VDC-Analyst: Design and Verification of Virtual Desktop Cloud Resource Allocations

Prasad Calyam, Sudharsan Rajagopalan, Sripriya Seetharam, Arunprasath Selvadurai, Khaled Salah, Rajiv Ramnath

University of Missouri, The Ohio State University, USA; Khalifa University, UAE
Email: {calyam,ssn68}@missouri.edu, {rajagopalan.32, selvadurai.1, ramnath.6}@osu.edu, khaled.salah@kustar.ac.ae

Abstract—One of the significant challenges for Cloud Service Providers (CSPs) hosting “virtual desktop cloud” (VDC) infrastructures is to deliver a satisfactory quality of experience (QoE) to the user. In order to maximize the user QoE without expensive resource overprovisioning, there is a need to design and verify resource allocation schemes for a comprehensive set of VDC configurations. In this paper, we present “VDC-Analyst”, a novel tool that can capture critical quality metrics such as Net Utility and Service Response Time, which can be used to quantify VDC platform readiness. This tool allows CSPs, researchers and educators to design and verify various resource allocation schemes using both simulation and emulation in two modes: “Run Simulation” and “Run Experiment”, respectively. The Run Simulation mode allows users to test and visualize resource provisioning and placement schemes on a simulation framework. Run Experiment mode allows testing on a real software-defined network testbed using emulated virtual desktop application traffic to create a realistic environment. Results from using our tool demonstrate that a significant increase in perceived user QoE can be achieved by using a combination of the following techniques incorporated in the tool: (i) optimizing Net Utility through a “Cost-aware Utility-Maximal Resource Allocation Algorithm”, (ii) estimating values for Service Response Time using a “Multi-stage Queuing Model”, and (iii) appropriate load balancing through software-defined networking adaptations in the VDC testbed.

Keywords—Virtual desktop cloud, Cloud simulator, Utility-directed resource allocation, Software-defined networking, Cloud testbed experiment

I. INTRODUCTION

In recent years, desktop virtualization through Desktop-as-a-Service (DaaS) offerings has become an increasingly obvious choice for enterprises because of: the seamlessly realized benefits of centralized desktop administration, standard security configurations, data protection and simplified management of thin-clients on “virtual desktop clouds” (VDCs) [1]. DaaS offerings can cater to the needs of user communities such as virtual classroom labs in education, remote volume visualization in biomedical applications, and virtual modeling in manufacturing. As more users switch from traditional desktop environments to VDCs, Cloud Service Providers (CSPs) are faced with unique scalability challenges in elastically handling rapid growth by resource sharing and load balancing of computation and network resources, so as to provide acceptable user quality of experience (QoE) in order to maintain their service level agreements (SLAs), while also minimizing costs.

In particular, perceived user QoE in virtual desktops (VDs) is sensitive to network degradation and cannot tolerate bursty cross-traffic patterns [2] [3]. Most of the earlier research avoided focusing on network optimization techniques for handling user QoE deterioration because it is challenging to control routing dynamics in the Internet. The research community has been developing resource allocation schemes such as [4] and [5] in order to increase the user experience of VDCs based on thin-client protocol adaptations. However, given the complexity of a cloud system, infrastructure and tools to test or simulate such schemes are not easily accessible. There are many simulation tools [6] - [13] which either mimic the cloud infrastructure or allow researchers to work on real cloud platforms. Nevertheless, these tools are focused on testing application-oriented resource allocation policies at the server side, and there is a dearth of tools for simulating and emulating VDC systems to serve the salient needs of CSPs looking to offer DaaS, as well as educators and researchers of VDC resource allocation schemes.

In this paper, to address the lack of tools for design and verification of human, network, and system aware resource allocation schemes in VDCs, we have developed a novel tool called, “VDC-Analyst”. VDC-Analyst allows estimation of two novel user QoE metrics: Net Utility and Service Response Times (SRTs). Net Utility of a VDC system not only measures the server side utility but also evaluates the VDC system under realistic network settings that can impact the overall user QoE. End-to-end system and network latency also affects the SRTs of users’ VD requests by causing a considerable time lapse between the VD request and the provisioning of the VD from a desktop pool within a data center. The components of this SRT include network path determination, resource usage estimation, data center selection and finally, the VD instantiation.

To estimate the Net Utility of all the VD requests in a VDC, we have developed a new scheme called the “Cost-Aware Utility-Maximal Resource Allocation Algorithm” that computes the metric in a simulation framework. The scheme also enables software-defined networking capabilities to control the routing dynamics for improving user QoE on a real network emulation testbed with OpenFlow protocol [14] compatible switches. To estimate the SRT metric of incoming VD requests in a VDC system, we have developed a “Multi-stage Queuing Model” with an embedded Markov chain process.

