion benchmarking of thin-clients [13,14], please refer to [8].

4.1. User load simulation based benchmarking

Fig. 8 shows the logical connection between all the layers in the VDBench toolkit. The *management service* is responsible for the provisioning of desktops, launching the load generation scripts on the VDs, monitoring their progress, and recording results of the measurements. An experiment begins when the VDBench management service provisions the first VD, and then spawns a *measurement service* on the VD, which then starts running the load generating script in order to establish a baseline of application response times. The load generation script automates the execution of a sample user workflow of *application tasks* as shown in Fig. 3. The workflow involves simulating a user launching applications such as Matlab, Excel, Internet Explorer and Media Player in his/her respective user group. Once the application is open, different application tasks are randomly selected for execution until all of the tasks are completed. Next, the script closes the launched application in preparation for the next iteration. A controllable delay to simulate user *think time* is placed between each of these steps, as well as the application tasks. An exaggerated user think time is configured in VDBench in order to incorporate slow-motion principles into thin-client protocol experiments.

The above process is repeated 10 times for each test so that a steady state of resource utilization measurements can be recorded in the *measurement log*. Once the initial baseline is established, an additional VD is provisioned by the management service and the load generation script is run concurrently on both VDs while application response time measurements are collected in the measurement log. This pattern of provisioning a new VD running the load generation script, and collecting application response time data is continued until the response times hit the *response time ceiling* negotiated in SLAs, and subsequently the experiment is terminated.

4.2. Slow-motion application interaction based benchmarking

To measure performance characteristics of thin-client protocols (e.g., RDP, RGS, PCoIP), we employ the slow-motion benchmarking technique originally developed by [13]. This technique employs two fundamental approaches to obtain an accurate proxy for the user-perceived performance: (i) monitoring server-side network activity, and (ii) using slow-motion versions of on-screen display events in applications. Fig. 9 shows a sample packet capture of a segment of a slow-motion benchmarking session with several on-screen display events. The session involves atomic tasks i.e., web-page loads with different content such as a blank page, a page containing only a text version of the US Constitution, a web page with mixed graphics and text, and a web page containing only high-resolution images. We can distinctly notice the render times in the case of different thin-client protocols for the screen updates of the atomic tasks, and thus can compare their performance. Recall that our definition of render time is the time between the first and last packet of network activity (characterized by increase in packets per second) to go from-and-to an idle state after an activity burst during an atomic task. It is relevant to note that render time increases (i.e., thin-client protocol shows reluctance to return to idle traffic state) as the network health conditions degrade owing to the monitoring and adaptation algorithms used in the auto-scaling of the thin-client protocols.

The start and completion of screen-events are marked in VDBench by the transmission of 'marker packets' shown in Fig. 9 that are sent by the VDBench automation script at the server-side. A marker packet is a UDP packet containing information on the screen-event that is being currently displayed. The marker packets allow VDBench to synchronize thin-client user events, screen updates with server-side resource performance events. The render time can be visualized based on the duration of the network activity between marker packets. Over this render time interval, the amount data transmitted is recorded in order to calculate the data transmission rate or bandwidth consumption for an atomic task. For the slow-motion benchmarking of video playback workloads, a video is first played back at 1 frame per second (fps) and statistics of the corresponding packet captures are analyzed. The video is then replayed at full speed a number of times through all of the thin-client protocols, and over various network health conditions to obtain video quality measurements. This video quality metric thus relates the slow-motion playback to the full speed playback to see how many frames were dropped, merged, or otherwise not transmitted. Before recording all the slow-motion based measurements in the VDBench measurement logs, each experiment session is repeated 3 times and the obtained measurements are averaged.

5. Performance results

In this section, we start by describing the testbed setup and results for the offline utility functions characterization. Following this, we describe the simulation methodology and results to validate our U-RAM iterative algorithm's ability to online maximize VDC scalability with satisfactory user QoE.

5.1. Offline utility functions characterization

5.1.1. Testbed setup

Fig. 10 shows the various physical components and dataflows in our testbed setup that can be envisaged as a data center in a VDC. The testbed hardware resources can

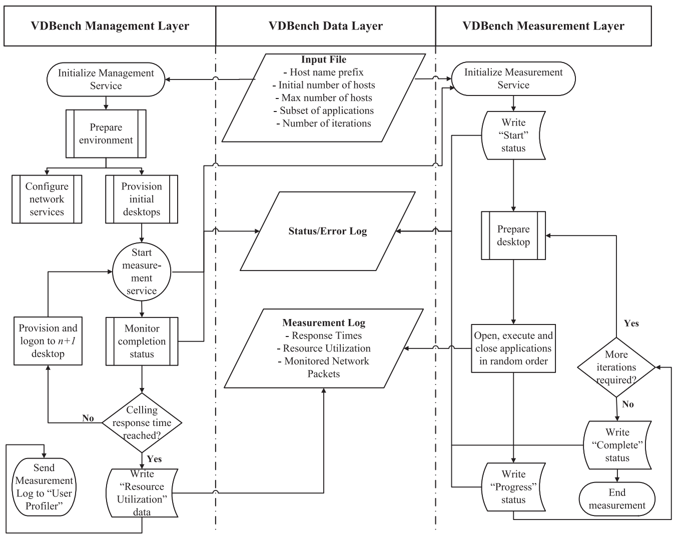
VDBench Management Layer
VDBench Data Layer
VDBench Measurement Layer
Input File
- Host name prefix
- Initial number of hosts
- Max number of hosts
- Subset of applications
- Number of iterations
Initialize Management Service
Prepare environment
Configure network services
Provision initial desktops
Start measurement service
Provision and logon to n+1 desktop
Monitor completion status
Celling response time reached?
No
Yes
Send Measurement Log to "User Profiler"
Write "Resource Utilization" data
Status/Error Log
Measurement Log
- Response Times
- Resource Utilization
- Monitored Network Packets
Initialize Measurement Service
Write "Start" status
Prepare desktop
Open, execute and close applications in random order
More iterations required?
Yes
No
Write "Complete" status
End measurement
Write "Progress" status

Fig. 8. VDBench control logic for benchmarking.

