Dynamic Resource Allocation and Power Management in Virtualized Data Centers

Rahul Urgaonkar, Ulas C. Kozat, Ken Igarashi, Michael J. Neely
urgaonka@usc.edu, {kozat, igarashi}@docomolabs-usa.com, mjneely@usc.edu

***Abstract*—We investigate optimal resource allocation and power management in virtualized data centers with time-varying workloads and heterogeneous applications. Prior work in this area uses prediction based approaches for resource provisioning. In this work, we take an alternate approach that makes use of the queueing information available in the system to make online control decisions. Specifically, we use the recently developed technique of Lyapunov Optimization to design an online admission control, routing, and resource allocation algorithm for a virtualized data center. This algorithm maximizes a joint utility of the average application throughput and energy costs of the data center. Our approach is adaptive to unpredictable changes in the workload and does not require estimation and prediction of its statistics.**

***Index Terms*—Data Center Automation, Cloud Computing, Virtualization, Resource Allocation, Lyapunov Optimization**

I. Introduction and Related Work

There is growing interest in improving the energy efficiency of large-scale enterprise data centers and cloud computing environments. Recent studies [1] [2] indicate that the costs associated with the power consumption, cooling requirements, etc., of servers over their lifetime are significant. As a result, there have been numerous works in the area of power management for such data centers (see [3] and references therein).

At the data center level, application consolidation has been studied for reducing the total power consumption. Virtualization is a promising technique that enables consolidation of heterogeneous applications onto a fewer number of servers, while ensuring secure co-location between competing applications. This results in higher resource utilization and reduction in energy costs (by turning off extra servers). However, since multiple applications now contend for the same resource pool, it is important to develop scheduling algorithms that allocate resources in a fair and efficient manner. At the individual server level, techniques such as Dynamic Voltage and Frequency Scaling, low power P-states, etc. are available that allow a tradeoff between performance and power consumption. Several recent works (e.g., [4] [5]) have studied the problem of dynamically scaling the CPU speed for energy savings. In this work, we consider the problem of maximizing a joint utility of the long-term throughput of the hosted applications and the average total power expenditure in a virtualized data center. Our formulation unifies these two techniques for power control under a common analytical framework.

This work was performed when Rahul Urgaonkar worked as a summer intern at DOCOMO USA Labs.

This material is supported in part by the NSF Career grant CCF-0747525.

Dynamic resource allocation in virtualized data centers has been studied extensively in recent years. The work in [6]–[8] formulates this as a feedback control problem and uses tools from adaptive control theory to design online control algorithms. Such techniques use a closed-loop control model where the objective is to converge to a target performance level by taking control actions that try to minimize the error between the measured output and the reference input. While this technique is useful as a tracking problem, it cannot be used for utility maximization problems where the target optimal value is unknown. Work in [9] considers the problem of maximizing a joint utility of the profit generated by satisfying given SLA and the power consumption costs. This is formulated as a sequential optimization problem and solved using limited lookahead control. This approach requires building estimates of the future workloads. Much prior work on resource allocation is based on prediction-based provisioning and steady state queueing models [10]–[12]. Here, statistical models for the workloads are first developed using historical traces offline or via online learning. Resource allocation decisions are then made to satisfy such predicted demand. This approach is limited by its ability to accurately predict future arrivals.

In this work, we do not take this approach. Instead, we make use of the recently developed technique of Lyapunov Optimization [15] to design an online admission control, routing, and resource allocation algorithm for a virtualized data center. This algorithm makes use of the queueing information available in the system to implicitly learn and adapt to unpredictable changes in the workload and does not require estimation and prediction of its statistics. The technique of Lyapunov Optimization has been used to develop throughput and energy optimal cross-layer control algorithms in time-varying wireless networks (see [15] and references). This technique has certain similarities with the feedback control based approach as it also uses a Lyapunov function based analysis to design online control algorithms. In addition, this technique also allows stability and utility optimization to be treated in the same framework. Unlike works that use steady state queueing models, this approach takes into account the full effects of the queueing dynamics by making use of the queue backlog information to make online control decisions.

II. Basic Virtualized Data Center Model

We consider a virtualized data center with M servers that host a set of N applications. The set of servers is denoted by $\mathcal{S}$ and the set of applications is denoted by $\mathcal{A}$. Each server $j \in \mathcal{S}$

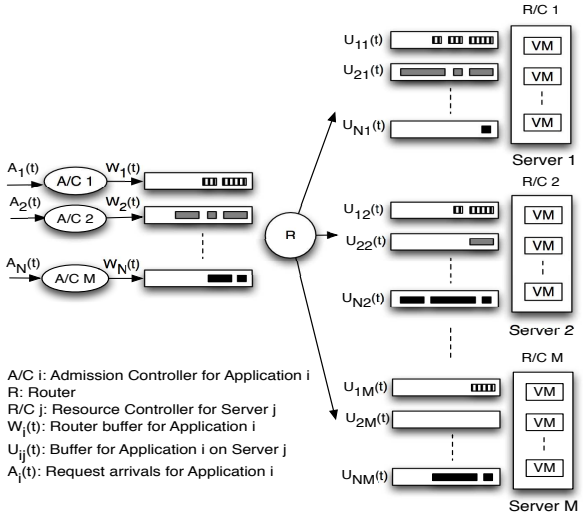

Fig. 1. Illustration of the Virtualized Data Center Architecture.

hosts a subset of the applications. It does so by providing a virtual machine (VM) for every application hosted on it. An application may have multiple instances running across different VMs in the data center. We define the following indicator variables for $i \in \{1, 2, ..., N\}, j \in \{1, 2, ..., M\}$:

$$a_{ij} = \begin{cases} 1 & \text{if application } i \text{ is hosted on server } j \\ 0 & \text{else} \end{cases}$$

For simplicity, in the basic model, we assume that $a_{ij} = 1 \forall i, j$, i.e., each server can host all applications. In general, applications may be multi-tiered and the different tiers corresponding to an instance of an application may be located on different servers and VMs. For simplicity, in the basic model we assume that each application consists of a single tier. These assumptions are relaxed in Sec. V where we discuss extensions to the multi-tier as well as inhomogeneous hosting scenario.

