NFRA: Generalized Network Flow Based Resource Allocation for Hosting Centers

Kimish Patel, Murali Annavaram, *Senior Member, IEEE*, Massoud Pedram, *Fellow, IEEE*

Abstract — Due to prohibitive cost of datacenter setup and maintenance, many small-scale businesses rely on hosting centers to provide the cloud infrastructure to run their workloads. Hosting centers host services of the clients on their behalf and guarantee quality of service as defined by service level agreements (SLAs.) To reduce energy consumption and to maximize profit it is critical to optimally allocate resources to meet client SLAs. Optimal allocation is a non-trivial task due to two factors: hosting centers have wide resource heterogeneity where energy consumption of a client task varies depending on the allocated resources. Second, due to lack of energy proportionality energy cost for a task varies based on server utilization. In this paper we introduce a generalized Network Flow based Resource Allocation framework, called NFRA, for energy minimization and profit maximization. NFRA provides a unified framework to model profit maximization under a wide range of SLAs. We will demonstrate the simplicity of this unified framework by deriving optimal resource allocations for three different SLAs. We derive workload demands and server energy consumption data from SPECWeb2009 benchmark results to quantify profit gains obtained by NFRA over a greedy approach and to compare NFRA with a pseudo optimal approach.

Index Terms — Resource allocation, network flow, data center, hosting center, clouds, energy proportionality.

1. INTRODUCTION

Datacenters are the backbone of the growth in e-commerce and digital services and continue to grow in size as the demand for information technology increases. But datacenters themselves are now faced with a major impediment of power consumption. A significant fraction of the datacenter power consumption is due to resource over-provisioning. A recent EPA report predicts that if datacenter resources are managed with state-of-the-art solutions, the power consumption in 2011 can be reduced from 10 Gigawatts to below 5 Gigawatts [25]. These solutions require perfectly provisioned servers. Perfect provisioning requires allocating only the absolute minimum resources for completing the tasks while meeting the specified SLAs. Perfect provisioning is difficult to achieve due to two reasons: resource heterogeneity and lack of energy proportionality. We provide a brief overview of these two impediments.

Resource Heterogeneity: Datacenter resources become heterogeneous, even if a datacenter is initially provisioned with homogeneous resources. For instance, replacing non-operational servers or adding a new rack of servers to accommodate demand typically leads to installing new servers that reflect the advances in current state-of-the-art. In this research we focus on performance heterogeneity where servers differ in CPU speed, memory and disk capacity, which also leads to heterogeneity in power consumption. Heterogeneity makes perfect provisioning at the datacenter level difficult since different tasks may occupy different amount of resources across different servers with varying energy profiles.

Energy Proportionality: Energy proportionality is the notion that the energy consumed by a resource must scale linearly with its utilization. Hence, a perfectly energy proportional server consumes zero power at zero utilization and its power consumption increases linearly with utilization. However, servers typically consume 80% of the peak power even at 20% utilization [21]. The consequence of this lack of proportionality is that when a task is assigned to a server the energy cost of completing that task is dependent on the resulting server utilization. Hence, it is insufficient to consider energy cost of operating a datacenter based solely on the number of active servers; in fact, it is critical to consider the energy cost as a function of server utilization.

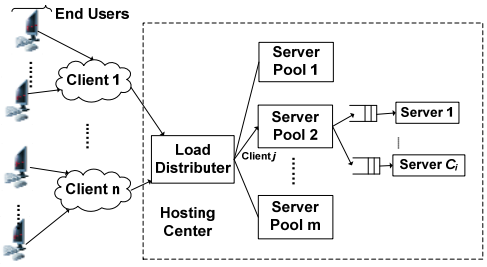

Fig. 1 Hosting Center Architecture.

Hosting Centers: We target our research toward hosting centers in particular, which are an important incarnation of datacenters. Hosting centers provide compute services to clients, such as small business owners, that need computing capabilities of a

Murali Annavaram, and Massoud Pedram are with Electrical Engineering Department, University of Southern California, Los Angeles, CA, 90089, USA, E-mail: {annavara, pedram}@usc.edu
Kimish Patel is at Nvidia Corportation, Santa Clara, CA, 95050, USA, E-mail: kimish.patel@gmail.com

datacenter but do not have the wherewithal to operate one. Hosting centers may be organized in multiple ways. The particular organization used in this research is shown in Fig. 1 where requests from multiple hosting center clients are first directed to a load distributer. The load distributer is connected to pools of servers, where all servers in a pool are homogeneous, but pools are heterogeneous which is hidden from the clients by the load distributer. Clients' requests are in turn generated by each client's own end-user requests. For instance, end users may generate browsing/ purchasing transactions to an e-commerce client which is in-turn routed to the hosting center. Hosting center operator and clients are bound by SLAs where hosting center guarantees a minimum level of service. Each client can define its own QoS requirement, such as bounds on response time of a request, or request throughput. When the hosting center operator fails to meet SLAs they may even have to pay a penalty to the client. It is the job of the load distributor to allocate resources to satisfy client's tasks while making sure SLAs are met. Conservative resource allocation to meet SLAs can lead to over provisioning and thus increase energy consumption thereby reduce profit. On the other hand under provisioning of resources may lead to profit loss when operator pays penalty for missing SLAs or loses customers.

The goal of this research is to provide a unified framework to maximize profit under a wide range of SLAs by allocating resources optimally in the presence of server heterogeneity and lack of energy proportionality. We will formulate this optimization problem as a generalized network flow based resource allocation framework, where nodes in the network represent server pools and clients and the flow from server pools to clients represents resource allocation.

1.1 *Research Contributions*

Previous works ([1][2][3][11][12]) have addressed the issue of energy minimization and profit maximization which are covered in more detail in related work section. We will highlight the research contribution of this paper particularly in comparison to two closely related previous works, namely [11] and [12]:

1) NFRA accounts for server heterogeneity. In particular, it considers how performance and energy costs scale non-uniformly for different clients across different servers. Consider two heterogeneous servers, A and B, and two clients, X and Y. For client X, server A is able to process 100 requests/second, while server B is able to process 150 requests/second. Whereas for client Y, A is able to process 100 requests/second, while B is able to process 200 requests/second. Thus the scaling of performance across these two servers in not the same for the two clients. Such heterogeneity also exists in energy consumption. In contrast, experimental results presented in [11] considered the same service rate across different resource types for a given class of requests.
2) NFRA accounts for non linear energy cost of operating a server at different utilization levels. In [11] the energy cost of operating a server was not considered as they maximize profit by only considering generated revenues. In [12] the authors account for the energy cost of operating a server using a simple cost model; if a server is ON it consumes fixed energy whereas if it is OFF it consumes zero energy. Due to lack of energy proportionality, the energy cost of servicing a request can change dramatically at different utilization levels, which is not considered in [12].
3) NFRA accounts for response time constraints while minimizing energy costs. Reference [12] considers energy optimization without taking into account response time constraints, e.g., average or maximum response time not to exceed the stipulated response time requirement.