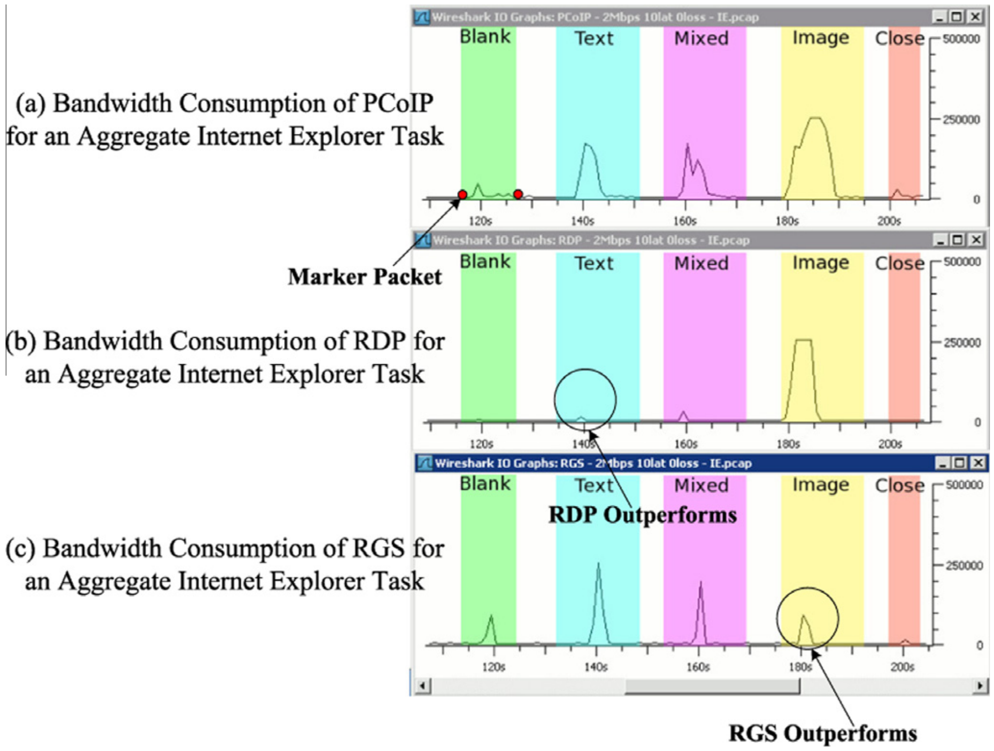

Fig. 9. Example traces to illustrate slow-motion benchmarking.

be proportioned to support multiple VDs, and the network emulator allows emulation of different thin-client network paths with different network health conditions. The hypervisor layer is used for infrastructure management. The hypervisor kernel's memory management functions are invoked during the user-load simulations and the virtual network switch is employed in the slow-motion application interaction measurements. Our VDBench management virtual appliance along with a fileserver/database, as well as the desktop pools containing individual VDs are provisioned on top of the hypervisor. The VDC environment is run using VMware ESXi 4.0 on top of IBM HS22 Intel Blade Servers in a IBM Blade Center chassis. Each blade server has two Intel Xeon E5504 quad-core processors and 32 GB of RAM, with access to a 9 TB shared, 'serial attached SCSI' (SAS). Each VD in our testbed ran a Windows XP OS instance. The network emulation is done using NetEm, which uses the traffic control `tc` command, which is part of the *iproute2* toolkit. Bandwidth limiting is done by implementing the token-bucket filter queuing discipline on NetEm. Traffic between the client and the server is monitored using a span port configured on a Cisco 2950 switch. The span port sends a duplicate of all the packets transmitted between the VD and the thin client to a machine running Wireshark tool to capture the packet traces and to filter out non-display protocol traffic during offline profiling.

5.1.2. Influence of user loads

In this subsection, we show results from our testbed that illustrate how *individual applications* are affected in terms of *timeliness* and *coding efficiency* metrics *with increasing user loads*. We remark that these performance results also serve to validate the VDBench methodology for user load simulation based benchmarking explained earlier in Section 4.1.

The time taken to open applications clearly increases with the increasing user load as shown in Fig. 11. Excel, Internet Explorer, and Matlab went from 1.3 s, 2.3 s, and 10.8 s, to 5.9 s, 7.7 s, 38.5 s corresponding to 472%, 301%, 359% increases with increasing user load, respectively. The loading of applications is heavily dependent on transferring data from the hard disks into memory. When the memory granted to a VD is constricted due to the "balloon driver" [18][2] that performs memory management within our VMware ESXi hypervisor used in the testbed, the VD must make room for this new application in memory. If the VD has not exhausted its memory shares in the allocated resources, the memory management tools and balloon driver will decrease the memory pressure on a VD, thus granting the VD more memory to use for applications. However, if the VD has exhausted its allocated share of the memory, the VD must invoke its own memory management tools and start deleting old memory pages using a garbage collection process, or swap them to its own virtual disk. These

[2] The balloon driver is controlled by the VMware ESXi hypervisor in order to force the guest operating system in a provisioned VD to free up the pages using the guest operating system's native memory management algorithms. The freed up pages are utilized by the hypervisor for redistribution. The balloon driver reports to the guest operating system in the VD just like a normal program that has higher and higher memory utilization.