We assume a time-slotted system. Every slot, new requests arrive for each application i according to a random arrival process $A_i(t)$ that has a time average rate λ_i requests/slot. This process is assumed to be independent of the current amount of unfinished work in the system and has finite second moment. However, we do not assume any knowledge of the statistics of $A_i(t)$. For example, $A_i(t)$ could be a Markov-modulated process with time-varying instantaneous rates where the transition probabilities between different states are not known. This models a scenario with *unpredictable* and *time-varying* workloads.

A. Control Architecture

Our control architecture for the virtualized data center consists of three components as shown in Fig. 1. Every slot, for each application $i \in \mathcal{A}$, an *Admission Controller* determines whether to admit or decline the new requests. The requests that are admitted are stored in the Router buffer before being routed to one of the servers hosting that application by the *Router*. Each server $j \in \mathcal{S}$ has a set of resources $\mathcal{W}_j$ (such as CPU, disk, memory, network resources, etc.) that are allocated to the VMs hosted on it by its *Resource Controller*. In practice, this Resource Controller resides on the host Operating System (Dom0) of each virtualized server. The control options available to the Resource Controller are discussed in detail in Sec. II-C. For simplicity, in the basic model, we assume that the sets $\mathcal{W}_j$ contain only one resource. Specifically, we focus on case where the CPU is the bottleneck resource. This can happen, for example, when all applications running on the servers are computationally intensive. Examples of such applications include Hadoop, MapReduce and video transcoding. Our formulation can be extended to treat the multiple-resource case by representing them as a vector of resources and appropriately redefining the control options and expected service rates. All servers in the data center are assumed to be resource constrained. Specifically, in the basic model, we focus on CPU frequency and power constraints. This is discussed in detail in Sec. II-B.

B. CPU Power-Frequency Relationship

Modern CPUs can be operated at different speeds at runtime by using techniques such as Dynamic Frequency Scaling (DFS), Dynamic Voltage Scaling (DVS), or a combination Dynamic Voltage and Frequency Scaling (DVFS). These techniques result in a *non-linear* power-frequency relationship. For example, Fig. 2 shows the power consumed by a Dell PowerEdge R610 server for different operating frequencies and utilization levels. This curve was obtained by running a CPU intensive application at different CPU frequencies and utilization levels and measuring the power consumption. We observe that at each utilization level, the power-frequency relationship is well-approximated by a quadratic model, i.e., $P(f) = P_{min} + \alpha(f - f_{min})^2$. Similar results have been observed in recent works [4] . In our model, we assume that CPUs follow a similar non-linear power-frequency relationship that is known to the Resource Controllers. The CPUs can run at a finite number of operating frequencies in an interval $[f_{min}, f_{max}]$ with an associated power consumption $[P_{min}, P_{max}]$. This allows a tradeoff between performance and power costs. All servers in our model are assumed to have identical CPU resources.

Additionally, the servers may be operated in an inactive mode (such as P-states, CPU hibernation, or turning OFF) in order to further save on energy costs. This can be advantageous if the workload is low. Indeed, we note from Fig. 2 that the minimum power P_{min} required to maintain the server in the active state is typically substantial. It can be as high as 65% of P_{max} as reported in other works [1]. Therefore, turning idle servers to OFF mode, or to some low power hibernation state, can yield significant savings in power consumption. While an inactive server does not consume any power, it also cannot provide any service to the applications hosted on it. We thus assume that, in any slot, new requests can only be routed to active servers. Inactive servers can be turned active to handle increases in workload.

Since turning servers ON/OFF frequently may be undesirable (for example, due to hardware reliability issues), we will focus on frame-based control policies in which time is divided

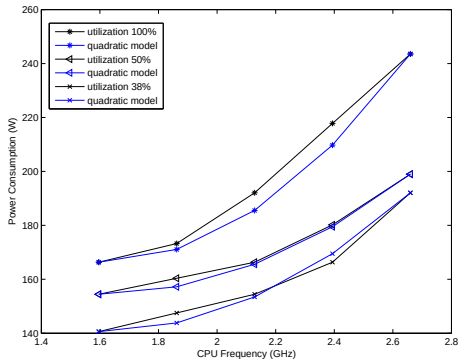

Fig. 2. Power vs CPU frequency for a Dell PowerEdge R610 server. The utilization value is the measured CPU utilization at a given CPU frequency and loading level during the experiment. Also note the significant idle power.

into frames of length T slots. The set of active servers is chosen at the beginning of each frame and is held fixed for the duration of that frame. This set can potentially change in the next frame as workloads change. We note that while this control decision is taken at a slower time-scale, other resource allocations decisions (such as admission control, routing, CPU frequency scaling and resource allocation at each active server) are made every slot. The choice of an appropriate value for T is an implementation issue. We do not focus on optimizing this parameter in this work. The choice of T affects a complexity-utility tradeoff as discussed in Sec. IV-B.