The above metrics estimation development and their integration into the VDC-Analyst tool architecture builds and significantly extends our prior works on VD resource provisioning and placement optimization schemes, as well as the slow-motion based thin-client performance benchmarking for VDC environments [3] [15] [16].

This material is based upon work supported by the National Science Foundation under award numbers CNS-1050225, CNS-1205658, and VMware. 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 or VMware.
Our ultimate aim in developing the VDC-Analyst tool is to realize maximization of the user QoE without expensive resource over-provisioning by allowing design and verification of resource allocation schemes for a comprehensive set of VDC configurations. The tool provides 'control knobs' for educators and researchers to fine tune their system and network resource allocation policies and study trade-offs in design decisions. By executing several test cases using the tool’s two modes: “Run Simulation” and “Run Experiment”, respectively - we show that an iterative synergy between simulation and realistic experiment environments is needed to assess the readiness of a VDC infrastructure. The Run Simulation mode allows researchers to test and visualize resource provisioning and placement schemes on a modular, extensible, and repeatable simulation framework; the Run Experiment mode leverages a real software-defined OpenFlow-based network testbed within the Global Environment for Network Innovations (GENI) infrastructure [18] [19] to emulate VD application traffic for testing, visualization and load-balancing of resource allocations in a realistic environment with network cross-traffic.

The remainder of the paper is organized as follows: Section II discusses related work on cloud simulators. Section III discusses VDC-Analyst’s architecture. Section IV describes the implementation goals, workflow and User Interface (UI) features of the tool. The schemes for metric estimations in the VDC-Analyst tool are described in Section V. Section VI presents exemplar use cases of VDC-Analyst in Run Simulation and Run Experiment modes for research and education purposes. Section VII concludes the paper.

II. RELATED WORK

Currently, there are several tools such as [6] - [13], which simulate cloud infrastructures and provide testbeds for simulation and verification of resource allocation schemes. The CloudSim toolkit [6] offers a cloud computing framework for modeling and simulation of resource allocation policies on cloud platforms. It provides ways to model virtual machine allocations, cloud markets, dynamic workloads and other policies with respect to application services. NetworkCloudSim [7] is an extension of CloudSim that features network models across data centers to simulate complex application services and improve scalability by maximizing network bandwidth.


While all of the above work on tools is mainly targeted towards transaction-oriented data clouds, VDC-Analyst is designed to test and validate resource allocation schemes relevant to virtual desktop applications (e.g., MS Word, Media Player, Matlab) within VDC infrastructures, where user QoE is highly sensitive to system and network performance fluctuations. The VDC-Analyst tool provides Run Simulation and Run Experiment modes to plugin and test resource allocation schemes in both environments on a single platform, which many existing tools do not support. Another novelty of our work is that we harness software-defined networking principles using the OpenFlow protocol to monitor the network health and adapt the network allocation mechanism based on user QoE considerations, to maximize the Net Utility and SRT of VDC infrastructures. Nevertheless, the tool also provides an intuitive UI to select input parameters such as VD requests load, cross traffic and faults and visualize output graphs to compare varied system and network states, similar to some of the above related works.

III. VDC-ANALYST ARCHITECTURE

The architecture of VDC-Analyst as depicted in Figure 1 enables interactions between the data plane and control plane to estimate user QoE metrics of VD requests in a VDC. The main module of VDC-Analyst is the ‘Unified Resource Broker’ (URB). The URB is the ‘brain of the VDC’ and is responsible for handling all optimization and routing decisions across the VDC infrastructure. The ‘Logic Module’ within URB enables large-scale simulations involving initial resource allocations, and subsequent calculation of utility changes that determine the Net Utility measurement across data centers. Net Utility calculation is performed based on the selected resource allocation scheme (explained in Section V) to place VDs in a cost-aware and utility-maximal manner. The ‘Logic Module’ within URB also features a multi-stage finite queuing model (explained in Section V) that captures the various stages of a VD request in a VDC system and estimates the SRT in serving individual VD requests. VD user profiles and real time application usage is monitored using the Thin-client Performance Benchmarking module developed in our prior work [17].

In the UI layer, workload generation module handles inputs such as VD request load, and cross-traffic level settings relevant to Run Simulation and Run Experiment modes (detailed
workflows of the modes are explained in Section IV). The output contains the VD Placement Allocations that indicate every VD’s placement at different data centers based on the provisioning and placement scheme selection in the input. The Result Graphing Module is responsible for displaying the SRT and Net Utility graphs for a given resource allocation scheme. The Net Utility and SRT obtained in Run Simulation mode are evaluated under a realistic environment in Run Experiment mode with network cross-traffic and fault-conditions to measure the VDC infrastructure’s performance impact on perceived user QoE for VD applications. A reduced user QoE in Run Experiment mode in turn synergistically guides further enhancements in resource allocation schemes in Run Simulation mode.