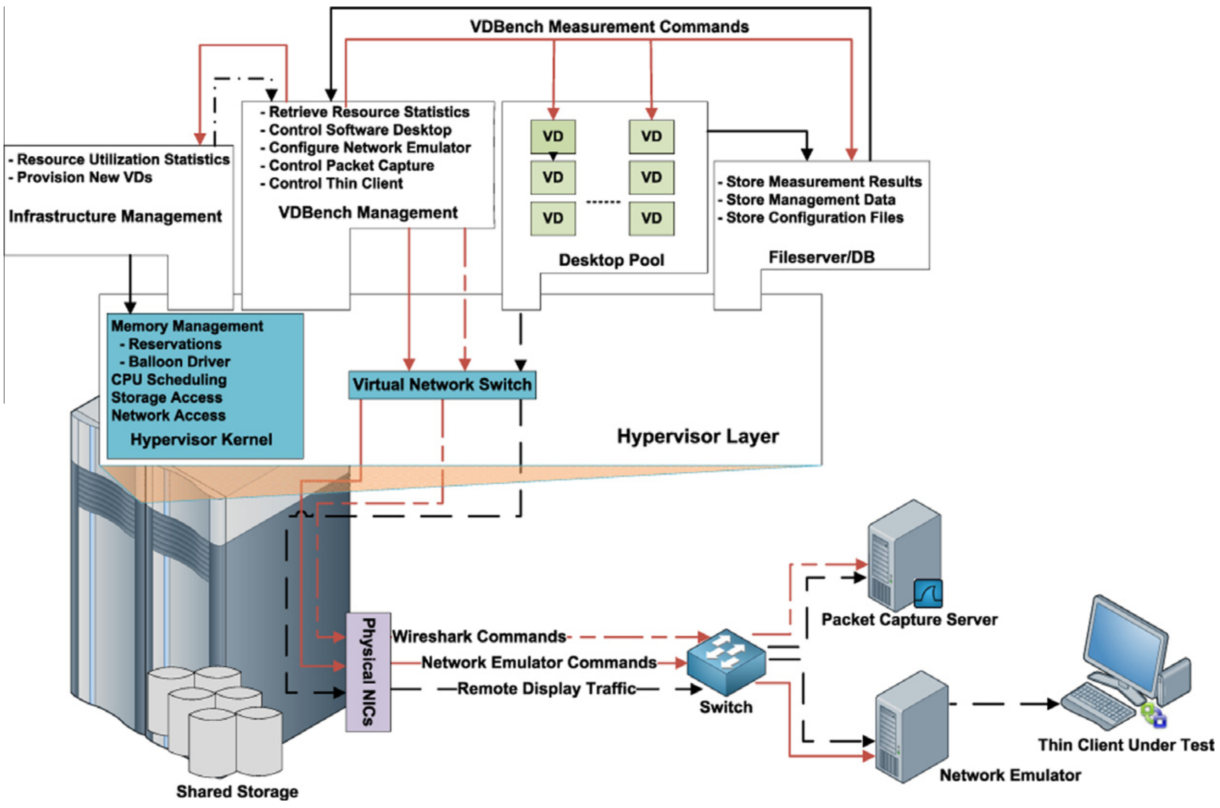

Fig. 10. Components of VDBench and data flows.

processes take time to complete, thus extending the application open times as the user load increases.

The time taken for actual tasks to complete within an application are shown in Fig. 12. The task titled '*Matlab Graph spin*' first involved spinning a point-cloud model of a horse, and then pre-computing the surface visualization of the point-cloud data. The data sets are precomputed in order to limit CPU utilization and consume more memory. The task initially took 34 s and grew to take 127 s, corresponding to a 273% increase. This result highlights the fact that applications such as Matlab are highly sensitive to resource overcommitment and need special desktop provisioning considerations. The Internet Explorer tasks involved loading a page with different types of content. The time taken to load a page of only an image saw the biggest increase starting at .75 s and grew to 2 s. The other two page types both remained under .5 s to complete, even under the highest system load. This increase, while statistically significant, is not obviously perceivable to the user. The task titled '*Excel Save*' is the time taken for 'Save As...' dialog box to appear. This Excel task originally took .9 s and later took 1.3 s only showing a 44% increase.

These results underscore the fact that applications behave uniquely from the point of atomic and aggregate task times under varying user loads. Hence, it is critical to manage resource overcommitment appropriately to

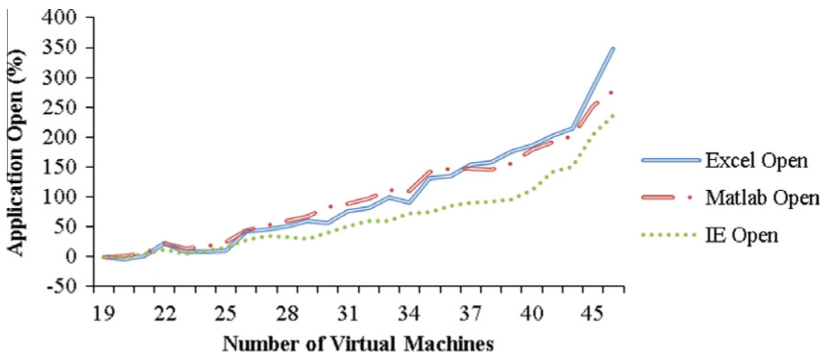

Fig. 11. Application open times with increasing system loads.

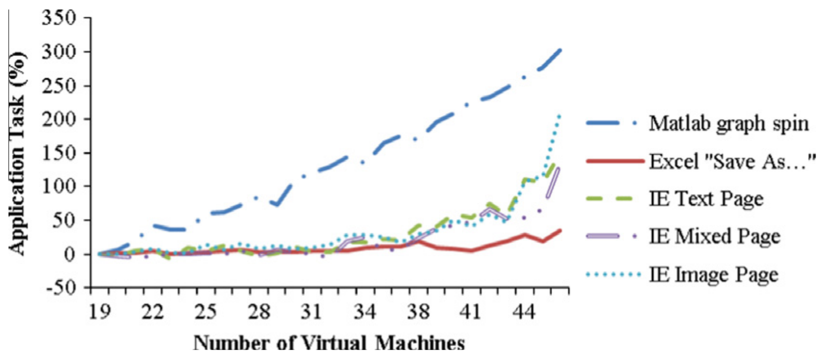

Fig. 12. Application task times with increasing system loads.

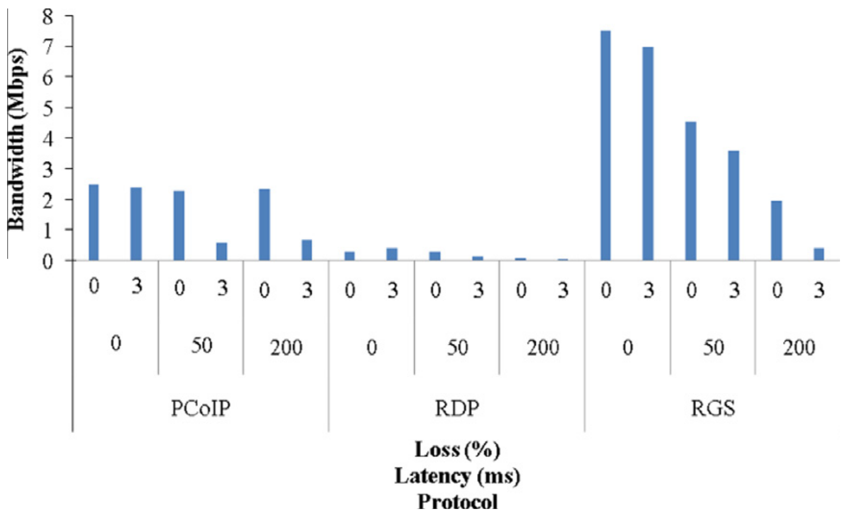

Fig. 13. Bandwidth consumed by text page under network emulation.

avoid wastage of resources by overprovisioning, while not adversely impacting user QoE.