2. Model Parameters and Assumptions

Let us assume that the hosting center has m heterogeneous server pools, $i=1, 2,\ldots, m$. All servers within a given server pool are homogenous. (Note that we will use servers and resources synonymously in the remainder of this paper.) Each pool is characterized by C_i, representing the number of homogenous servers in pool i. As shown in Fig. 1, each server within a pool is modeled as a single server queuing system for each of the clients it is serving. Thus requests of a given client are distributed across servers where they wait in a single server queue.

Assuming that the hosting center is hosting n clients, $j=1, 2,\ldots, n$, each client is characterized by the following parameters.

- λ_j: average request arrival rate, in requests per second, for client j, accounting for any user think times.
- μ_{ij}: average service rate in requests per second for client j's requests on a server from server pool i at system utilization U=1. Hence, execution time of client j's request on a server from server pool i, $T_{ij}=1/\mu_{ij}$. We don't make any assumptions about the distribution of random variables λ_j and μ_{ij}.
- e_{ij}: average energy cost in joules per request to serve a request of client j on server pool i at server utilization U=1.
- $\tau_{j,max}$: average or maximum tolerable response time in seconds for client j.
- β_j: stochastic upper bound on the fraction of client j's requests that can violate $\tau_{j,max}$, i.e.:

$$\Pr(\tau_j > \tau_{\max}) \le \beta_j$$

  where τ_j is the response time of a request of client j.
- R_j: per request price paid to hosting center by client j as long as the response time is not larger than $\tau_{j,max}$.
- L_j: per request penalty incurred on hosting center by client j whenever the response time exceeds $\tau_{j,max}$.

Parameters, $\tau_{j,max}$, β_j, R_j, L_j, are specified in the SLA. Usually, the client also provides λ_j; if, however, λ_j is unavailable at the time granularity of making resource allocation decisions, then one can use a history based predictor such as the one in [22] to estimate λ_j. Computing μ_{ij} and e_{ij} is the responsibility of the hosting center which can be obtained by profiling a given client workload on each server pool. We demonstrate one such profiling methodology here, used to derive the aforementioned parameters in our experiments.

We assume that when the hosting center signs on a new client, it conservatively allocates servers from each server pool it has to this client and monitors the resulting performance in terms of number of requests served, server utilization, and power consumption. Let us assume that during such a phase hosting center allocates L_{ij} servers from server pool i to client j. We observe at these L_{ij} servers, over a period of $\delta_{profile}$ seconds, that the total number of requests served for a client j on some server l of pool i is $\mu_{ij,l}$ and the server utilization is $U_{ij,l}$. Based on these values, we obtain μ_{ij} by first scaling the service rate ($\mu_{ij,l}/\delta_{profile}$) using $U_{ij,l}$ to obtain service rate at U=1 at server l and then taking average across L_{ij} servers, as shown in eq. (1).

$$\mu_{ij} = \frac{1}{L_{ij}} \sum_{l} \left(\frac{\mu_{ij,l}}{\delta_{profile}} \right) \frac{1}{U_{ij,l}}; T_{ij} = \frac{1}{\mu_{ij}} \quad (1)$$

Furthermore we also measure the average power (energy/second) consumed by server l in server pool i at utilization $U_{ij,l}$ during the period of $\delta_{profile}$ seconds. Let us denote this power by $P_{ij,l}$. Given $P_{ij,l}$ and $U_{ij,l}$ we obtain $P_{ij,(1)}$, server power at U=1 for client j on pool i, as shown in eq. (2), by first scaling $P_{ij,l}$ linearly to obtain power at U=1 on server l and then taking average across L_{ij} servers.

$$e_{ij} = \left(\frac{\mu_{ij}}{P_{ij,(1)}} \right); P_{ij,(1)} = \frac{1}{L_{ij}} \sum_{l} \left(P_{i,idle} + \left(P_{ij,l} - P_{i,idle} \right) / U_{ij,l} \right) \quad (2)$$

We assume that $P_{i,idle}$, idle state server power for server pool i, is given. Using $P_{ij,(1)}$ we obtain e_{ij} as well, as shown in eq. (2). Note also that the heterogeneity in server pools, in terms of processor speed, memory, IO etc., will be captured by and reflected in μ_{ij}, T_{ij} and e_{ij}. The T_{ij} and e_{ij} characterization results presented in experimental section will help clarify this further (cf. Fig. 7). Note that we have assumed that any client can be mapped to any server pool. This is not a limiting assumption for the proposed framework and is made solely to simplify the presentation.

3. NFRA Framework

In this section we describe a generalized Network Flow based Resource Allocation framework for energy minimization and profit maximization in hosting centers, which we call NFRA.

3.1 Resource Allocation Flow in NFRA

Fig. 2 shows overall flow of the proposed resource allocation framework. Whenever a new client arrives the hosting center performs client's resource and energy requirement profiling and stores the relevant data. This profiled data along with profiled data of the other clients hosting center is serving is fed to NFRA network flow formulation. This is the crux of the proposed approach explained in the following section. Solution to the network flow yields the resource allocation result which goes to performance monitoring which constantly evaluates the optimality of the obtained solution. We present the relevant details in sensitivity analysis section of experimental results. Based on the variations observed in the efficiency of the current solution, the decision to solve another instance of the resource allocation problem is taken.

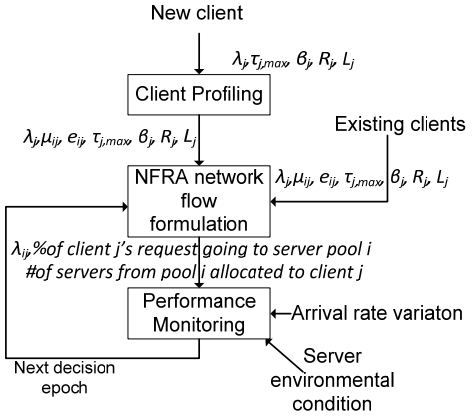

Fig. 2 Resource allocation flow.

3.2 Brief Introduction to Generalized Networks

Generalized networks [18] are similar to regular networks. In regular networks each edge from source, v, to destination, w, is associated with capacity, u. Each edge has cost, κ, which defines the cost of sending one unit of flow from v to w. In generalized networks the edge (v, w) of the network has a new parameter called gain factor, γ ($\gamma \geq 0$). If an edge (v, w) has a gain factor of γ, then one unit of flow that leaves node v becomes γ units when it arrives at w. Generalized networks are useful to model financial systems with interest rates, oil pipeline networks with leaks, currency exchange rate problems, and so on. In such problems the objective is to find the maximum flow, much like maximum flow in regular networks, into some sink node t such that the return on the investment is maximized or the maximum amount of oil is received at the sink t, etc. Based on these parameters we can characterize each edge (v, w) of the generalized network by a triplet (γ, κ, u).