In Run Experiment mode, the ‘Service Applications’ within URB interact with the physical infrastructure resources (i.e., data centers, thin-clients, OpenFlow switches) to instantiate the VDs within compute nodes and also determine the network paths from the thin client node to the corresponding VDC data center. The Hypervisor Interface is responsible for securely provisioning system resources at the data center with appropriate sizing of CPU, memory and network interface bandwidth settings. The OpenFlow controller application comprises the flow identification and load balancer modules that are responsible for initializing the flow tables of VD network paths, and for reconfiguring them dynamically on switches in cases such as cross-traffic congestion or device failures.

IV. VDC-ANALYST IMPLEMENTATION

In this section, we first outline VDC-Analyst development goals followed by workflow implementations for simulation and emulation modes. Next, we describe the implementation details of the individual modules in the UI.

A. Goals

VDC-Analyst has been developed based on a self-contained, unified framework to toggle between simulation and emulation. Some of the goals we set for ourselves and addressed were: the tool should provide modularity and extensibility in adding resource allocation schemes without configuration or worrying about details of the underlying components; the tool should be robust in order to perform repeatable simulations at-scale and emulations for diverse input parameters; the tool should serve educational and research use cases that involve experimentation with different resource allocation schemes in order to measure the two important user QoE metrics of VDC infrastructures viz., Net Utility and SRTs; the tool should provide a simple intuitive UI allowing users to configure input values and visualize output values on a single view in both Run Simulation and Run Experiment modes.

B. Workflow

In this section, we describe the VDC-Analyst workflow shown in Figure 2, which illustrates the synergy between Run Simulation and Run Experiment modes. The synergy is created by the exchange of Net Utility and SRT metrics obtained in Run Simulation mode, which guides the resource configuration and VD placements in the Run Experiment mode. Subsequently, the perceived user QoE measurements from the real VDC environment can be used as feedback to adapt for dynamic cases where user QoE is below acceptable thresholds. The adaptation can invoke new simulations that re-evaluate the previous resource configurations and VD placements in order to improve Net Utility and SRT, and so on.

1) Workflow for Run Simulation: The workflow for Run Simulation is as follows: Step-1 Input Selection: Input parameters for VD requests, faults and cross traffic, and provisioning and placement schemes are selected through the tool’s UI panel; Step-2 Resource Allocation: On triggering the Run Simulation, the selected provisioning scheme computes the amount of resource allocation required for the VD request. The placement scheme then determines the data center location to host the VD request; Step-3 Net Utility Calculation: The URB allocates the required number of resources on the selected data center that gives rise to maximal utility. Once VD resources are allocated, the URB performs optimization decisions across data centers to reduce operational costs and increase server side utility. The Net Utility is calculated as a summation of utility values at all data centers. The Net Utility of system is then plotted against the current run along with Net Utility values of previous runs; Step-4 SRT Calculation: The SRT for every VD request is estimated using the tool’s analytical model and plotted on the UI to observe the overall performance of the VDC system.

2) Workflow for Run Experiment: The workflow for Run Experiment is as follows: Step-1 Node Reservation: Thin-client Wide-Area ProtoGENI (WAPG) nodes on the GENI infrastructure are reserved and configured for triggering VD requests and cross traffic; VDC data centers are instantiated with VD pools. The OpenFlow controller application installs the initial flow tables for OpenFlow switches that make up the VDC network; Step-2 Network Path Determination: Once VD requests are initiated from the thin-client nodes, Run Experiment triggers the URB to allocate network paths on the GENI infrastructure and resources on VDC data centers using the selected provisioning and placement scheme; Step-3 Flow Installation: The URB communicates the routing information to the OpenFlow controller through marker packets that are used to identify flows for forwarding decisions [3]. The controller programs the OpenFlow switches to route the VD data packets between the data center and thin-client and vice versa; Step-4 End-to-End Monitoring: The URB then allocates resources to provision the VD on the selected data center and monitors user QoE through the Thin-client Performance Benchmarking module. In case, the interactive response time of VD applications between the VD thin-client and data center goes below a certain benchmarked threshold, the URB tries to re-evaluate the resource allocations on the data center while maintaining the Net Utility of the system. In the event of network cross traffic bursts or down links, the URB detects the network degradation and switches the VD application path by communicating with the OpenFlow controller. Thus, the URB performs load balancing on the network and maximizes the Net Utility of the VDC system.