C. Queueing Dynamics and Control Decisions

Let $A_i(t)$ denote the number of new request arrivals for application i in slot t. Let $R_i(t)$ be the number of requests out of $A_i(t)$ that are admitted into the Router buffer for application i by the Admission Controller. We denote this buffer by $W_i(t)$. We assume that any new request that is not admitted by the Admission Controller is declined. This can easily be generalized to the case where arrivals that are not immediately accepted are stored in a buffer for future admission decision. Thus, for all i, t, we have:

$$0 \leq R_i(t) \leq A_i(t) \tag{1}$$

Let $R_{ij}(t)$ be the number of requests for application i that are routed from its Router buffer to server j in slot t. Then the queueing dynamics for $W_i(t)$ is given by:

$$W_i(t+1) = W_i(t) - \sum_j R_{ij}(t) + R_i(t) \tag{2}$$

Let $\mathcal{S}(t)$ denote the set of active servers in slot t. For each application i, the admitted requests can only be routed to those servers that host application i and are active in slot t. Thus, the routing decisions $R_{ij}(t)$ must satisfy the following constraints every slot:

$$R_{ij}(t) = 0 \text{ if } j \notin \mathcal{S}(t) \text{ or } a_{ij} = 0 \tag{3}$$

$$0 \leq \sum_{j \in \mathcal{S}(t)} a_{ij} R_{ij}(t) \leq W_i(t) \tag{4}$$

Every slot, the Resource Controller allocates the resources of each server among the VMs that host the applications running on that server. This allocation is subject to the available control options. For example, the Resource Controller may allocate different fractions of the CPU (or different number of cores in case of multi-core processors) to the VMs in that slot.[1] The Resource Controller may also use available techniques such as DFS, DVS, DVFS, etc. to modulate the current CPU speed which affects the CPU power consumption. We use $\mathcal{I}_j$ to denote the set of all such control options available at server j. This includes the option of making server j inactive (so that no power is consumed) if the current slot is the beginning of a new frame. Let $I_j(t) \in \mathcal{I}_j$ denote the particular control decision taken at server j in slot t under any policy and let $P_j(t)$ be the corresponding power consumption. Then, the queueing dynamics for the requests of application i at server j follows:

$$U_{ij}(t+1) = \max[U_{ij}(t) - \mu_{ij}(I_j(t)), 0] + R_{ij}(t) \tag{5}$$

where $\mu_{ij}(I_j(t))$ denotes the service rate (in units of requests/slot) provided to application i on server j in slot t by taking control action $I_j(t)$. We assume that, for each application, the expected value of this service rate as a function of the control action $I_j(t)$ is known for all $I_j(t) \in \mathcal{I}_j$. This can be obtained by application profiling and application modeling techniques (e.g., [13] [14]). It is important to note that we do not need to implement the dynamic (5). We will only require a measure of the current backlog and knowledge of the expected service rate as a function of control decisions to implement our control algorithm.

Thus, in every slot t, a control policy needs to make the following decisions:

1) If $t = nT$ (i.e., beginning of a new frame), determine the new set of active servers $\mathcal{S}(t)$. Else, continue using the active set already computed for the current frame.
2) Admission control decisions $R_i(t)$ for all applications i.
3) Routing decisions $R_{ij}(t)$ for the admitted requests.
4) Resource allocation decision $I_j(t)$ at each active server (this includes selecting the CPU frequency that affects the power consumption $P_j(t)$ as well as CPU resource distribution among different VMs).

Our goal is to design an online control policy that maximizes a joint utility of the sum throughput of the applications and the energy costs of the servers subject to the available control options and the structural constraints imposed by this model. It is desirable to develop a flexible and robust resource allocation algorithm that *automatically* adapts to time-varying workloads. In this work, we will use the technique of Lyapunov Optimization [15] to design such an algorithm.

III. Control Objective

Consider any policy η for this model that takes control decisions $\mathcal{S}^\eta(t), R_i^\eta(t), R_{ij}^\eta(t), I_j^\eta(t) \in \mathcal{I}_j, P_j^\eta(t)$ for all i, j in slot t. Note that under any feasible policy η, these control decisions must satisfy the admission control constraint (1),

[1] Additional constraints such as allocating a minimum amount of CPU share to all active VMs can be included in this model.

routing constraints (3), (4), and the resource allocation constraint $I_j(t) \in \mathcal{I}_j$ every slot for all i, j.

Let r_i^η denote the time average expected rate of admitted requests for application i under policy η, i.e.,

$$r_i^\eta = \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\left\{R_i^\eta(\tau)\right\} \tag{6}$$

Let $\boldsymbol{r} = (r_1, \ldots, r_N)$ denote the vector of these time average rates. Similarly, let e_j^η denote the time average expected power consumption of server j under policy η:

$$e_j^\eta \triangleq \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\left\{P_j^\eta(\tau)\right\} \tag{7}$$

The expectations above are with respect to the possibly randomized control actions that policy η might take.

Let α_i and β be a collection of non-negative weights that act as normalizing parameters. Then our objective is to design a policy η that solves the following *stochastic optimization problem*:

$$\begin{aligned} \text{Maximize:} \quad & \sum_{i\in\mathcal{A}} \alpha_i r_i^\eta - \beta \sum_{j\in\mathcal{S}} e_j^\eta \\ \text{Subject to:} \quad & 0 \le r_i^\eta \le \lambda_i \ \forall\, i \in \mathcal{A} \\ & I_j^\eta(t) \in \mathcal{I}_j \ \forall\, j \in \mathcal{S},\ \forall t \\ & \boldsymbol{r} \in \Lambda \end{aligned} \tag{8}$$

Here, Λ represents the *capacity region* of the data center model as described above. It is defined as the set of all possible long-term throughput values that can be achieved under *any* feasible resource allocation strategy.