5.1.3. Influence of network health conditions

In this subsection, we show results from our testbed that illustrate how *individual applications* are affected in terms of *timeliness* and *coding efficiency* metrics *with degrading network health conditions*. We remark that these performance results also serve to validate the VDBench methodology for slow-motion application interaction based benchmarking explained earlier in Section 4.2.

Figs. 13 and 14 show results from our slow-motion benchmarking testing of RDP, RGS, and PCoIP under a combination of both a range of network latencies (0 ms, 50 ms, and 200 ms) as well as no packet loss and a relatively high packet loss of 3%. These network health conditions were selected because they provide insights into how these thin-client protocols may behave on actual LAN, WAN and wireless last-mile connections. The ratio of the data transmitted and time taken for each screen update are used to calculate the ‘data transmission rate’ or bandwidth consumed. We can see that in general, there was a reduction in bandwidth consumed for each of the protocols when comparing low-latency, no loss with low-latency, and high loss network health conditions. This is not surprising as this likely indicates a throttling of bandwidth consumption as packet loss increased. RDP transports the ‘text only’ web page with very minimal data transmitted and thus has a higher *coding efficiency* for text. Both RDP and RGS maintain a relatively constant amount of data transmitted across increased latency and loss. The UDP-based PCoIP also in most cases exhibited a significant increase in the amount of data transmitted as latency and packet loss was increased. However, the user experience subjectively was found to be acceptable for PCoIP even at relatively high packet loss rates of 3%. While the TCP-based protocols (RDP and RGS) were affected by increased latency and loss, the UDP-based PCoIP was largely unaffected by increases in latency.

Fig. 15 shows the video performance of PCoIP, RDP, and RGS under a variety of network health conditions. PCoIP under no loss conditions showed relatively stable performance with increased latency. However, under high loss conditions, PCoIP suffered a severe drop in video quality. This behavior is due to the display protocol being based on UDP, with recovery from loss being complex. While RDP performed the best under idealized conditions, RDP suffered dramatically under either loss or high latency

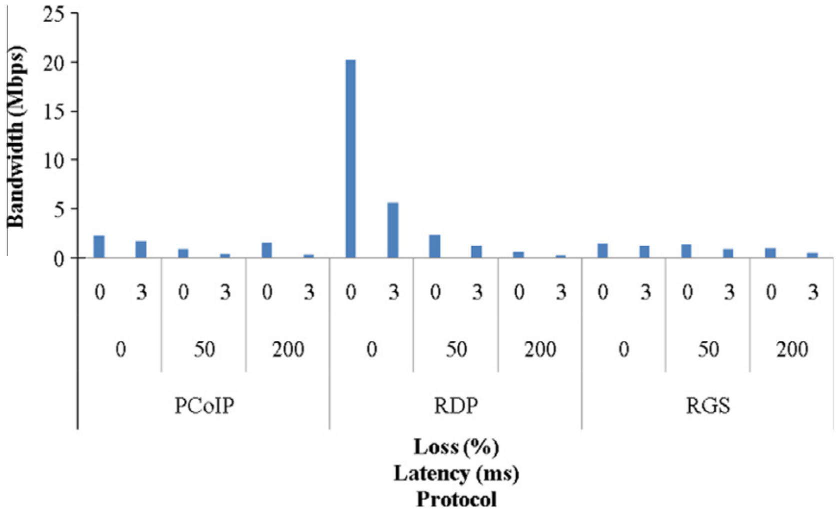

Fig. 14. Bandwidth consumed by image page under network emulation.

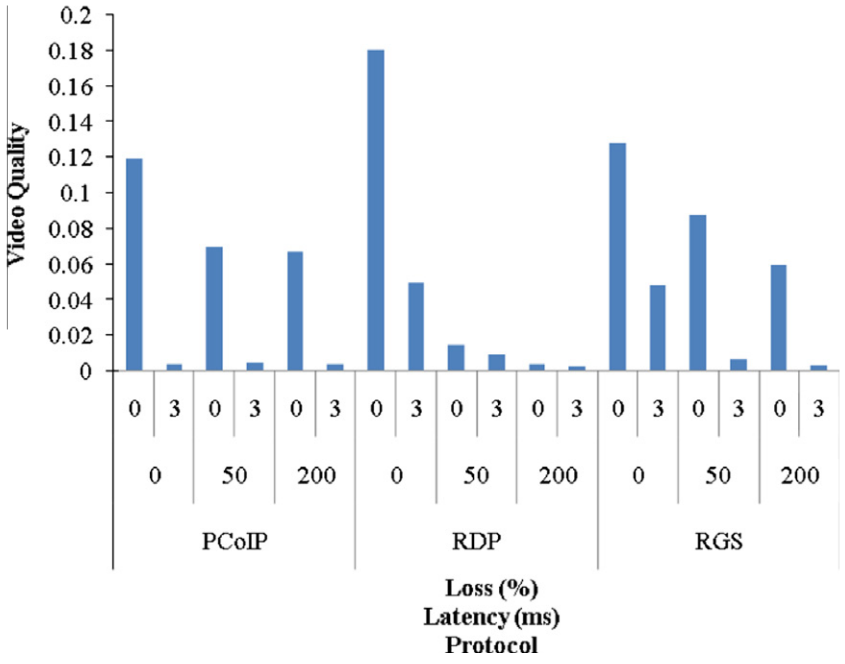

Fig. 15. Video Quality comparison under network emulation.

conditions. RGS was not as dramatically affected as RDP by increased latency. However, under high loss conditions, RGS had significantly reduced video quality.