C. User Interface

The VDC-Analyst tool’s UI shown in Figure 3 (built using the Matlab package) enables the performance as well as resource allocation analysis and visualization features. We remark that the back-end scripts are independent of the UI technology, and can be reused as plug-ins with other integrated
development environments such as Eclipse [20]. The provisioning and placement schemes to be simulated or emulated are selected as part of the Cloud Broker panel [Frame a) of Figure 3]. The input parameters including VD requests load, faults and cross traffic can be set through the Workload Generator [Frame b) of Figure 3]. Users can either select Run Simulation or Run Experiment mode as shown in Run modes panel [Frame c) of Figure 3]. The Status Window [Frame d) of Figure 3] displays the current steps executed by the tool in Run Simulation or Run Experiment mode.

The Resource Allocation Location graph of VD requests and data centers on the UI gives a holistic understanding of placement of VDs across the data centers [Frame e) of Figure 3]. This graph helps us gain insights on how different placement schemes allocate VD requests across data centers to maximize Net Utility. The color code associated with VD requests represents the profile type of VD user. In our work, we have assumed VD requests can arise from three kinds of user profiles - a student in campus computer lab, a distance learning user or a scientist in Engineering Site, each having specific resource requirements needed for satisfactory QoE of their VD application sets. For example, distance learning user will use videoconferencing, and hence needs higher network bandwidth than the student in a campus computer lab. The tool also plots the Net Utility graph for the past five runs [Frame f) of Figure 3]. The graph helps us compare different schemes under given set of VD requests load, cross traffic and faults. The graph also helps us analyze an individual scheme subjected to different VD requests load, cross traffic and faults which is a characteristic test case in a real world VDC scenario. The last panel [Frame g) of Figure 3] consists of SRTs plotted against increasing VD requests load on the VDC system to measure the system’s behavior in serving multiple VD requests.
V. MODELS AND ALGORITHMS USED IN VDC-ANALYST


Our cost-aware utility-maximal resource allocation algorithm builds upon and significantly extends our provisioning schemes (i.e., Utility-directed Resource Allocation Model (U-RAM), Fixed Resource Allocation Model (F-RAM), Economics-directed Resource Allocation Model (E-RAM)) and placement schemes (i.e., ‘Least Load’, ‘Least Latency’, ‘Least Cost’ and ‘Least Joint’) from our prior works [15] and [16]. The algorithm extensions to estimate Net Utility in a VDC infrastructure are more suitable for OpenFlow controller application development in the Run Experiment mode. The detailed pseudocode of this algorithm is shown in Algorithm 1. The terminologies and assumptions described in the algorithm are summarized as follows: VDC consists of \( \{L_1, L_2, ..., L_l\} \) number of data centers. Every data center contains corresponding \( d \) virtual desktops \( \{v_1, v_2, ..., v_d\} \) and \( m \) resources \( \{R_1, R_2, ..., R_m\} \) for CPU, memory and network bandwidth allocations. Also, \( v_{di} \) is the \( i \)-th VD request and \( n \) is the number of VD requests. Further, \( U_i \) gives the utility at data center \( L_i \) and \( U_{net} \) represents Net Utility value of the VDC infrastructure and can be defined as:

\[
U_{net} = \sum_{i=1}^{l} U_i
\] (1)

As we do not know the nature of arrival of random VD requests, we provision the VDs sequentially in an online manner by executing the algorithm to determine the data center location \( L_i \) and the resource \( R_{set} \) required by the VD request \( v_{di} \). The algorithm chooses the network path providing maximum end-to-end bandwidth between the thin-client node and data center in an opportunistic manner. In a software-defined network, a single physical network can be broken down into several logical networks \( \{P_{i,1}, P_{i,2} ..., P_{i,m}\} \) available from the thin-client’s OpenFlow switch \( i \) to every data center where, each path is a set \( \{I_1, I_2 ... I_k\} \) of interconnects that forms the path. Each of these logical network paths can be visualized as a ‘packet stream processor’. Proper path selection and load balancing amongst these packet stream processors can help provide better SRTs and optimized bandwidth usage for VD applications. We choose the path that provides maximum standard deviation from the required bandwidth usage \( B \) as the candidate path for a given data center.