The objective in problem (8) is a general weighted linear combination of the sum throughput of the applications and the average power usage in the data center. This formulation allows us to consider several scenarios. Specifically, it allows the design of policies that are *adaptive* to time-varying workloads. For example, if the current workload is inside the instantaneous capacity region, then this objective encourages scaling down the instantaneous capacity (by running CPUs at slower speeds and/or turning OFF some active servers) to achieve energy savings. Similarly, if the current workload is outside the instantaneous capacity region, then this objective encourages scaling up the instantaneous capacity (by running CPUs at faster speeds and/or turning ON some inactive servers). Finally, if the workload is so high that it cannot be supported by using all available resources, this objective allows prioritization among different applications. Furthermore, it allows us to assign priorities between throughput and energy by choosing appropriate values of α_i, β.

A. Optimal Stationary, Randomized Policy

Problem (8) is similar to the general stochastic network utility maximization problem presented in [15] in the context of wireless networks with time-varying channels. Suppose (8) is feasible and let r_i^* and e_j^* $\forall i, j$ denote the optimal value of the objective function, potentially achieved by some arbitrary policy. Using the techniques developed in [15], it can be shown that to solve (8) and achieve the optimal value of the objective function, it is sufficient to consider only the class of stationary, randomized policies that take control decisions independent of the current queue backlog every slot. Specifically, at the beginning of each frame, this policy chooses an active set of servers according to a stationary distribution in an i.i.d. fashion. Once chosen, other control decisions are likewise taken in an i.i.d. fashion according to stationary distributions. For the basic model of Sec. II with homogeneous application hosting and identical CPU resources, in choosing an active server set, we do not need to consider all possible subsets of $\mathcal{S}$. Specifically, we define the following collection $\mathcal{O}$ of subsets of $\mathcal{S}$:

$$\mathcal{O} \triangleq \Big\{\emptyset, \{1\}, \{1,2\}, \{1,2,3\}, ..., \{1,2,3,...,M\}\Big\} \tag{9}$$

Then we have the following. For brevity, we state this fact here without proof:

Fact 1: (Optimal Stationary, Randomized Policy) For any arrival rate vector $(\lambda_1, \ldots, \lambda_N)$ (inside or outside of the data center capacity region Λ), there exists a frame-based stationary randomized control policy that chooses active sets from $\mathcal{O}$ every frame, makes admission control, routing and resource allocation decisions every slot independent of the queue backlog and yields the following steady state values:

$$\begin{aligned} \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \Big[\sum_{i\in\mathcal{A}} \alpha_i \mathbb{E}\left\{R_i(\tau)\right\} - \beta \sum_{j\in\mathcal{S}} \mathbb{E}\left\{P_j(\tau)\right\} \Big] \\ = \sum_{i\in\mathcal{A}} \alpha_i r_i^* - \beta \sum_{j\in\mathcal{S}} e_j^* \end{aligned} \tag{10}$$

However, computing the optimal stationary, randomized policy explicitly can be challenging and its implementation impractical as it requires knowledge of all system parameters (like workload statistics) as well as the capacity region in advance. Even if this policy can be computed for a given workload, it would not be adaptive to unpredictable changes in the workload and must be recomputed. In the next section, we will present an *online* control algorithm that overcomes these challenges.

IV. Optimal Control Algorithm

In this section, we use the framework of Lyapunov Optimization to develop an optimal control algorithm for our model. Specifically, we present a dynamic control algorithm that can be shown to achieve the optimal solution r_i^* and e_j^* $\forall i, j$ to the stochastic optimization problem (8). This algorithm is similar in spirit to the backpressure algorithms proposed in [15] for problems of throughput and energy optimal networking in time varying wireless networks.

A. Data Center Control Algorithm (DCA)

Let $V \ge 0$ be a control parameter that is input to the algorithm. This parameter serves as a control probe that allows the system administrator to tradeoff average delay for total average utility as discussed later in Sec. IV-B. Appropriate choice of this parameter depends on the particular system as well as the desired tradeoff between performance and power

cost. This parameter may also be varied over time to affect this tradeoff.

Let $W_i(t), U_{ij}(t) \forall i,j$ be the queue backlog values in slot t. These are initialized to 0. Every slot, the DCA algorithm uses the backlog values in that slot to make joint Admission Control, Routing and Resource Allocation decisions. As the backlog values evolve over time according to the dynamics (2) and (5), the control decisions made by DCA adapt to these changes. However, we note that this is implemented using knowledge of current backlog values only and does not rely on knowledge of statistics governing future arrivals. Thus, DCA solves for the objective in (8) by implementing a sequence of optimization problems over time. The queue backlogs themselves can be viewed as dynamic Lagrange multipliers that enable stochastic optimization [15].

The DCA algorithm operates as follows.

Admission Control: For each application i, choose the number of new requests to admit $R_i(t)$ as the solution to the following problem:

$$\begin{aligned} \text{Maximize:} \quad & R_i(t)[V\alpha_i - W_i(t)] \\ \text{Subject to:} \quad & 0 \leq R_i(t) \leq A_i(t) \end{aligned} \tag{11}$$

This problem has a simple threshold-based solution. In particular, if the current Router buffer backlog for application i, $W_i(t) > V\alpha_i$, then $R_i(t) = 0$ and no new requests are admitted. Else, if $W_i(t) \leq V\alpha_i$, then $R_i(t) = A_i(t)$ and all new requests are admitted. Note that this admission control decision can be performed separately for each application.