The unique property of generalized network flow models is their gain factors. We use gain factor to capture the heterogeneity of server pools in a hosting center. For example, the execution times, or service rates, for the requests generated by a client may vary widely across different server pools due to heterogeneity. Such differences can be encapsulated as gain factors as we will show in the next section. Note that gain factors can also capture other parameters

affecting hosting center like rising server temperature which may result in either slowing down servers or shutting them off. If servers are slowed down using voltage/frequency scaling or throttled then their service rate gets affected which can be captured using gain factors. If the servers are shut down then they can be removed from the server pool which will reflect in server pool's capacity. Thus generalized network provides a framework to captures various such parameters which can affect the runtime behavior of a hosting center.

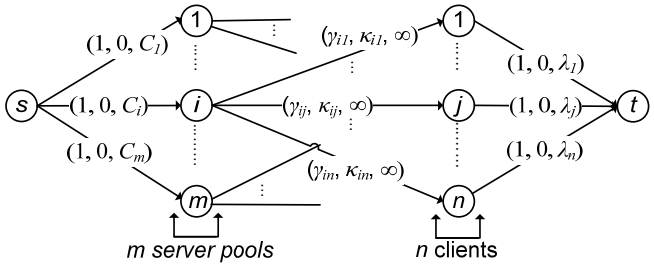

Fig. 3 Generalized Network Model for NFRA.

3.3 Modeling Resource Allocation in NFRA

In order to construct a generalized network for hosting center resource allocation problem, let us assume that each server pool i and each client j represents a node in a bi-partite graph. As shown in Fig. 3, the left side of the bi-partite graph represents a hosting center comprising of different server pools whereas the right part of the graph represents different clients. Each server pool i is connected to a client j by a directed edge (i, j). Each such edge (i, j) is characterized by a gain factor, γ_{ij}, which captures the amount of service one unit of server pool i (i.e., one server in that pool) provides to client j. In other words, one unit of flow that is sent from server pool i becomes γ_{ij} serviced requests when it arrives at client j. Note that γ_{ij} is the same as μ_{ij} when server utilization is 1. For any other utilization ($U < 1$), $\gamma_{ij} < \mu_{ij}$.

Furthermore, each edge is characterized by the cost κ_{ij} that captures the cost of sending one unit of flow from node i to node j, which captures the cost of allocating one unit of server pool i to client j. As we will explain in Section 3.5, the cost parameter is a function of the SLA type; in some cases the cost is purely the energy cost of allocating one server to satisfy a client's requests and in other cases the cost could account for complex profit functions. Finally, each edge (i, j) in this bipartite graph is assumed to have capacity $u_{ij}=\infty$.

To this bi-partite graph, we add a source node s and a sink node t. Each server pool node i is connected to s by an edge (s, i). The gain factor and cost associated with this edge are set to: $\gamma_{si}=1$ and $\kappa_{si}=0$. The capacity of such an edge (s, i) is set to be $u_{si}=C_i$. Intuitively this implies that from node s, we will push flow in terms of servers, with a maximum of C_i servers, towards a server pool node i. The sink node t is connected to each client node j by an edge (j, t). The gain factor and cost associated with this edge (j, t) are also set to $\gamma_{jt}=1$ and $\kappa_{jt}=0$, while the capacity is set to be $u_{jt}=\lambda_j$. Intuitively this means that from client nodes, we will push flow in terms of number of serviced requests, with a maximum of λ_j serviced requests, onto the edges towards t. Fig. 3 shows the resulting generalized network graph where each edge (v, w) is characterized by the triplet (γ_{vw}, κ_{vw}, u_{vw}). Such a network is denoted by G from here on. The value of some flow F in such a network is defined by the value of the net amount of flow going into the sink t, i.e., the total number of client requests that can be serviced by the server pools.

Having defined flow F in G as above, the maximum amount of flow in G, F_{max}, satisfies the following inequality:

$$F_{\max} \le \sum\nolimits_j \lambda_j$$

This equation states that the value of F_{max} can never be more than the sum of request arrival rates. It is however possible that F_{max} is less than the sum of the arrival rates in which case no feasible solution exists in G that satisfies all the requests of every client. In this case hosting center may have to some kind of employ admission control. One such admission control policy based on the current framework is presented in the next section.

Note that the solutions achieved through generalized network are fractional. However under the assumption, which is not unrealistic, that the number of requests generated by clients is much larger than the number of servers hosted, the fractional result does not have any dire implications. In any case, the fractional ratio of client request distribution across different server pools can be achieved over a sufficiently large time interval.

In order to realize various SLAs, we will modulate the gain factors γ_{ij} and associated costs κ_{ij} of the edges in G. The following subsections cover different SLA types considered in this paper and their corresponding realizations in G.

3.4 Admission Control in NFRA

Whenever, due to insufficient resources, no feasible solution exist we must employ admission control to find a feasible solution that satisfies at least some requests from all the clients. Admission control is relatively out of scope for the presented work; however, in order to demonstrate the strength of NFRA framework we present an outline of the admission control policy.

If F_{max} is smaller than the sum of arrival rates then there is no solution that satisfies all the requests from all the clients. However we did find a min-cost max flow in the form of F_{max} implying that there is some flow going through edges (j, t). If the flow through such an edge, F_{jt}, is lower than λ_j then the admission control must ensure that only F_{jt}/λ_j fraction of the requests from client j are accepted. Furthermore we can put a lower bound on the flow through edges (j, t) such that at least a given fraction of client j's requests are accepted which is decremented iteratively in some greedy fashion until we find a feasible solution.

3.5 SLA Types

3.5.1 Throughput Constraints

Throughput constrained SLA is the simplest form of SLA, where a client pays a fixed price for meeting its throughput requirement. Since the price paid is fixed the hosting center's profit is purely a function of its energy consumption. Hence, the objective of profit maximization translates into energy minimization.

In NFRA, throughput constrained SLA is formulated as follows. Throughput requirement of client j is stipulated by λ_j. Now the maximum throughput provided by a server of pool i for client j is given by μ_{ij}, the service rate. Therefore we simply set the gain factors $\gamma_{ij}=\mu_{ij}$, thus providing maximum possible throughput and forcing servers to operate at 100% utilization. Correspondingly the edge costs are set to be $\kappa_{ij}=e_{ij}\mu_{ij}$. This cost essentially represents the power cost of servicing μ_{ij} requests per second for client j on a server from pool i. Finding the min cost max flow in G thus coincides with minimizing average power (maximizing expected profit) while meeting the throughput requirement.