The data center \( L_i \) selection is similar to the Hill Climbing Algorithm implemented in [21]. \( U_{i,new} \) and \( U_{i,cur} \) represent the new and current utility values at a data center \( L_i \), and \( \Delta U_i \) represents the increase in utility at \( L_i \). The \( U_{i,new} \) at \( L_i \) is calculated using the utility function \( U_p(R_{set}) \) for \( v_{di} \) given \( P_{i,k} \) based on the U-RAM scheme. If the \( U_{i,new} \) is greater than the cost involved in reserving the resources on the data center (expressed as a factor of amount of resources needed) and the new \( \Delta U_i \) is greater than its current local value, \( \Delta U_i \) is updated as the current maximum utility increase. Once the above steps are performed iteratively for all the available data centers, the data center \( L_i \) that gives rise to maximal \( \Delta U_i \) is selected for placing the VD request.

Algorithm 1 Cost-Aware Utility-Maximal Resource Allocation Algorithm for a given VD request

1: Input: VD request \( v_{di} \), Utility function \( U_p(R) \) from U-RAM scheme, list of available data centers \( \{L_1, L_2, ..., L_l\} \) each containing \( d \) virtual desktops \( \{v_1, v_2, ..., v_d\} \) and \( m \) resources \( \{R_1, R_2, ..., R_m\} \), list of loop free paths \( \{P_{i,1}, P_{i,2} ..., P_{i,k}\} \), predicted bandwidth usage \( B \) for \( v_{di} \).
2: Output: Resource allocation \( R_{i,di} \) at data center \( L_i \) and path \( P_{i,k} \) for new VD request \( v_{di} \) that maximizes raise in utility \( \Delta U_i \)
3: begin procedure
4: \( max \Delta U = 0 \)
5: for each data center \( L_i \) do
6: /*Step-1: Network path determination*/
7: for each path \( P_{i,k} \) to data center \( L_i \) do
8: if \( \forall I_j \in P_{i,k} \) bandwidth at \( I_j \geq B \) then
9: Add \( P_{i,k} \) to the candidate paths set \( C_p \)
10: end if
11: end for
12: Select path \( P_{i,k} \) which provides the maximum Standard Deviation from \( B \) among all \( P_{i,k} \) ∈ \( C_p \)
13: /*Step-2: Utility estimation*/
14: \( U_{i,new} = U_p(R_{i,di}) \) for \( v_{di} \) at \( L_i \) given path \( P_{i,k} \)
15: /*Step-3: Data center selection*/
16: if \( U_{i,new} > \) Cost involved in allocating \( v_{di} \) at \( L_i \) then
17: \( \Delta U_i = U_{i,new} - U_{i,cur} \)
18: if \( \Delta U_i > max \Delta U \) then
19: \( max \Delta U = \Delta U_i \)
20: end if
21: end if
22: end for
23: Place \( v_{di} \) at \( L_i \) which gives \( max \Delta U \)
24: /*Step-4: VD instantiation*/
25: Allocate \( R_{i,di} \) to \( v_{di} \) on \( L_i \) with \( P_{i,k} \)
26: Update the selected data center utility \( U_{i,cur} = U_{i,new} \)
27: end procedure

B. Multi-stage Queuing Model for Service Response Time Estimation

Our analytical model for estimating SRT in the VDC application is a finite FCFS (First Come First Served) queuing model. The behavior of the VDC infrastructure for a given VD request can be represented as four stages of service in the queue namely: Stage-1) Network path Determination, Stage-2) Resource Usage Estimation, Stage-3) Data center Selection, and Stage-4) VD Instantiation. As shown in Figure 4, incoming VD requests get first queued in a buffer of size \( K-l \) where \( K \) represents the maximum number of VD requests served by the VDC infrastructure in \( n \) stages with each stage having a different mean service rate, i.e., \( \mu_1, \mu_2, ..., \mu_4 \). A new VD request enters the input queue and is processed only after the previous request has departed the queuing system, i.e., left Stage-4. The incoming VD requests are sequentially executed in the queue and the execution of all stages are mutually exclusive (i.e., if one of the stages is executing, no other stage will be executing concurrently). Also, in order to capture the incoming VD requests distribution over a period of time, we have assumed a Poisson arrival \( \lambda \) of VD requests.
We calculate the mean service time of the fourth stage (i.e., 1/μ₄) as the mean of the maximum of two service times: (i) the time taken for comparing a new VD request’s user profile with existing VDs in the desktop pool, and (ii) the time taken for creating a new VD image (when user profile does not match existing VDs) or the time taken for adding a new user to an already created VD pool (when user profile matches existing VDs). Both values are exponentially distributed with means of 1/μ₄ and 1/μᵢ, respectively. Hence, we have -

\[
1/\mu_4 = 1/\mu_u + 1/\mu_4 - 1/(\mu_u + \mu_i)
\] (2)

The analytical finite queuing model for a VDC infrastructure is based on embedded Markov chain shown in Figure 5 with a state space \( S = \{(k, n), 0 \leq k \leq K, 0 \leq n \leq 4\} \), where \( k \) denotes the number of requests in the VD application. State (0,0) represents the special case when the VD application is idle, i.e., the state of system idleness. State (k,n) represents the state where the application is busy executing service of stage \( n \) with \( k \) requests in the system.