Routing and Resource Allocation: Let $S(t)$ be the active server set for the current frame. If $t \neq nT$, then we continue to use the same active set. The Routing and Resource Allocation decisions are given as follows:

Routing: Given an active server set, routing follows a simple *Join the Shortest Queue* policy. Specifically, for any application i, let $j' \in S(t)$ be the active server with the smallest queue backlog $U_{ij'}(t)$. If $W_i(t) > U_{ij'}(t)$, then $R_{ij'}(t) = W_i(t)$, i.e., all requests in the Router buffer for application i are routed to server j'. Else, $R_{ij}(t) = 0 \forall j$ and no requests are routed to any server for application i. In order to make these decisions, the Router requires the queue backlog information $U_{ij}(t) \forall i,j$. Given this information, we note that this routing decision can be performed separately for each application.

Resource Allocation: At each active server $j \in \mathcal{S}(t)$, choose a resource allocation $I_j(t)$ that solves the following problem:

$$\begin{aligned} &\text{Maximize: } \sum_i U_{ij}(t)\mathbb{E}\left\{\mu_{ij}(I_j(t))\right\} - V\beta P_j(t) \\ &\text{Subject to: } I_j(t) \in \mathcal{I}_j, P_j(t) \geq P_{min} \end{aligned} \tag{12}$$

The above problem is a generalized max-weight problem where the service rate provided to any application is weighted by its current queue backlog. Thus, the optimal solution would allocate resources so as to maximize the service rate of the most backlogged application.

The complexity of this problem depends on the size of the control options $\mathcal{I}_j$ available at server j. In practice, the number of control options such as available DVFS states, CPU shares, etc. is small and thus, the above optimization can be implemented in real time. It is important to note that each server solves its own resource allocation problem independently using the queue backlog values of applications hosted on it and this can be implemented in a *fully distributed* fashion.

If $t = nT$, then a new active set $\mathcal{S}^*(t)$ for the current frame is determined by solving the following:

$$\begin{aligned} \mathcal{S}^*(t) = \underset{\mathcal{S}(t)\in\mathcal{O}}{\text{argmax}} \Big[&\sum_{ij} U_{ij}(t)\mathbb{E}\left\{\mu_{ij}(I_j(t))\right\} - V\beta\sum_j P_j(t) \\ &+ \sum_{ij} R_{ij}(t)(W_i(t) - U_{ij}(t)) \Big] \\ \text{subject to: } & j \in \mathcal{S}(t), I_j(t) \in \mathcal{I}_j, P_j(t) \geq P_{min} \\ & \text{constraints } (1), (3), (4) \end{aligned} \tag{13}$$

The above optimization can be understood as follows. To determine the optimal active set $\mathcal{S}^*(t)$, the algorithm computes the optimal cost for the expression within the brackets for every possible active server set in the collection $\mathcal{O}$. Given an active set, the above maximization is separable into Routing decisions for each application and Resource Allocation decisions at each active server. This computation is easily performed using the procedure described earlier for Routing and Resource Allocation when $t \neq nT$.

Since $\mathcal{O}$ has size M, the worst-case complexity of this step is polynomial in M. However, the computation can be significantly simplified as follows. It can be shown that if the maximum queue backlog over all applications on any server j exceeds a finite constant U_{thresh}, then that server must be part of the active set. Thus, only those subsets of $\mathcal{O}$ that contain these servers need to be considered when searching for the optimal active set.

We note that it is possible for this algorithm to inactivate certain servers even if they have non-zero queue backlog (and process it later when the server is activated again). This can happen, for example, if the backlog on the server is small and the optimization (13) determines that the energy cost of keeping the server ON (the second term) exceeds the weighted service rate achieved (the first term). While we can show optimality of this algorithm in terms of solving the objective (8), we also consider a more practical (and potentially suboptimal) strategy DCA-M that migrates or reroutes such unfinished requests from inactive servers to other active servers in the next frame. Our simulation results in Sec. VI suggest that the performance of this strategy is very close to that under DCA.

Finally, the computation in (13) requires knowledge of the values of queue backlogs at all servers as well as the router buffers. This can be implemented by a centralized controller (that also implements Routing) that periodically gathers the backlog information and determines the active set for each frame. See further discussion in Sec. IV-C.

B. Performance Analysis

Theorem 1: (Algorithm Performance) Assume that all queues are initialized to 0. Suppose all arrivals in a slot $A_i(t)$ i.i.d. and are upper bounded by finite constants so that

$A_i(t) \leq A_i^{max}$ for all i, t. Also let μ_{max} be the maximum service rate (in requests/slot) over all applications in any slot. Then, implementing the DCA algorithm every slot for any fixed control parameter $V \geq 0$ and frame size T yields the following performance bounds:

1) The worst case queue backlog for each application Router buffer $W_i(t)$ is upper bounded by a finite constant W_i^{max} for all t:

$$W_i(t) \leq W_i^{max} \triangleq V\alpha_i + A_i^{max} \tag{14}$$

Similarly, the worst case queue backlog for application i on any server j is upper bounded by $2W_i^{max}$ for all i, t:

$$U_{ij}(t) \leq 2W_i^{max} = 2(V\alpha_i + A_i^{max}) \tag{15}$$

2) The time average utility achieved by the DCA algorithm is within BT/V of the optimal value:

$$\liminf_{t\to\infty} \frac{1}{t}\sum_{\tau=0}^{t-1}\Big[\sum_{i\in\mathcal{A}}\alpha_i\mathbb{E}\left\{R_i(\tau)\right\} - \beta\sum_{j\in\mathcal{S}}\mathbb{E}\left\{P_j(\tau)\right\}\Big] \geq \sum_{i\in\mathcal{A}}\alpha_i r_i^* - \beta\sum_{j\in\mathcal{S}} e_j^* - \frac{BT}{V} \tag{16}$$

where B is a finite constant (defined precisely in the Appendix) that depends on the second moments of the arrival and service processes. We note that the performance bounds above are quite strong. In particular, part (1) establishes *deterministic* worst case bounds on the maximum backlogs in the system at all times. Therefore, by part (2) of the theorem, the achieved average utility is within $O(1/V)$ of the optimal value. This can be pushed arbitrarily close to the optimal value by increasing the control parameter V. However, this increases the maximum queue backlog bound $(14), (15)$ linearly in V. Thus, by Little's Theorem, this directly leads to an $O(1/V, V)$ utility-delay tradeoff.

We next prove the first part of Theorem 1. Proof of part (2) uses the technique of Lyapunov Optimization [15] and is provided in the Appendix.

Proof of part (1): Suppose that $W_i(t) \leq W_i^{max}$ for all i for some time t. This is true for $t = 0$ as all queues are initialized to 0. We show that the same holds for time $t + 1$. We have 2 cases. If $W_i(t) \leq W_i^{max} - A_i^{max}$, then from (2), we have $W_i(t+1) \leq W_i^{max}$ (because $R_i(t) \leq A_i^{max}$ for all t). Else, if $W_i(t) > W_i^{max} - A_i^{max}$, then $W_i(t) > V\alpha_i + A_i^{max} - A_i^{max} = V\alpha_i$. Then, the flow control part of the algorithm chooses $R_i(t) = 0$, so that by (2):

$$W_i(t+1) \leq W_i(t) \leq W_i^{max}$$

This proves (14). To prove (15), note that new requests are routed from a Routing buffer $W_i(t)$ to an application queue $U_{ij}(t)$ only when $W_i(t) > U_{ij}(t)$. Since $W_i(t) \leq W_i^{max}$ and since the maximum number of arrivals in a slot to $U_{ij}(t)$ is W_i^{max}, $U_{ij}(t)$ cannot exceed $2W_i^{max}$. □

C. Instrumentation Requirements

The proposed framework does not require explicit modeling of the arrival and service processes. However, we require a measure of job backlogs at each VM and we require a control signaling to exchange the backlog information from physical servers to controller nodes. The measure of backlog can be direct or indirect depending on the application environment. In one extreme, as a direct approach, the datacenter provides an application platform such as MapReduce or Hadoop over which all the applications are written. In such a set up, there are explicit job monitoring, assignment, and collection facilities in place that can support backlog measurements. Another direct approach is to require applications (VMs) to log job states and make them accessible to the control framework. Various indirect approaches mainly depend on monitoring the explicit signals that indicate the job arrivals and departures such as client request and server response messages (e.g., web services), session initiation and termination (e.g., video session), thread creation and termination, etc. The accuracy and feasibility of indirect approaches highly depend on the particular application and hence different instrumentation methods must be used to obtain a measure of backlog. The optimality properties of our algorithm hold even when the backlog values used are different from the actual values (as long as they are off by a bounded value that does not depend on V) making it robust to such noisy backlog estimates.

V. Extensions

Here, we briefly discuss two extensions to the basic model.

A. Inhomogeneous Placement and CPU Resources

In this case, the a_{ij} variables need not be equal to $1 \forall i, j$ so that requests for an application i can only be routed to one of those servers that hosts this application. The routing constraints in (3), (4) are already general enough to capture this. In the case of inhomogeneous CPU resources, the DCA algorithm needs the following modification. In the active server determination step, instead of only searching over the collection $\mathcal{O}$ of subsets in (9), now it may have to search over all possible subsets of $\mathcal{S}$. This can be computationally intensive when $\mathcal{S}$ is large. It is possible to tradeoff complexity for utility optimality by resorting to sub-optimum heuristic approaches, investigation of which is left out for brevity in this paper.

B. Multi-tier Applications

Modern enterprise applications typically have multiple components working in a tiered structure [12] [14]. An incoming request for such a multi-tier application is serviced by each tier in a certain order before departing the system. Our framework can be extended to treat this scenario by modeling the multi-tier application as a *network of queues*. Specifically, we define $U_{ij}^k(t)$ as the queue backlog for the k^{th} tier of application i on server j (where $k \geq 1$). Then, the queueing dynamics for $U_{ij}^k(t)$ are given by:

$$U_{ij}^k(t+1) = \max[U_{ij}^k(t) - \mu_{ij}^k(I_j(t)), 0] + \sum_{l\in\mathcal{S}(t)} R_{il}^{k-1}(t)$$

where $R_{il}^{k-1}(t)$ denotes the arrivals to $U_{ij}^k(t)$ from the $(k-1)^{th}$ tier of application i on server l. For $k = 1$, this corresponds to (5) where accepted requests are routed to the first tier of

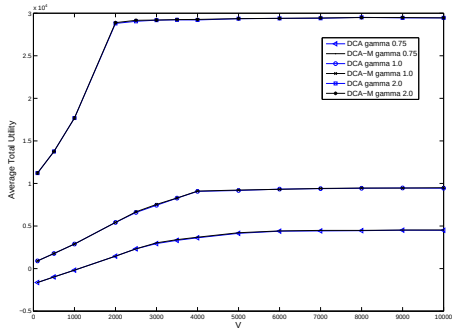

Fig. 3. Average total utility vs V for different values of γ.