3.5.2 Average Response Time Constraint

Average response time constraint SLA stipulates that the average per request response time, $\tau_{j,avg}$, for requests of client j under a given arrival rate λ_j shall never exceed $\tau_{j,max}$. The client pays a fixed price for meeting the average response time constraint. Here the objective of profit maximization translates into energy minimization while still honoring the response time requirement. Note that the response time is a function of system utilization and, based on queuing theory, as utilization reduces response time decreases. However operating servers at lower utilization results in, 1) increased number of active servers and 2) increased energy cost due to lack of energy proportionality.

In order to understand the impact of utilization on server energy let us look at how energy, for a unit of work done, scales as a function of server utilization. Assume that a server can process 100 requests/second at 100% utilization while consuming energy $P_{(1)}$. Per request energy consumption is given as $P_{(1)}/100$. Consider that the server performance scales linearly with utilization, the same server when operating at 80% utilization can satisfy only 80 requests/second consuming $P_{(0.8)}$ power. Per request energy consumption is given as $P_{(0.8)}/80$. The ratio of energy per request at 80% utilization to energy per request at 100% utilization is defined as *Energy Increase Factor* (EIF) representing the scaling of energy per unit of work at different utilizations with respect to 100% utilization. More generally EIF at utilization U is given as, EIF_U, in eq. (3),

$$EIF_U = \frac{P_{(U)}}{P_{(1)}U} \tag{3}$$

where $P_{(U)}$ is power at utilization U and $P_{(1)}$ is power at 100% utilization. System is most energy efficient when EIF=1 which is at U=1. Fig. 4 plots EIF on the primary Y-axis and scaling of power on secondary Y-axis vs. utilization. The data for Fig. 4 was obtained from the SPEC website for SPECWeb2009 [24] Power benchmarks for the server configuration, HP_C, detailed in the experimental results section. Even for more recent generation of servers (HP proliant [28]) the idle power is about 50%. Note that the power value $P_{(U)}$ at different U's was obtained using linear regression on the power data resulting in $P_{(U)}= mU + P_{idle}$ with m=102.5 and P_{idle}=238.5 where P_{idle} represents the idle state power. As shown in the figure, EIF increases super-linearly as utilization decreases. This analysis shows that there is a tradeoff between increasing utilization to reduce the energy cost and reducing utilization to meet response time constraint.

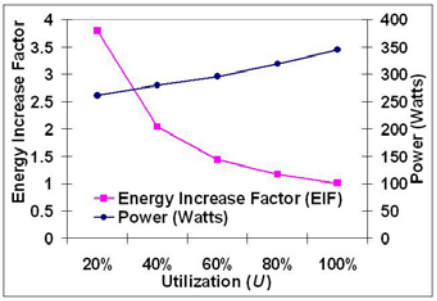

Fig. 4 Power and per Request Energy Consumption

In order to account for this tradeoff let us first look at the average per request response time of client j's requests on a server from server pool i as a function server utilization. Let us denote this average per request response time by $\tau_{ij,avg}$. $\tau_{ij,avg}$ depends on the queuing model and distribution of service rate μ_{ij} and arrival rate λ_j. Given that we are considering single server queuing system, shown in Fig. 1, if arrival and service rates are Poisson distributed (M/M/1 queue),then $\tau_{ij,avg}$ is given as [20]:

$$\tau_{ij,avg} = \frac{1}{\mu_{ij}-\lambda_{ij}} = \frac{1}{\mu_{ij}\left(1-U_{ij}\right)} \tag{4}$$

where, λ_{ij} denotes the effective arrival rate of client j's requests to a server of pool i, i.e., the number of client j's requests that are assigned to a server of pool i, and $U_{ij}=\lambda_{ij}/\mu_{ij}$ denotes the effective per server utilization for client j on server pool i. If we want to make sure that $\tau_{ij,avg} \le \tau_{j,max}$, then from equation (4) we can upper bound the per server utilization U_{ij} as follows:

$$U_{ij} \le 1 - \frac{1}{\mu_{ij}\tau_{j,\max}} \tag{5}$$

$$U_{ij,\max} = 1 - \frac{1}{\mu_{ij}\tau_{j,\max}} \tag{6}$$

Since energy efficiency of servers reduces at lower utilization, we set the utilization U_{ij} to be equal to $U_{ij,max}$ as derived in eq. 6. Obviously a reduction in utilization comes at the expense of reduced effective service rate, $\mu_{ij'}$. Thus, in order to account for the average response time constraint, we modify the service rate as shown in eq. (7). Once the service rate is updated we set the gain

factors γ_{ij} to this new service rate. Note that, by enforcing the upper bound on utilization on every server, the average response time constraint is met on every server and is thus met overall.

$$\gamma_{ij} = \mu'_{ij} = U_{ij,\max}\mu_{ij} \quad (7)$$

The reduction in server utilization results in increased per request energy cost due to non energy proportionality. We initially characterized e_{ij} at utilization U=1. We account for increased e'_{ij} as follows:

$$e'_{ij} = e_{ij}EIF_{U_{ij,\max}} = e_{ij}\left(\frac{P_{ij,(U_{ij,\max})}}{P_{ij,(1)}U_{ij,\max}}\right) \quad (8)$$

where $P_{ij,(U)}$ is per server power dissipation value at utilization U, $P_{ij,(1)}$ is per server power at U=1 for server pool i and client j. Using e'_{ij} we compute the value for κ_{ij} as $\kappa_{ij}=e'_{ij}\gamma'_{ij}$.

Now we have updated all the necessary variables of our initial generalized network flow formulation, in order to account for the average response time constraints. Using this new set of variables finding the min cost max flow in G will result in an average response time constrained energy minimizing resource allocation.

Note that eq. (4) and subsequent analysis assumed the M/M/1 model. However we can use any other distribution of service and arrival rate to obtain upper bound on utilization. For example under G/G/1 model we can derive the upper bound on utilization as follows [20]:

$$\tau_{ij} = \frac{U_{ij}}{(1-U_{ij})}\left(\frac{C_A^2 + C_A^2}{2}\right)\frac{1}{\mu_{ij}} + \frac{1}{\mu_{ij}}$$

$$U_{ij,\max} = \frac{1}{1+\left(C_A^2 + C_A^2/2\left(\tau_{i,avg}\mu_{ij} - 1\right)\right)}$$

where C_A and C_B are coefficients of variation of request inter-arrival times and service times, respectively. Rest of the analysis would use this new upper bound to compute remaining variables.

One point is worth mentioning here. Since the min cost max flow in G can yield fractional solution, some server in server pool i might be shared among multiple clients. If each client demands a different utilization (U_{ij}) level from the same server, then it may seem that the computation of the energy cost e'_{ij}, based on a given client utilization level, may not hold. However, if the server is shared in a time multiplexed fashion, then we shall compute e'_{ij} at a fine time granularity only accounting for the duration of time a client j is assigned to resource i. For example, if a server is equally shared among two clients, and during the first half of a second one client uses server at 10% utilization while during the second half the other client uses it at 70% utilization, then for each half second we can compute e'_{ij} for the two clients with the corresponding utilization levels. Given the assumption that the number of service requests is much larger than the available resources, such selection, on average, will not impact the end result much.