These results underscore the fact that thin-client protocols are designed differently to display specific type of content. Consequently, guesswork of thin-client configuration in VD sessions without considering application context and network health conditions between user and data center network paths, can greatly impact user QoE.

5.1.4. Utility surfaces

In this subsection, we leverage the performance results of individual applications under increasing user loads and degrading network health conditions to construct utility functions of user groups. As concluded in Section 3.1, the U-RAM applies Kuhn–Tucker optimality conditions for "desktop pools" that comprise of subsets of applications pertaining to a user group to maximize global utility *U*.

For the sake of argument, we consider three user groups: (i) Engineering Site, (ii) Campus Computer Lab, and (iii) Distance Learning Site. We assume that the Engineering Site users are connected to a VDC via enterprise network paths with low latency and low loss, and routinely use Matlab, Internet Explorer and Excel applications. Also, we assume Campus Computer Lab users are novice students connecting from computing sites to a VDC via

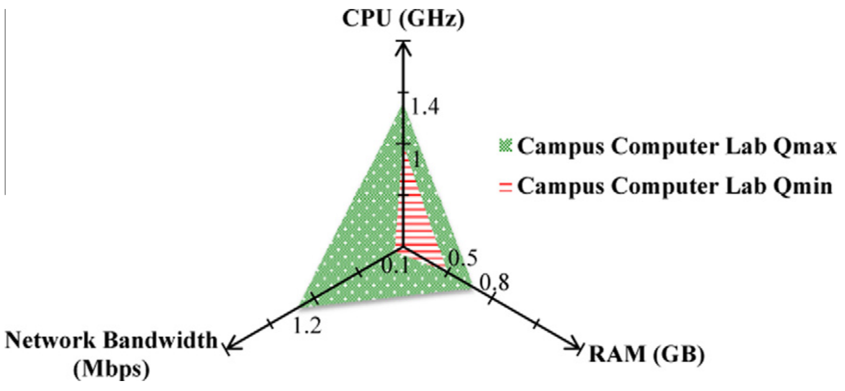

Fig. 16. Utility curve of Campus Computer Lab user group.

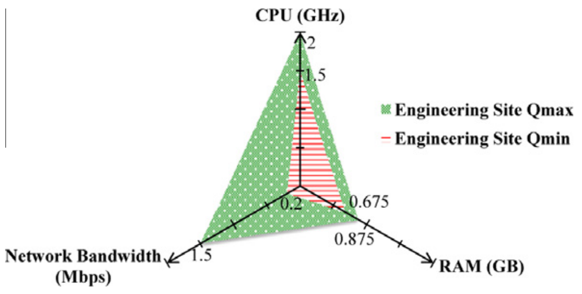

Fig. 17. Utility curve of Engineering Site user group.

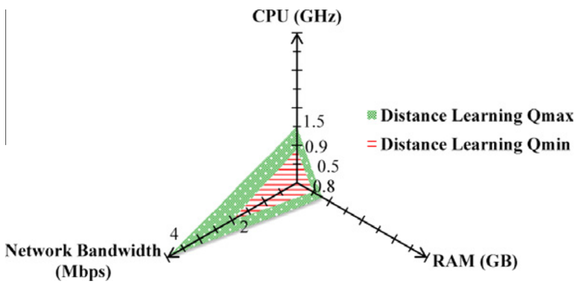

Fig. 18. Utility curve of Distance Learning user group.

community network paths with high latency and low loss, and routinely use Internet Explorer and Excel applications. Further, we assume Distance Learning Site users connecting to a VDC via residential network paths with high latency and high loss, and routinely use Media Player (e.g., to watch classroom lecture videos), Internet Explorer and Excel applications.

Figs. 16–18 show the surfaces of the utility functions of the three user groups, respectively. To create the surface for a user group, we combine the individual application profiles of that user group and average the resources needed to obtain utility surfaces of Q_{min} and Q_{max} for the corresponding network health characteristics. The operating points in the space between the Q_{min} and Q_{max} surfaces of a user group represent feasible VD configurations in a desktop pool i.e., VD configurations that are successfully provisioned to meet SLA requirements. Any operating point that lies above the Q_{max} surface represents a VD configuration that is overprovisioned. Similarly, any operating point that lies below the Q_{min} surface represents a VD configuration that is underprovisioned, which violates SLA requirements. Table I shows the SLA specifications that we chose to generate the utility functions of the three user groups. As discussed in Section 2.1, determining tolerance thresholds of *timeliness* and *coding efficiency* metrics for actual users need to be specified in SLAs after extensive subjective testing with human subject experimentation [20]. Nevertheless, the SLA specifications in Table I have been chosen based on our intuition of what general users perceive as an interactive response time 'lag' or a degraded data transmission 'throughput'. Further, we remark that choosing other SLA specifications does not change the validity of the results presented in this paper.

5.2. U-RAM iterative algorithm validation

In this section, we use the utility functions that were generated in the previous section using the VDBench toolkit in a simulation to show how the cloud scalability is increased using U-RAM in terms of 'VDs per core' and 'user connections quantity'.

The simulation setup is as follows: we simulate a VDC with several data center sites with each data center site defined to mimic the testbed (see Fig. 10) resources that were used to create the utility functions. It is relevant to note that if the system and network resources are different from the ones we used in our testbed, CSPs would need to re-run our VDBench scripts to create utility functions for Q_{min}, Q_{max}, and Q_{excess} that are pertinent to their unique

Table I
SLA specifications used in utility surfaces.

SLA metric	Applications	Max. demand	Threshold
Open time	All	Memory	20% increase
Atomic task time	Matlab	CPU	30% increase
Video quality	Media player	Network	15% decrease

setups. Hence, each site had 64 GB of RAM, a 100 Mbps duplex network bandwidth interface, and a scalable number of CPU cores that are each at 2 GHz. In the simulation, we varied the number of data center sites in the cloud, the number of CPU cores at each site, and the array of incoming VD request connection types. Each VD request connection was identified as belonging to one of user groups (i.e., Campus Computer Lab, Engineering Site, or Distance Learning Site) with varying resource requirements as defined by the respective utility functions. In order to account for the fact that thin-client protocols are affected by the network latency or physical distance between the end-user site and the cloud, we assumed that each VD request site was geo-located in close proximity to 2 data centers that had acceptable intermediate network latency. Hence, each connection was assigned a primary and secondary site preference by our simulation based on randomly selecting one of the considerations discussed in Section 3.3 that influence mapping of user desktop requests to distributed VDs. If U-RAM determined that neither the primary nor the secondary site had enough resources to successfully provision and place a VD connection request, the VD connection request was rejected.