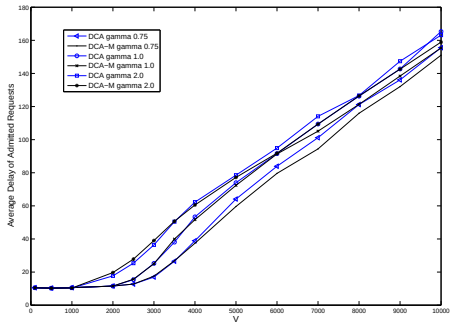

Fig. 4. Average delay of admitted requests vs V for different values of γ.

application i on server j after admission control decisions. Using the Lyapunov Optimization framework presented in the previous sections together with the technique of backpressure routing [15], DCA can be extended to treat such multi-tier scenarios.

VI. Simulations

We simulate the DCA and DCA-M algorithms in an example virtualized data center consisting of 100 servers and hosting 10 applications. Each application i is CPU intensive and receives requests exogenously according to a random arrival process of rate λ_i. In the simulation setup, each CPU is assumed to follow a quadratic power-frequency relationship similar to the experimentally obtained quadratic power-frequency curve in Fig. 2. Specifically, each CPU is assumed to have a discrete set of frequency options in the interval $[1.6GHz, ..., 2.6GHz]$ at increments of 0.2 GHz and the corresponding power consumption (in Watts) at frequency f is given by $P_{min} + \theta(f - 1.6GHz)^2$ where $P_{min} = 120W$ and $\theta = 120W/(GHz)^2$. Thus, the CPU power consumption at the highest frequency is $240W$. We assume that half of the servers in the data center are always ON and that decisions to dynamically turn servers ON/OFF are applied to the remaining servers. Note that the dynamic operating frequency decisions are still applied to all servers. The frame size $T = 1000$ slots and the simulations were run for one million slots.

The number of new requests generated for an application i in a slot is assumed to be uniformly and randomly distributed in $[0, 2\lambda_i]$. On average, a server running at the minimum (maximum) speed can process 200 (400) requests/slot. In the simulations, the throughput utility weights are chosen to be equal for all applications, so that $\alpha_i = \alpha \forall i$.

In the first experiment, we fix the input rate $\lambda_i = 2000$ requests/slot for all applications and simulate the DCA and DCA-M algorithms for different choices of the ratio $\gamma = \alpha/\beta$. Figs. 3 shows the total average utility for different values of the input parameter V under the two control algorithms. We observe that the performance of DCA-M is very close to DCA. Further, the total average utility achieved increases with V and converges to a maximum value for larger values of V as predicted by (16). Fig. 4 plots the average delay of the admitted requests vs V. It can be seen that the average delay increases linearly with V as predicted by the bounds in (14), (15). Fig. 5 shows the fraction of declined requests vs V under both algorithms. This, along with Figs. 3 and 4 shows the $O(1/V, V)$ utility-delay tradeoff offered by the DCA algorithm where the average utility achieved can be pushed closer to the optimal value with a tradeoff in terms of a linear increase in average delay.

In the second experiment, we fix the parameters $V = 5000, \gamma = 1.0$ and consider the scenario where the input rate changes in an unpredictable manner. Specifically, for the first $1/3$ of the simulation interval, the input rate $\lambda_i = 1000$ requests/slot for all applications. Then the input rate abruptly increases to 3000 requests/slot before dropping to 2000 requests/slot in the last $1/3$ of the simulation interval. In Fig. 6, we plot the number of active servers vs. frame number under the DCA algorithm. It can be seen that the algorithm quickly adapts to the new workload by increasing or decreasing the number of active servers (and hence the instantaneous capacity) even when the workload changes in an unpredictable manner.

VII. Conclusions and Future Work

In this paper, we considered the problem of dynamic resource allocation and power management in virtualized data centers. Prior work in this area uses prediction based approaches for resource provisioning. In this work, we have used an alternate approach that makes use of the queueing information available in the system to make online control decisions. This approach is adaptive to unpredictable changes in workload and does not require estimation and prediction of its statistics. Our approach uses the recently developed technique of Lyapunov Optimization that allows us to derive analytical performance guarantees of the algorithm.

The main focus of this work was on building an analytical framework. As part of future work, we plan to have real system implementation of our algorithm and use standard benchmark workloads and applications to evaluate its performance.

Appendix - Proof of Theorem 1 part (2)

Here, we prove part (2) of Theorem 1 using the technique of Lyapunov Optimization [15]. This technique involves constructing an appropriate Lyapunov function of the queue backlogs in the system, defining the conditional "Lyapunov

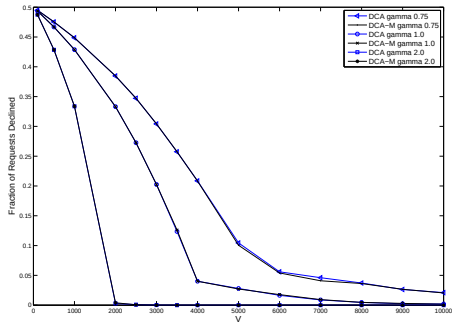

Fig. 5. Fraction of declined requests vs V for different values of γ.