Let \( P_{k,n} \) be the steady-state probabilities at State (k,n). The steady-state balance equations are shown for each state as follows:

\[
State(0,0) : 0 = -\lambda p_{0,0} + \mu_4 p_{1,4}
\] (3)

\[
State(1,0) : 0 = -(\lambda + \mu_n)p_{1,1} + \mu_{n-1}p_{1,n-1}, \quad (2 \leq n \leq 3)
\] (4)

\[
State(k,0) : 0 = -(\lambda + \mu_n)p_{k,0} + \mu_{n-1}p_{k,n-1}, \quad (2 \leq k \leq K - 1; \ 2 \leq n \leq 3)
\] (5)

\[
State(k,1) : 0 = -\mu_p p_{k,1} + \lambda p_{K-1,1}
\] (6)

\[
State(K,1) : 0 = -\mu_1 p_{K,1} + \lambda p_{K-1,1}
\] (7)

Therefore, the state probabilities of \( P_{k,n} \) can be expressed recursively in terms of \( p_{0,0} \) as follows:

From equation (3),
\[
p_{1,4} = \frac{\lambda}{\mu_4} p_{0,0}
\] (8)

From equation (4),
\[
p_{1,n-1} = \left(\frac{\lambda + \mu_n}{\mu_{n-1}}\right) p_{1,n}, \quad (2 \leq n \leq 3)
\] (9)

From equation (5),
\[
p_{k,n-1} = \left(\frac{\lambda + \mu_n}{\mu_{n-1}}\right) p_{k,n} - \left(\frac{\lambda}{\mu_{n-1}}\right) p_{k-1,n}, \quad (2 \leq k \leq K - 1; \ 2 \leq n \leq 3)
\] (10)

Obtaining \( P_{0,0} \) can be used to find all other state probabilities \( \{p_{k,n}; 1 \leq k \leq K, 1 \leq n \leq 4\} \).

The mean system throughput \( \gamma \) is basically the departure rate i.e., the rate at which the VD request finishes successfully after being processed in Stage-4:
\[
\gamma = \frac{\sum_{k=1}^{K} p_{k,4}}{\sum_{n=1}^{4} p_{K,n} p_{0,0}}
\] (14)

The departure rate \( \gamma \) can also be expressed as the effective arrival rate \( \lambda' \) which is \( \lambda(1 - P_{loss}) \). Therefore,
\[
\gamma = \lambda(1 - P_{loss})
\] (15)

Where \( P_{loss} \) is the loss (or blocking) probability. \( P_{loss} \) can be expressed as the probability of being in States (K, 1 - 4):
\[
P_{loss} = \sum_{n=1}^{4} p_{K,n}.
\] (16)

The mean number of VD requests in the system can be expressed as:
\[
E[K] = \sum_{k=1}^{K} \sum_{n=1}^{4} kp_{k,n}.
\] (17)

Using Little’s law, the mean SRT of a VD request spent in the system succeeding in entering the queue can be expressed as:
\[
W = \frac{E[K]}{\gamma} = \frac{1}{\gamma} \sum_{k=1}^{K} \sum_{n=1}^{4} kp_{k,n}.
\] (18)
VI. VDC-ANALYST USE CASES AND RESULTS

In this section, we first analyze the simulation results for a given provisioning and placement scheme in the Run Simulation mode. We then demonstrate and emulate our simulation results with an experiment on the GENI testbed in Run Experiment mode. Lastly, we discuss the tool’s use case as an educational curriculum for a classroom lab exercise.

A. Run Simulation Use Case

To simulate the VDC system under various real-world scenarios, we performed different combinations of provisioning and placement scheme tests with varying number of VD requests load, faults and cross traffic levels. Figure 6 shows the Net Utility graph for a sample set of 15 runs. The first four runs are the outputs of different placement schemes under U-RAM provisioning when there is no fault in the system, the VD requests load is 500 and cross traffic level in the network is set to 10. As observed, Least Latency and Least Joint schemes give rise to higher Net Utility compared to Least Cost and Least Load schemes. While Least Latency places VDs on the nearest data centers from request locations, Least Joint scheme considers the overall utility of the system by optimizing VD requests load on the data centers and gives rise to higher utility. Also, we observe that Least Cost placement scheme reduces the Net Utility since it allocates VDs with minimum number of resources and finally, Least Load scheme has the lowest utility since it allocates minimum VD requests load at every data center, reducing the overall system utility.