For collecting the ‘VDs per core’ performance measurements in our simulation, we scaled up the number of CPU cores at each data center while all other resources remained constant. This metric highlights the inflection point at which adding more CPU cores to a data center no longer increases the capacity of that particular data center since one of the other resources (i.e., memory or network bandwidth) have been exhausted, and are the limiting factors. For deriving the ‘user connections quantity’, we calculate the total number of incoming VD request connections that have been handled by a VDC with *l* data centers after attaining the inflection point in the case of ‘VDs per core’ performance measurements. To collect each data point of these 2 metrics, we average the result from 200 runs in the simulation.

We compare cloud scalability performance of U-RAM with 4 different resource allocation models:

- *Fixed Resource Allocation Model* (*F-RAM*): in this scheme, each VD is overprovisoned and is given fixed resources that produce utility in Q_{excess} range. This is the most commonly used resource allocation method where the CSP does not have any actionable information about the resource requirements of VDs. The natural tendency is thus to provision fixed (identical) amount of overprovisioned resources to each VD regardless of its expected workload.
- *Network-aware Resource Allocation Model* (*N-RAM*): in this scheme, the CSP is aware of the Q_{max} required for network resources, but overprovisions Q_{excess} for system (RAM and CPU) resources due to lack of system-awareness information. The network-awareness can be obtained by a CSP for e.g., by using a thin-client protocol specification recommended as a rule-of-thumb by an application vendor.
- *System-aware Resource Allocation Model* (*S-RAM*): this scheme is the opposite of N-RAM in that Q_{max} is provisioned for the system resources and Q_{excess} is provisioned for the network resources. The system-awareness can be obtained by a CSP for e.g., by using CPU speed and memory size specifications recommended as a rule-of-thumb by an application vendor.
- *Greedy Resource Allocation Model* (*G-RAM*): in this scheme, we assume that the CSP is aware of the Q_{max} requirement in terms of both the system as well as the network resources based purely upon rule-of-thumb information. Consequently, this scheme greedily (a.k.a eagerly) provisions fixed amount of system and network resources to produce utility in Q_{max} range. Hence, it can be treated as a variant of our U-RAM without the adaptive resource sizing based on utility functions.
- Our proposed *Utility-Directed Resource Allocation Model* (*U-RAM*): Our proposed scheme uses a dynamic resource allocation strategy and operates a VD with utility in Q_{max} range while there are abundant resources available. When it is no longer possible for each VD to have Q_{max} resources, the VD resources are re-sized to a smaller value Q_{set}. This shrinking continues up to a Q_{min} value that corresponds to the least quality level acceptable to a user as defined in the SLAs.

5.2.1. VDs per core density results

Referring to Fig. 19 that shows the VDs handled successfully per data center with increasing number of cores, we can see that U-RAM outperforms the other schemes by supporting more VDs per core density. When VDs are provisioned using Q_{min} resources in the U-RAM scheme, the number of VDs supported at each data center increases with the number of cores per data center up to 112 VDs, which inherently requires 68 cores. At this inflection point, the memory at each data center becomes the bottleneck. When provisioning VDs with Q_{max} resources using the
G-RAM scheme, the VDs per core density at each data center increases to 44 VDs requiring 32 cores. At this inflection point, the density was limited by the network bandwidth available at each data center. The N-RAM scheme was able to achieve as much density as G-RAM, and VDs were provisioned with Q_{min} network resources. Owing to this,
N-RAM exhausted the memory at each data center with spare capacity in the number of cores, which is useless to provision any VDs due to the memory limitation. The difference in the S-RAM and N-RAM scheme results highlights the importance of having awareness of both the system and network resource requirements in resource allocation schemes. Note that S-RAM performed slightly better than F-RAM, and initially performed as good as G-RAM but reached its inflection point due to network limitation. VDs in the F-RAM scheme were provisioned with Q_{excess} resources, and hence the network bandwidth at each data center became exhausted once 24 cores were provided.

5.2.2. User connections quantity results

Referring to Fig. 20 that shows the user connections handled successfully with increasing number of data centers, we can see that U-RAM outperforms the other

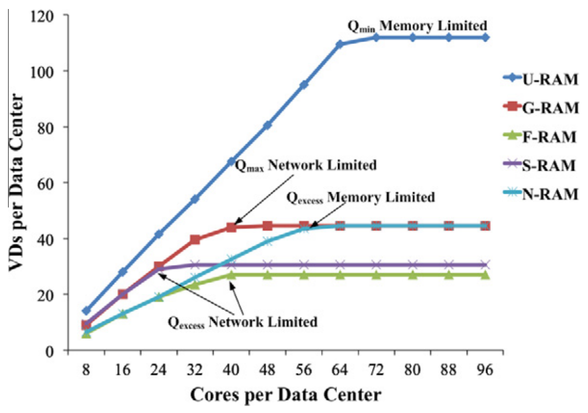

Fig. 19. Comparison of VDs per core density results.

schemes by supporting greater number of user connections. Note that the initial values and slopes shown are determined by the number of cores that are being used. The shown results are for the setting of 68 cores per site, which we observed from Fig. 19 is the number of cores needed to get maximum VD density per data center. Since U-RAM allows each data center to support a maximum of 112 VDs, a VDC consisting of 3 data centers could successfully support just over 336 user connections. Thus, in comparison with other schemes, U-RAM scales better as the number of data centers increase owing to the fact that the VD density per data center attainable is higher at every step.