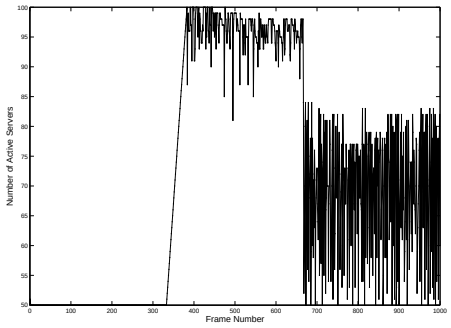

Fig. 6. Number of active servers over time.

drift" of this function, and then developing a dynamic algorithm that minimizes this drift over all control policies. The performance bounds for this algorithm are obtained by comparing the Lyapunov drift under this algorithm with that of the backlog independent optimal stationary, randomized policy described in Sec. III-A.

Let $\boldsymbol{Q}(t) = (U_{11}(t), \ldots, U_{NM}(t), W_1(t), \ldots, W_M(t))$ represent the collection of all queue backlogs in the system. We define a Lyapunov function: $L(\boldsymbol{Q}(t)) \triangleq \frac{1}{2}\left[\sum_{i\in\mathcal{A}, j\in\mathcal{S}} U_{ij}^2(t) + \sum_{i\in\mathcal{A}} W_i^2(t)\right]$. Define the conditional Lyapunov drift $\Delta(\boldsymbol{Q}(t))$ as follows: $\Delta(\boldsymbol{Q}(t)) \triangleq \mathbb{E}\left\{L(\boldsymbol{Q}(t+1)) - L(\boldsymbol{Q}(t))|\boldsymbol{Q}(t)\right\}$. Using queueing dynamics (2) and (5), the conditional Lyapunov drift $\Delta(t)$ under any control policy (including DCA) can be computed as follows:

$$\Delta(t) \leq B - \sum_{ij} U_{ij}(t)\mathbb{E}\left\{\mu_{ij}(I_j(t)) - R_{ij}(t)|\boldsymbol{Q}(t)\right\} - \sum_i W_i(t)\mathbb{E}\left\{\sum_j R_{ij}(t) - R_i(t)|\boldsymbol{Q}(t)\right\} \quad (17)$$

where $B \triangleq \frac{\sum_i (A_i^{max})^2 + NM\mu_{max}^2}{2}$.

For a given control parameter $V \geq 0$, we subtract the reward metric $V\mathbb{E}\left\{\sum_i \alpha_i R_i(t) - \beta\sum_j P_j(t)|\boldsymbol{Q}(t)\right\}$ from both sides of (17) and rearrange the terms to get the following:

$$\begin{aligned}\Delta(t) - V\mathbb{E}\left\{\sum_i \alpha_i R_i(t) - \beta\sum_j P_j(t)|\boldsymbol{Q}(t)\right\} &\leq B \\ - \sum_{ij} U_{ij}(t)\mathbb{E}\left\{\mu_{ij}(I_j(t))|\boldsymbol{Q}(t)\right\} + V\beta\sum_j \mathbb{E}\left\{P_j(t)|\boldsymbol{Q}(t)\right\} \\ - \sum_{ij}\mathbb{E}\left\{R_{ij}(t)(W_i(t) - U_{ij}(t))|\boldsymbol{Q}(t)\right\} \\ - \sum_i \mathbb{E}\left\{R_i(t)(V\alpha_i - W_i(t)|\boldsymbol{Q}(t)\right\} \quad (18)\end{aligned}$$

From the above, it can be seen that the dynamic control algorithm DCA described in Sec. IV-A is designed to take Admission Control, Routing and Resource Allocation decisions that *minimize* the right hand side of (18) over all possible options, including the stationary policy of Sec. III-A. Theorem 1 part (2) can now be shown using a direct application of the Lyapunov Optimization Theorem (see Theorem 5.4 in [15]) along with a T-slot delayed Lyapunov analysis.

REFERENCES

[1] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, Jan. 2009.

[2] X. Fan, W. Weber, and L. Borroso. Power provisioning for a warehouse-sized computer. In *Proceedings of ISCA*, June 2007.

[3] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No "power" struggles: coordinated multi-level power management for the data center. In *Proceedings of ASPLOS*, March 2008.

[4] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. Optimal power allocation in server farms. In *Proceedings of SIGMETRICS*, June 2009.

[5] A. Weirman, L. L. H. Andrew, and A. Tang. Power-aware speed scaling in processor sharing systems. In *Proceedings of INFOCOM*, April 2009.

[6] P. Padala, K-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automatic control of multiple virtualized resources. In *Proceedings of EuroSys*, April 2009.

[7] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *Proceedings of EuroSys*, March 2007.

[8] X. Liu, X. Zhu, P. Padala, Z. Wang, and S. Singhal. Optimal multivariate control for differentiated services on a shared hosting platform. In *Proceedings of CDC*, Dec. 2007.

[9] D. Kusic and N. Kandasamy. Power and performance management of virtualized computing environments via lookahead control. In *Proceedings of ICAC*, June 2009.

[10] Y. Chen, A. Das, W Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing server energy and operational costs in hosting centers. In *Proceedings of SIGMETRICS*, June 2005.

[11] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini. Statistical profiling-based techniques for effective power provisioning in data centers. In *Proceedings of EuroSys*, April 2009.

[12] X. Wang, D. Lan, X. Fang, M. Ye, and Y. Chen. A resource management framework for multi-tier service delivery in autonomic virtualized environments. In *Proceedings of NOMS*, April 2008.

[13] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In *Proceedings of OSDI*, Dec. 2002.

[14] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An analytical model for multi-tier internet services and its applications. In *Proceedings of SIGMETRICS*, June 2005.

[15] L. Georgiadis, M. J. Neely, L. Tassiulas. Resource allocation and cross-layer control in wireless networks. *Foundations and Trends in Networking*, vol. 1, no. 1, pp. 1-149, 2006.