3.5.3 Stochastic Maximum Response Time Constraint

This is the third and the most complex SLA type we will use to demonstrate the effectiveness of NFRA. Under this SLA type, a given client j stipulates its response time requirement as follows: No more than β_j fraction of requests shall violate the maximum response time $\tau_{j,max}$, i.e., $\Pr(\tau_j > \tau_{j,max}) \le \beta_j$, where τ_j is the response time of a request of client j. As can be inferred, the client cares about the response time of every request and therefore the distribution of request response times and not just the average response time as was the case in the previous SLA (cf. Section 3.5.2). We propose to meet this response time constraint by ensuring that the constraint is met at every server of pool i to which client j is mapped, much like the solution for average response time constraint. This translates into the following requirement: For a given client j and server pool i, the per request response time τ_{ij} shall not violate the maximum tolerable response time $\tau_{j,max}$ for more than β_j fraction of the requests: i.e., $\Pr(\tau_{ij} > \tau_{j,max}) \le \beta_j$. Hence by imposing β_j of client j on every server pool i we ensure that the probabilistic response time guarantee is met at every pool i and hence is met overall. For our single server queuing model, this probability can be upper bounded as shown in eq. (9), for M/M/1 queues [20],

$$\Pr(\tau_{ij} > \tau_{j,\max}) \le e^{-(\mu_{ij}-\lambda_{ij})\tau_{j,\max}} = e^{-(1-U_{ij})\mu_{ij}\tau_{j,\max}} \le \beta_j \quad (9)$$

Using the upper bound from eq. (9) to meet the SLA, we obtain the effective utilization using eq. (10):

$$U_{ij} \le 1 + \left(\ln \beta_j / \mu_{ij}\tau_{j,\max}\right) \quad (10)$$

3.5.3.1 Energy Minimization

We will consider two variants under this SLA. In the first variant we assume that the client pays the hosting center a fixed total (lump sum) price, as long as the hosting center abides by this SLA. For instance, SLA stipulates that 95% of a client requests shall be completed within 10 milliseconds (ms). Here, never does the client pay any incentive if ***more*** than 95% of the requests are satisfied within 10ms; similarly, the hosting center pays no penalties as long as 95% of the requests are completed within 10ms. Given this scenario the hosting center strives to satisfy 95% of the requests within 10ms – no more, no less. Thus, the objective of our resource allocation problem in this scenario is of energy minimization.

Since the energy efficiency of servers reduces at lower utilization, as mentioned earlier, we set the utilization to the upper bound provided in eq. (10) as shown in eq. (11).

$$U_{ij,\max} = 1 + \left(\ln \beta_j / \mu_{ij}\tau_{j,\max}\right) \quad (11)$$

Obviously a reduction in utilization comes at the expense of reduced effective service rate. Thus, in order to account for the response time constraint, we modify the service rate, and therefore the gain factor, as before, as shown in eq. (7). Note that, since the probabilistic response time guarantee of eq. (9) is

provided at every server of pool i to which client j is mapped, the same guarantee is also provided by server pool i as a whole. Furthermore, we must account for the increased per request energy cost due to reduced server utilization, as before. We account for e_{ij} increase as shown in eq. (8) and subsequently update the edge costs to κ_{ij} as before. In essence, once $U_{ij,max}$ is computed from eq. (11) the rest of variables for the network flow are computed exactly the same way as we did in Section 3.5.2. Finding the min cost max flow will result in stochastic maximum response time constrained energy minimizing resource allocation.

Note that, in order to obtain $U_{ij,max}$ we assumed M/M/1 queuing model with exponential service times. However, for general service time distributions, eq. (9) can be used as an approximation when service times have heavy tail distribution [11]. In general, NFRA is independent of queuing model used (M/M/1 or M/M/2 or any other) and only requires a suitable upper bound be found on U_{ij} to meet SLA.

3.5.3.2 Profit Maximization

So far we looked at resource allocation that minimizes energy consumption assuming the client pays a fixed price as long as the stipulated SLAs are met. However, when the price paid by the client is dependent on the quality of service received, the problem of resource allocation becomes more complex. In this last problem formulation we assume that the client pays a fixed ***per*** request price, instead of a fixed total (lump sum) price, as long as the hosting center provides an agreed upon probabilistic response time guarantee. However, while providing probabilistic guarantee, the hosting center pays per request penalty to the client whenever the response time of a request violates the stipulated response time requirement. For example, SLA may stipulate ***at least*** 95% of all requests from a particular client shall be completed within 10ms ***and*** whenever requests are not completed within 10ms the hosting center may pay a penalty to the client. In this scenario a client pays for every request that meets response time requirement, while the hosting center pays a penalty for every request that fails to meet the response time requirement. Thus hosting center has an incentive to meet the maximum response time requirement of more than 95% of the requests since this will decrease the penalties hosting center has to pay and increase the revenues. Thus allocating more resources can reduce the penalty and hence generate more revenue, but it can also adversely affect the energy cost. Therefore profit maximization must account for generated revenue and energy cost.

Before formulating the problem as generalized network flow, we will show the relationship between utilization and profit. Let us assume that request arrival rate λ and service rate μ are Poisson with $\lambda \leq \mu$. The ratio of λ to μ gives us the utilization, U, of the system. As defined earlier, the price paid per request is R as long as the response time $\tau \leq \tau_{max}$ and when $\tau > \tau_{max}$ hosting center pays a penalty of L ($L \geq 0$). Let σ represent the ratio of τ_{max} to service time $1/\mu$, i.e., $\sigma=\mu\tau_{max}$. Assuming M/M/1 model, we can calculate an upper bound on the probability of failure, β, i.e., $\Pr(\tau > \tau_{max})$, as: $\beta = e^{-(\mu-\lambda)\tau_{\max}} = e^{-(1-U)\sigma}$. Given the value of β, the net price paid per request, π, is given by eq. (12).

$$\pi = R(1-\beta) - L\beta \tag{12}$$

The net profit per request Φ is obtained as shown in eq. (13) where e is the energy cost. Note that, for a given server, Φ, π and e are functions of the server utilization.

$$\phi = \pi - e \tag{13}$$

We quantitatively demonstrate how net profit varies with utilization. Assume that the hosting center operator sets 300% profit margin over the energy cost, e_1, of servicing a request at utilization U=1. Thus the price paid per request, R, is 3 times e_1. Let L=R. We obtained e at different U's using eq. (3) and data of Fig. 4. The graph of net profit per request vs. utilization for σ values of 5, 10 and 15 is shown in Fig. 5.