When we introduce faults in the system, the behavior of the four schemes are almost similar with only difference in the magnitude of Net Utility as observed from run numbers 5, 6, 7 and 8. Though the fault rate is same across all schemes, the magnitude at with each scheme has scaled down is different. If we closely look at the Least Latency and Least Joint placement scheme, there is a marginal difference in the utility values. The Least Joint placement scheme is more adaptable towards changes in the environment. Thus, Least Joint scheme out performs Least Latency scheme and could be preferable to use in a cloud environment when fault occurrence is more.

The run numbers 9, 10, 11 and 12 show the results of E-RAM and Fixed Resource Allocation Model algorithms. The Net Utility of E-RAM scheme is slightly less as compared to U-RAM. This is because the E-RAM scheme while maintaining an relatively biased fairness between the VDs of different user profiles, decreases the number of VD requests served per data center, which in turn marginally reduces the VDC utility. On the other hand, F-RAM utility is very low as very few VDs are placed at any given data center. Contrary to the above observation, utility of E-RAM and U-RAM in the subsequent runs 13 and 14 remains same for a VD requests load of 200 while the F-RAM utility has further scaled down. Equal utility values for U-RAM and E-RAM can be attributed to the reason that the data centers have sufficient resources to allocate maximum amount of CPU and memory for every VD request. The utility starts varying once some data centers exhaust their resources after a certain threshold of VD requests load is reached.

The URB can provision VDs in two different ways. If a previously created VD matches a new VD request’s user profile and is available for provisioning, then that VD will be provisioned by adding the new user profile to it. The URB takes about 1 minute (from our testbed experiences) on an average in performing the operation. If such a VD is not
available, a new VD will be created by booting up a fresh image for the user and the user profile is added to the VD for the request. In this case, the URB takes around 15 minutes (from our testbed experiences) approximately. Figure 7 shows SRTs for the two VD provisioning methods for a VD requests load of 500. Figure 7a shows a snapshot of the SRT window for provisioning on already existing VDs when a new VD request’s user group matches existing user profiles. We observe that the difference between the lowest provisioning time and the highest provisioning time is around 2 minutes. Similarly, Figure 7b shows a snapshot of SRT results for provisioning new VDs that do not match user group profiles. The difference is now less than 0.5 minutes. The reason can be attributed to the fact that, in the second case, the VD request spends more time in the request queue and hence the difference in provisioning times is less when compared to the first case.

B. Run Experiment Use Case

To emulate the Run Experiment mode in VDC-Analyst, we reserved the OpenFlow Topology of GENI infrastructure with OpenFlow switches at different university campus locations shown in Figure 8a. We had the VMLab data center at The Ohio State University (location: Columbus, Ohio) (running VMware ESXi hypervisor for creating VD pools) hosting popular applications (e.g., Excel, Internet Explorer, Media Player) as well as advanced scientific computing applications (e.g., Matlab, Moldflow), connected to the GENI OpenFlow network. A mirror of VMLab setup was at the Emulab data center at The University of Utah. Additionally, we reserved WAPG nodes PG48, PG49 in Stanford University; PG50 and PG51 in Rutgers University to act as cross-traffic generators. We used Floodlight [22] as the OpenFlow controller application. From the topology in Figure 8a, we can observe that the path between Stanford and Rutgers nodes overlap with the path between the thin-client node at Clemson and the VMLab data center. This was done to generate cross traffic using high bandwidth for disrupting the VD traffic between the thin-client node and the data center.

On triggering the Run Experiment with U-RAM and Least Cost as the provisioning and placement schemes in the Cloud Broker UI panel, Figure 8a shows the flow rules that were installed in the OpenFlow switch for the thin-client traffic. We initiated a VD request from the Clemson thin-client node. Once the VD resource based on the selected resource allocation schemes was reserved on the VMLab data center, we ran a high-definition (HD) wildlife video clip on Windows platform to show the Media Player capability. We observed that the video ran smoothly, without any lag or jitter and the Windows Graphical User Interface (GUI) was responsive. When we opened other applications in parallel, GUI interactions responded in a smooth manner. The graph in Figure 9a shows the average bandwidth consumed by the thin-client traffic in the network.