6. Related work

Works such as [2–5] propose approaches for server-side resource allocation based on utility models and adaptation schemes to cope with system performance events and meeting SLA requirements. In [2], service utilization patterns are detected and a fuzzy controller is used to allocate server-side resources based on SLAs and system events such as service migration/replication. An adaptive resource allocation system based on a black-box control system model is proposed in [3] to handle applications with multi-tier quality SLAs without high resource utilization in data centers. Authors in [4] address the multi-tier application quality problem using a flexible queuing model and develop resource provisioning schemes based on per-tier utility predictions at small and large timescales. A more general resource allocation framework is proposed in [5] that aims to maximize a global utility function, which integrates SLA fulfillment and operating costs. The resource allocation problem is formulated as a constraint satisfaction problem and is deemed as an NP-Hard knapsack problem instance. Alternately, [6] formulates the general utility-based resource allocation problem as an integer program, again an NP-Hard problem, and attempts to solve the problem with a commercial solver on random problem cases. We remark that all of the above works target server-virtualization scenarios, where user workloads can be modeled as transactions that have predictable resource consumption characteristics.

Two works that are closely related to our specific resource allocation problem involving desktop virtualization are [9,10]. Both these works provide conceptual frameworks for leveraging measurements from performance benchmarking tools for VD resource sizing and placement. In [9], the resource allocation is entirely done by an online algorithm that is based on profiling active and idle time periods of desktop activity. Whereas in [10], the resource allocation on a virtual desktop is entirely done by an offline algorithm that is based on resource predictions from profiling user workloads on traditional desktops. To the best of our knowledge, our work is the first to propose a conceptual framework, an utility-driven model (U-RAM), as well as results from an actual implementation (i.e., VDBench results) to incorporate offline benchmarking to guide online resource allocation within VDCs. Further, we address meeting multi-tier application quality SLAs based on profiling applications within user groups, and correspondingly narrow the resource allocation problem to optimizing global utility within user desktop pools.

Our proposed U-RAM framework has similarities with resource allocation models in works such as [11,12]. In [11], an analytical model was developed for the RT-phone multimedia application considering end-to-end delay as the quality dimension with CPU as the resource to be allocated. In [12], a more extensive analytical model was developed to solve a resource allocation problem with multi-dimensions (i.e., track error, target drop probability, reliability) and multi-resources (i.e., radar bandwidth, short-term power, long-term power, CPU, memory) in radar tracking systems. In our U-RAM for VDCs, the resource allocation involves three resources (i.e., CPU, memory and network bandwidth) and two quality dimensions (i.e., timeliness, coding efficiency).

There have been earlier studies and toolkit developments related to performance measurement in thin-client systems. In [13], the slow-motion benchmarking technique was first developed to measure user QoE of web and video applications on thin-clients. Works such as [14] extended the use of slow-motion benchmarking for analyzing different design choices for thin-client implementations on wide-area networks. A novel video quality metric to

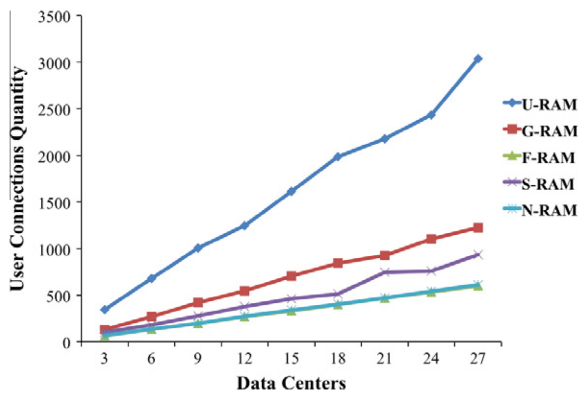

Fig. 20. Comparison of user connections quantity results.

compare performance of TCP and UDP based thin-clients was also developed in [14]. There have been user-workload scripting toolkits such as "Login VSI" [15] that automate performance measurements of application events (e.g., application open time) at the server-side in a controllable and repeatable manner. They are mainly suited for determining resource allocations based on server-side processing. In contrast, thin-client performance benchmarking toolkits such as [16,17] focus on recording and playback of keyboard and mouse events on the client-side, and do not consider synchronization with server-side events and network health measurements. Our VDBench toolkit leverages the earlier slow-motion benchmarking technique and user-workload scripting principles, and extends them with novel metrics (e.g., 'render time', 'bandwidth consumed') and introduces the concept of 'marker packets' to synchronize thin-client user events with server-side resource performance events.

7. Conclusion and future work

Our key contributions of this paper can be summarized as follows:

- We develop U-RAM that uses offline benchmarking based utility functions of system, network and human components to dynamically (i.e., online) create and place virtual desktops in resource pools at distributed data centers, all while optimizing resource allocations along *timeliness* and *coding efficiency* quailty dimensions.
- We propose an iterative algorithm for U-RAM (an NP-Hard problem) to optimize resource allocation with fast convergence based on combined utility functions.

Our work is unique as it studies the impact of increasingly constrained memory and network health conditions on the performance of various application tasks in a virtual desktop cloud environment. Further, our work is the first to propose a conceptual framework, an utility-driven model (U-RAM), as well as results from an actual implementation (i.e., VDBench results) to incorporate offline benchmarking to guide online resource allocation within VDCs. Our results from actual implementation show that both network-awareness (i.e., utility functions of thin-client protocols) as well as system-awareness (i.e., CPU and memory resource utility functions) are critical to improve VDC scalability. Also, optimization techniques for resource allocation (e.g., those that involve Kuhn–Tucker optimality conditions) when applied in VDCs need to consider creation of desktop pools based on utility functions of user groups.

By combining utility functions of both system and network components in U-RAM, we were able to optimally circumscribe the kind and amount of resources required to deliver adequate performance (i.e., satisfied user experience). This inturn significantly improved VDC scalability in terms of 'VDs per core density', and 'user connections quantity' in comparison with existing resource allocation methods. Using our results, CSPs looking to deploy thin-clients based VDCs will be able to greatly reduce the amount of costly guesswork and over-provisioning commonly encountered in this domain.

Future work for this paper could include leveraging context awareness of user tasks within VDCs in order to cache atomic tasks of highly interactive applications on thin-clients. Using such awareness, network round trips to the VDC can be avoided for user actions such as menu unfolding, and text-typing. Integration of the awareness will need to be non-intrusive on thin-client resources during context monitoring, and could leverage concepts such as application-specific "latency hiding" and "caching" [23] to deliver a comparable user experience to that of a traditional desktop.

References

[1] G. Briscoe, A. Marinos, Digital Ecosystems in the Clouds: Towards Community Cloud Computing, Proc. of IEEE DEST (2009).