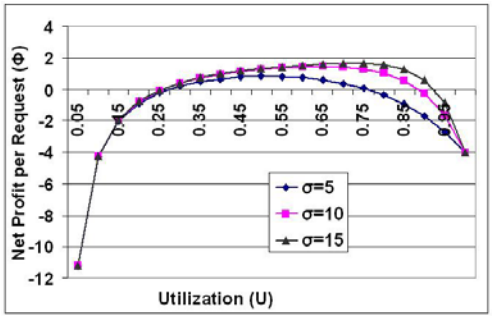

Fig. 5 Net Profit per Request at Different Utilization Levels.

Lemma 1: The Profit function Φ of eq. (13) is a concave downwards function.

Proof: Expanding the function Φ we get the eq. (14).

$$\phi = (R(1-e^{-(1-U)\sigma}) - Le^{-(1-U)\sigma}) - \left((mU + P_{idle})/(P_{(1)} \bullet U)\right) \tag{14}$$

Without loss of generality let us assume R=L. Let us denote $P_{idle}/P_{(1)}$=A, which is a constant for any given server. Taking first and second order derivate of Φ with respect to U we get eq. (15) and (16) respectively.

$$\frac{\partial \phi}{\partial U} = -2R\sigma e^{-(1-U)\sigma} + AU^{-2} \tag{15}$$

$$\frac{\partial^2 \phi}{\partial U^2} = -2(R\sigma^2 e^{-(1-U)\sigma} + AU^{-3}) \tag{16}$$

Given that $A \geq 0$, $R \geq 0$ and $\sigma \geq 0$ the second order derivative of Φ with respect to U is negative. Therefore function Φ is concave downwards. ■

Thus when system utilization is high the energy cost per request is low but β (fraction of SLA violations) increases, thereby causing hosting center to pay penalties to the client. Conversely when utilization is low, β may be lower but energy costs increase. Thus net profit is a complex interplay between system utilization, energy costs, and probability of missing response time constraint. Under such a scenario, profit maximizing resource allocation needs to account for utilization dependent per request profit, available resources and response time constraint.

Response time constraint puts an upper bound on server utilization as shown in eq. (10). Therefore for profit maximization, per server utilization of client j on server pool i must lie in [0, $U_{ij,max}$]. If we can shrink the search range [0, $U_{ij,max}$] by providing a better lower bound for utilization without compromising the optimality, then the efficiency of our resource allocation algorithm can be improved. Here we obtain one such lower bound and establish a theorem of optimality. Let us denote the utilization, at which the per request profit function, Φ_{ij}, for a server pool and client pair (i, j), achieves its maximum value by $U_{ij,lopt}$. $U_{ij,lopt}$ is server pool and client pair (i, j)'s local optima for maximum profit. We establish following theorem.

Theorem 1: In the optimal solution to the resource allocation for maximum profit problem, the utilization level for any server pool and client pair (i, j) can never lie below utilization $U_{ij,lopt}$.

Proof: Let us assume that the optimal solution uses some utilization $U_{ij,*}$ for server pool and client pair (i, j) such that $U_{ij,*} < U_{ij,lopt}$. For this $U_{ij,*}$ the corresponding net profit per request is $\Phi_{ij,*}$. Due to concave (downwards) nature of net profit per request curve Φ (Lemma 1) we know that the following relation holds:

$$\left(\partial \phi_{ij} / \partial U\right)\Big|_{U_{ij,*}} > 0 .$$

This implies that at utilizations higher than $U_{ij,*}$ but lower than $U_{ij,lopt}$ the net profit per request is strictly increasing. Now then if we had some fraction $x_{ij,*}$ of server pool i allocated to client j to support utilization $U_{ij,*}$, then we know with certainty that a fraction smaller than $x_{ij,*}$ can be used (but at higher utilization) to obtain higher net profit per request. Note that while we use a fraction smaller than $x_{ij,*}$ at higher utilization to obtain higher per request profit, all the other x_{yz}'s, $y \neq i$ and $z \neq j$, remain unchanged. Therefore rest of the solution does not get affected. Hence for $U_{ij,*} < U_{ij,lopt}$ and for corresponding $x_{ij,*}$ we can safely use a fraction smaller than $x_{ij,*}$ at higher utilization to improve the solution. This contradicts our assumption that the optimal solution lies at some optimal utilization $U_{ij,*}$ such that $U_{ij,*} < U_{ij,lopt}$. ■

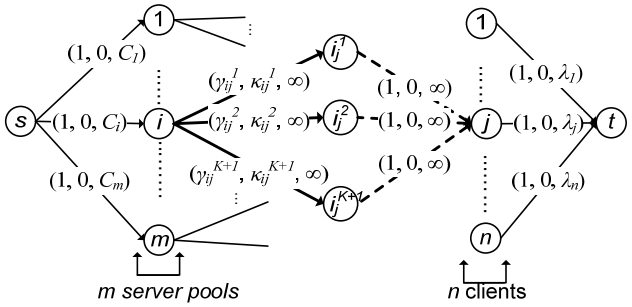

Fig. 6 Edge Splitting for Profit Optimization.

Based on theorem 1 we can establish that the optimal solution must lie between $U_{ij,lopt}$ and $U_{ij,max}$ of each server pool and client pair (i, j) as long as we allocate some resources of pool i to client j. Note that the utilization level for pair (i, j) introduces a new decision variable in our resource allocation problem and hence we must find not only what fraction of pool i to allocate to client j but also at what utilization level. In our problem formulation we account for decision variables corresponding to different utilization levels in the following way: For each server pool and client pair (i, j) we split the edge (i, j) into multiple edges corresponding to different utilization values that lie between $U_{ij,lopt}$ and $U_{ij,max}$. Because of this discretization, the new problem formulation will only be able to provide an approximate solution.

Fig. 6 shows the new graph G′. Figure shows the splitting of one such edge (i, j) where we divide the range [$U_{ij,lopt}$, $U_{ij,max}$] into K levels and add $K+1$ nodes between node i and node j. We denote K as the edge splitting cardinality. Given that the optimal solution lies in [$U_{ij,lopt}$, $U_{ij,max}$] we are essentially providing an approximation subject to the quantization error introduced due to edge splitting cardinality K. The granularity of sweeping the range [$U_{ij,lopt}$, $U_{ij,max}$] is only impeded by the computational complexity. Fine granularity, i.e. bigger edge splitting cardinality, improves the solution but increases the computational time. Let us denote each of these $K+1$ nodes by i_j^k for $k=1,..,\ K+1$ and assign them a utilization level U_{ij}^k. Then for each edge (i_j^k, j), shown by dotted thick edges in Fig. 6, we set the parameters (γ, κ, u)=(1, 0, ∞) whereas for each edge (i, i_j^k), shown by solid thick edges, we set the corresponding parameters as shown in eq. (17).