Next, we introduced cross traffic using the Iperf tool [23] in the network by setting the cross traffic level to 100 in the VDC-Analyst window and ran the experiment again with the
same provisioning and placement schemes. The shortest path chosen by the URB between the thin-client node and data center overlaps with the path between Stanford and Rutgers WAPG nodes used for cross traffic generation. The OpenFlow controller application installed the flow rules for the cross traffic in the OpenFlow network devices. Once the cross traffic flows were installed, the Iperf client and server at Stanford and Rutgers started to communicate with a traffic of 100 Mbps bandwidth, respectively. The cross traffic disrupted the thin-client traffic on the same path and made the thin-client node PG47 at Clemson to freeze, and its GUI became unresponsive. The path used by both these flows is shown in Figure 8a with the congestion path highlighted. Figure 9b shows the system characteristics - the flow tables in the OpenFlow switches with new entries for the cross traffic flows, the diminishing video quality at the thin-client site and the average bandwidth usage statistics as collected at WAPG nodes i.e., PG47 at Clemson (thin-client node) and PG50 at Rutgers (cross-traffic node).

We finally selected our cost-aware utility-maximal resource allocation scheme, kept the cross-traffic as 100 and ran the experiment again. With the selected scheme in place, load balancing was activated by the URB on the network and an alternate path in the multi-path GENI OpenFlow core was determined for cross-traffic generation. The new path did not overlap with the path used by the thin-client flow and is represented in Figure 8b. The new path was communicated to the OpenFlow network controller application by the URB and the controller installed the flow rules on the network devices for the path that was to be followed by the cross-traffic. Once the path was re-routed, the cross-traffic packets that had queued at the ports of network devices shared between the two traffic flows were flushed out. The thin-client became responsive again, and the video resumed. The GUI interaction was smooth and was similar to the working scenario when there was no cross-traffic in the network. Figure 9c shows the flow tables with new flow rules, and the average bandwidth usage statistics collected at the end-points of the two flows.

C. Educational Curriculum Use Case

Most of the current educational simulator tools allow students to test resource allocation schemes on simulation frameworks, but do not integrate real-world testbeds to measure the schemes’ actual performance and impact on user QoE. In this section, we describe how VDC-Analyst can be used as an educational curriculum in classroom lab exercises of cloud computing courses. The goal of the curriculum involving VDC-Analyst is to allow students to design and verify efficient resource allocation schemes for VDCs.

As part of the lab exercises, students can build their own resource allocation schemes and explore server, client and network-side intelligent adaptations using the tool to maximize user QoE for VDs. The tool can be used to measure the Net Utility and SRTs of the schemes under varying loads, faults and and cross-traffic in both modes of the tool. A low Net Utility or a high SRT indicates further refinement of resource allocation schemes and helps students build optimized schemes for VDCs. Students could also come up with their own quality metrics which can increase the VDC efficiency and integrate them easily into the tool.

In addition to developing resource allocation schemes, students could also explore various other optimizations of VDCs which contribute to maximizing the utility of schemes. On the server-side, students can design automated macro scripts for tasks having similar operations and reduce the end-to-end round trip time of thin-client control responses (i.e., implement latency hiding) during network bottleneck events. On the thin-client side, the tool can be extended to measure individual thin-client protocols’ (RDP, PCoIP, etc.) performance in serving VD requests and choose appropriate thin-client encodings which give rise to higher utility. Furthermore, on the network-side, students could build their own OpenFlow network controller applications and automate traffic flow management in order to reduce SRTs of VDCs. Some factors which can be considered for building adaptations within the controller application could involve: reduction of queuing delays, and avoidance of cyber-attacks. Students’ schemes could be graded based on relatively high or low Net Utility scores obtained using the VDC-Analyst tool. For example, a Net Utility score of 30+ could result in a Good grade, 20 – 30 range of Net Utility could result in a Fair grade, and < 20 Net Utility could result in a Poor grade.

VII. CONCLUSION

In this paper, we have presented a novel VDC-Analyst tool for simulation and emulation of large-scale VDC infrastructures measured through two important quality metrics Net Utility and Service Response Time. We demonstrated how load-balancing through “programmable” path switching based on perceived user QoE, can increase the performance of the VDC infrastructure by leveraging software-defined networking principles and the OpenFlow protocol. We described how VDC-Analyst can be used as an educational and research tool to measure user QoE metrics for different placement and provisioning schemes. Through our tool testing results, we demonstrated the need for different utility-directed resource allocation models as opposed to fixed resource allocation models, to substantially improve user QoE and VDC infrastructure scalability.

Our future work with VDC-Analyst is aimed at integrating the tool with emerging Virtual LAN (VLAN) technologies such as Virtual Private LAN service [24], Overlay Transport Virtualization [25], and Virtual Tenant Networks [26]. Since there are no open-solutions available that provide services such as predictable bandwidth performance for a virtual network along with traffic isolation, our planned work with VDC-Analyst aims to extend our method of leveraging OpenFlow for VDCs to accommodate virtual tenant network virtualization use cases for data-intensive and latency-sensitive VD applications.

REFERENCES