[2] D. Gmach, S. Krompass, A. Scholz, M. Wimmer, A. Kemper, Adaptive Quality of Service Management for Enterprise Services, ACM Transactions on the Web 2 (8) (2008) 1–46.

[3] P. Padala, K. Shin, et al., Adaptive Control of Virtualized Resources in Utility Computing Environments, Proc. of ACM SIGOPS/EuroSys (2007).

[4] B. Urgaonkar, P. Shenoy, et al., Agile Dynamic Provisioning of Multi-Tier Internet Applications, ACM Transactions on Autonomous and Adaptive Systems 3 (1) (2008) 1–39.

[5] H. Van, F. Tran, J. Menaud, Autonomic Virtual Resource Management for Service Hosting Platforms, Proc. of ICSE Workshop on Software Engineering Challenges of Cloud Computing, 2009.

[6] T. Kelly, Utility-Directed Allocation, Proc. of Workshop on Algorithms and Architectures for Self-Managing Systems, 2003.

[7] L. Grit, D. Irwin, A. Yumerefendi, J. Chase, Virtual Machine Hosting for Networked Clusters: Building the Foundations for Autonomic Orchestration, Proc. of Workshop on Virtualized Technology in Distributed Computing, 2006.

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

[9] R. Bhowmik, A. Kochut, K. Beaty, Managing Responsiveness of Virtual Desktops using Passive Monitoring, Proc. of IFIP/IEEE IM (2009).

[10] K. Beaty, A. Kochut, H. Shaikh, Desktop to Cloud Transformation Planning, Proc. of IEEE IPDPS (2009).

[11] R. Rajkumar, C. Lee, J. Lehoczky, D. Slewlorek, A Resource Allocation Model for QoS Management, Proc. of IEEE RTSS (1997).

[12] J. Hansen, S. Ghosh, R. Rajkumar, J. Lehoczky, Resource Management of Highly Configurable Tasks, Proc. of IEEE IPDPS (2004).

[13] J. Nieh, S. Yang, N. Novik, Measuring thin-client performance using Slow-motion benchmarking, ACM Transactions on Computer Systems 21 (1) (2003) 87–115.

[14] A. Lai, J. Nieh, On The Performance Of Wide-Area Thin-Client Comput ing, ACM Transactions on Computer Systems 24 (2) (2006) 175–209.

[15] R. Spruijt, J. Kamp, S. Huisman, Login Virtual Session Indexer (VSI) Benchmarking, Virtual Reality Check Project – Phase II Whitepaper, 2010. (http://www.projectvrc.com)

[16] N. Zeldovich, R. Chandra, Interactive Performance Measurement with VNCplay, Proc. of USENIX Annual Technical Conference, 2005.

[17] J. Rhee, A. Kochut, K. Beaty, DeskBench: Flexible Virtual Desktop Benchmarking Toolkit, Proc. of Integrated Management (IM) (2009).

[18] C. Waldspurger, Memory Resource Management in VMware ESX Server, ACM Operating Systems Review 36 (2002) 181–194.

[19] C.H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Dover Publication, 1998. ISBN 0-486-40258-4.

[20] B. Schneiderman, C. Plaisant, Designing the User Interface: Strategies for Effective Human-Computer Interaction, Addison-Wesley Publication, 2005. ISBN-10: 0321197860.

[21] A. Kochut, Power and Performance Modeling of Virtualized Desktop Systems, Proc. of IEEE MASCOTS (2010).

[22] C. Clark, K. Fraser, S. Hand, et al., Live Migration of Virtual Machines, Proc. of USENIX NSDI (2005).

[23] B. Vankeirsbilck, P. Simoens, et al., Bandwidth Optimization for Mobile Thin Client Computing through Graphical Update Caching, Proc. of ATNAC (2008).

Prasad Calyam received the BS degree in Electrical and Electronics Engineering from Bangalore University, India, and the MS and PhD degrees in Electrical and Computer Engineering from The Ohio State University, in 1999, 2002, and 2007 respectively. He is presently a Principal Investigator and Senior Systems Developer/Engineer at the Ohio Supercomputer Center/OARnet, The Ohio State University. His current research interests include multimedia networking, cyber security, cyberinfrastructure systems, and network management.

Rohit Patali received the BS degree in Computer Engineering from University of Mumbai in 2009. He is currently pursuing his MS degree in Computer Science and Engineering at The Ohio State University, and is a Graduate Fellow at the Ohio Supercomputer Center. His current research interests include multimedia networking and distributed systems.

Alex Berryman is currently pursuing his BS degree in Aeronautical and Astronautical Engineering at The Ohio State University, and is a National Science Foundation REU student at the Ohio Supercomputer Center. His current research interests include computer and network virtualization, network performance measurements, and distributed systems.

Albert M. Lai received the BS and MS in Computer Science at Columbia University's Fu Foundation School of Engineering and Applied Science in 2000 and 2001, respectively. He received his PhD in 2007 from the Department of Biomedical Informatics in the Graduate School of Arts and Sciences at the Columbia University Medical Center. He is currently an Assistant Professor in the Dept. of Biomedical Informatics at The Ohio State University, and also the Associate Director of the Biomedical Informatics Program for the Center for Clinical and Translational Science. In addition, he is a scientific advisor for the Center for IT Innovation in Healthcare at The Ohio State University Medical Center. His research interests include data mining, telemedicine, clinical and translational informatics, usability of clinical systems, and thin client computing.

Rajiv Ramnath is Director of Practice at the Collaborative for Enterprise Transformation and Innovation (CETI) and the Associate Director for the Institute of Sensing Systems at The Ohio State University. He was formerly Vice President and Chief Technology Officer at Concentus Technology Corp., in Columbus, Ohio, and led product-development and government-funded R&D – notably through the National Information Infrastructure Integration Protocols program funded by Vice President Gore's ATP initiative. He is now engaged in developing industry-facing programs of applied R&D, classroom and professional education and technology transfer. His expertise ranges from wireless sensor networking and pervasive computing to business-IT alignment, enterprise architecture, software engineering, e- Government, collaborative environments and work-management systems. He teaches software engineering at OSU and is involved in industry-relevant and inter-disciplinary curriculum development initiatives. Ramnath received his Doctorate and Masters' degrees in Computer Science from OSU and his Bachelors degree in Electrical Engineering from the Indian Institute of Technology.