$$U_{ij}^{\ k} = U_{ij,lopt} + \left(\frac{U_{ij,\max} - U_{ij,lopt}}{K}\right)(k-1)$$
$$\gamma_{ij}^{\ k} = \mu_{ij}^{\ k} = U_{ij}^{\ k}\mu_{ij}\ ;\ e_{ij}^{\ k} = e_{ij}\left(\frac{P_{ij,(U_{ij}^{\ k})}}{P_{ij,(1)}\,U_{ij}^{\ k}}\right) \quad (17)$$

Furthermore for each edge (i, i_j^k) with utilization U_{ij}^k, π_{ij}^k, the expected price paid per request by client j at utilization U_{ij}^k, changes as well. In order to determine π_{ij}^k let us denote τ_{ij}^k to be the response time of client j on server pool i at utilization U_{ij}^k. Then eq. (18) follows. Based on eq. (18) we calculate π_{ij}^k and the net profit per request, Φ_{ij}^k, as shown in eq. (19) and (20) respectively. With this new formulation, from the perspective of server pools, each of the newly added nodes is considered a separate client. Moreover, to maximize profit by solving a min cost max flow problem in G', the cost on the edges, κ, must be set such that when we minimize the cost, we maximize the profit. Hence we modify the edge costs as shown in eq. (21).

$$\Pr(\tau_{ij}^{\ k} > \tau_{j,\max}) = \beta_{ij}^{\ k} = e^{-(1-U_{ij}^{\ k})\mu_{ij}\tau_{j,\max}} \quad (18)$$

$$\pi_{ij}^{\ k} = R_j(1-\beta_{ij}^{\ k}) - L_j\beta_{ij}^{\ k} \quad (19)$$

$$\phi_{ij}^{\ k} = \pi_{ij}^{\ k} - e_{ij}^{\ k} \quad (20)$$

$$\kappa_{ij}^{\ k} = -\phi_{ij}^{\ k}\gamma_{ij}^{\ k} \quad (21)$$

It is worthwhile mentioning that the new formulation presented by graph G' provides us greater flexibility. For example, much like the price function used in [12], if we were to model R_j as a step function of the response time, τ_{ij}^k, of client j on server pool i for a given utilization U_{ij}^k, then we can modify eq. (18) to account for the new price

paid for the improved response time instead of the fixed price R_j. Note that so far we assumed that $U_{ij,lopt}$ was given. In practice, to find $U_{ij,lopt}$ we need to solve $\partial\phi_{ij}/\partial U = 0$. We use Newton's method to find $U_{ij,lopt}$.

- ***Iterative Edge Splitting***

Note that, a given fixed value of edge splitting cardinality, K, may not be applicable to all the scenarios. Hence it is advisable to iteratively increase the value of K until the incremental benefit between two consecutive iterations falls below a certain threshold. While such an iterative approach is useful, it is worthwhile mentioning that as we increase the value of K and therefore increase the number of utilization levels being introduced between pair (i, j), they shall be chosen in a manner such that they are inclusive of the U_{ij}^k values from the previous iteration. This means that if for an iteration with $K=K_1$ we chose K_1 utilization levels between pair (i, j) then in the next iteration with $K=K_2$, $K_2 > K_1$, the K_2 utilization levels between the pair (i, j) must include the same K_1 utilization levels from the previous iteration and introduce only $K_2 - K_1$ new utilization levels. We call such growth in K, *inclusive growth* of *edge splitting cardinality*. The *inclusive growth* of K is necessary to ensure that the new decision points, in the form of new utilization levels that are being introduced, contain all the decision points from the previous iteration and a few new ones. Thus, we ensure that our solution always improves.

3.6 Solving Min Cost Max Flow in NFRA

For the min cost max flow and max flow problem for generalized network, the fastest known polynomial time algorithm is based on interior point linear programming (LP) methods [19] ($O(m^{1.5}n^2\log(nB))$). Recently in [18] Wayne et al. proposed the first polynomial time combinatorial algorithm for min cost circulation with time complexity of $O(m^3n^3\log B)$. Given the complexity of combinatorial approaches we resort to LP to find min cost max flow solutions presented in this paper.

4. Experimental Results

4.1 Hosting Center and SLA Parameters

Before conducting our experimental evaluations we explain how we selected the model parameters described in Section 2. For server heterogeneity we considered four types of server pools (m=4), each being categorized by a three tuple {CPU frequency, memory and disk size}. The four server pools are *HP_A* = {3.2GHz, 96GB, 2TB}; *HP_B* = {3.2GHz, 48GB, 1.5TB Flash}; *HP_C* = {2.3GHz, 48GB, 1.9TB}; *HP_D* = {2.3GHz, 48GB, 1.1TB Flash}. *HP_B* and *HP_D* use flash drives, instead of traditional hard drives, which consume less power per read. Instead of arbitrarily selecting the server characteristics we selected servers used in reporting SPECWeb2009 [24] results by various vendors in the near past.

Briefly, SPECWeb2009 were devised for next generation web based workloads and consist of four different workload types: Banking, E-commerce, Support and Power. Power workload characterizes server power at different utilizations for E-commerce workload. The other three workloads operate at 100% utilization by issuing as many requests as needed to saturate the system utilization. The request rates from these workloads follow Poisson distribution. We looked up the published SPECWeb2009 Power benchmark results to obtain the power consumption at five different utilization levels. We then used linear interpolation to derive the EIF value at any utilization. We treated SPECWeb2009 Banking, E-Commerce and Support workloads as three different customer types. We again relied on the published results to obtain the energy per request of each workload type on each server type (e_{ij}) and the per request execution time (T_{ij}) on each server type using the profiling methodology presented in section 2.

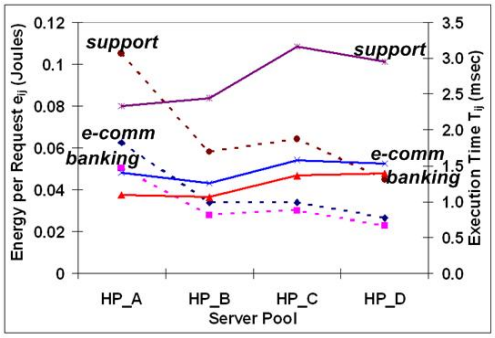

Fig. 7 Energy and Response Time Characterization.

Fig. 7 shows e_{ij} on primary Y-axis (dotted lines) and T_{ij} on the secondary Y-axis, for the three workload types (equivalent to clients in Fig. 1) and the four server types (equivalent to server pools in Fig. 1). Note that T_{ij} values for different workload types do not necessarily scale in a similar manner across different server types. For example, for E-commerce and Banking workloads T_{ij} value reduces from pool *HP_A* to *HP_B*, while it increases for Support. Thus the heterogeneity in servers can alter workload behaviors (e_{ij} and T_{ij}) in a complex manner.

We created two hosting center setups using the four server types by choosing the corresponding C_i values (number of servers in each pool) as follows: config_A: {*HP_A, HP_B, HP_C, HP_D*}={10, 10, 10, 10} and config_B={9, 9, 11, 11}. We also assume that we have three Banking clients, three E-commerce clients and two Support clients, a total of eight clients, i.e., n=8. We formulate three different instantiations using these eight clients, called *Instance_1*, *Instance_2* and *Instance_3* workloads. *Instance_1* is banking dominated, i.e., load generated due to banking clients dominates. Similarly *Instance_2* and *Instance_3* workloads are E-commerce and Support dominated, respectively. Table 1 shows the SLA parameters for the eight clients for *Instance_1*. The clients within a particular instance differ in the number of requests sent to the hosting center. For example, small scale (SS) Banking client generates 70K requests per second while large scale (LS) Banking client generates 98K requests per second, 40% more. We also assume that E-commerce and Banking clients have more stringent response time requirements than Support clients. Furthermore, the SLA specification for LS clients has lower tolerance for response time violation and hence has

lower β_j. But they are also willing to pay a premium, in terms of higher profit margins for the hosting center, denoted by M_j in Table 1, representing the profit margin percent over average per request energy cost across all server pools. Furthermore we assume that R_j=L_j.

It is worth noting that the values we selected for various parameters shown in Table 1 roughly reflect the relative importance of clients. The absolute values were arbitrarily selected while preserving the relative importance. Note that client SLA parameters across different instantiations change only in terms of arrival rates leaving rest of the parameters, i.e., $\tau_{j,max}$, β_j, M_j, unchanged. For the sake of brevity we omit the arrival rates of other instance types. Furthermore, in the interest of space, experimental results presented here assume Poisson arrival and service rates. Note that the framework itself can support other distributions as mentioned earlier. We implemented the min cost max flow using the LP package lpsolve [26].

Table 1: Client SLA specification for *Instance_1*

Client Type	*J*	λ_j	$\tau_{j,max}$ *(msec)*	β_j	M_j
Bank (LS)	*1*	*98K*	*15*	*0.05*	*80*
Bank (MS)	*2*	*82K*	*15*	*0.08*	*60*
Bank (SS)	*3*	*70K*	*15*	*0.1*	*40*
Ecomm (LS)	*4*	*41K*	*12*	*0.05*	*80*
Ecomm(MS)	*5*	*36K*	*12*	*0.08*	*60*
Ecomm (MS)	*6*	*20K*	*12*	*0.1*	*40*
Support (MS)	*7*	*8K*	*120*	*0.6*	*60*
Support (SS)	*8*	*4K*	*120*	*0.8*	*50*

4.2 Psuedo Optimal Solution

NFRA generates conservative resource assignments by ensuring that a constraint is met at every server of pool i to which client j is mapped. For instance, NFRA ensures $\tau_{ij,avg} \leq \tau_{j,max}$ rather than $\tau_{j,avg} \leq \tau_{j,max}$ for average response time constraint SLA. In an optimal solution this more restrictive constraint is unnecessary since at some servers the average response time can violate the stipulated response time requirement while making up for it at some other servers. In order to quantify the effect of this sub-optimal utilization selection of NFRA we compare NFRA with two approaches: (1) greedy and (2) pseudo optimal. Next we give a brief overview of the pseudo optimal solution, which is applicable to all the SLA types except the throughput constrained SLA for which NFRA already finds the optimal solution. Let us look at the pseudo optimal solution for average response time SLA, which is equally applicable to stochastic maximum response time SLA.

Since in optimal solution constraints are not imposed conservatively on every server of a pool, pseudo optimal solution treats every server in a pool as a different resource and tries to find the optimal resource allocation between every server and client pair, rather than for a server pool and client pair like NFRA. Second, an optimal solution does not need to set an upper bound on the utilization of any server, as done in NFRA. In essence, in an optimal solution any server can be allocated to any client at any utilization as long as the SLA constraint is met over all the servers. Therefore each individual server client pair must sweep the whole range of utilization in order to search for the optimal solution. In the pseudo optimal solution we divide the whole range of utilization (0, 1) into multiple levels. This discretization of utilization level makes the solution pseudo optimal instead of truly optimal. The following LP formulation models our pseudo optimal solution.

$$Min \sum_j \sum_i \sum_k P_{jik}(U_{jik}) x_{jik}$$

$$s.t. \quad \sum_i \sum_k \tau_{jik,avg}(U_{jik}) \left(U_{jik} \mu_{jik} x_{jik} / \lambda_j \right) \leq \tau_{j,\max}, \forall j$$

$$\sum_i \sum_k U_{jik} \mu_{jik} x_{jik} = \lambda_j, \forall j \quad \& \quad \sum_{j,k} x_{jik} \leq 1, \forall i$$

where, x_{jik} is the decision variable corresponding to client j on some server i at utilization level k (U_{jik}), denoting what fraction of server i at utilization level k is allocated to client j. P_{jik} denotes the power corresponding to the same triplet (j, i, k) at utilization U_{jik}. The first constraint is the average response time constraint which takes the weighted average of the average response times $\tau_{jik,avg}$ for triplet (j, i, k) at utilization, U_{jik}. $\tau_{jik,avg}$ is a function of U_{jik} (cf. eq. (4)). $\tau_{jik,avg}$ is weighted with respect to the fraction client j's requests that are served by server i at utilization level k, U_{jik}. The second constraint is the throughput constraint for each client j and third constraint is capacity constraint for each server i. We refer to this solution as Pseudo-opt. Note that we can easily replace $\tau_{jik,avg}$ with β_{jik} for stochastic maximum response time constraint and update the objective function for profit maximization. In the interest of space we omit the details of those formulations.

Two serious concerns for implementing Pseudo-opt are worth mentioning. 1) It treats each server in a pool individually causing a huge explosion in the state space that needs to be searched between all client-server pairs. 2) The utilization range (0, 1) is split into multiple levels and hence the search space multiplies by the number of levels. The net result of this explosion is that Pseudo-opt even for our small hosting center configurations runs approximately 75,000X slower than NFRA.

4.3 Throughput Constrained Optimization

The greedy approach implemented for throughput constrained energy minimization sorts energy per request values, e_{ij}, corresponding to client j and server pool i, in non-decreasing order. From this sorted list it picks each server pool and client pair (i, j), one at a time, and allocates resources so as to meet the throughput requirement.

Fig. 8 shows comparison between Greedy and NFRA. Note that Greedy is not able to find a feasible solution corresponding to *config_B* for *Instance_1* and *Instance_2* workloads. This means that even though we had enough servers to be allocated to clients so as to satisfy the throughput requirement, Greedy was unable to find a mapping. When Greedy does find a feasible solution, the energy consumption of NFRA is 3.9% lower than Greedy.

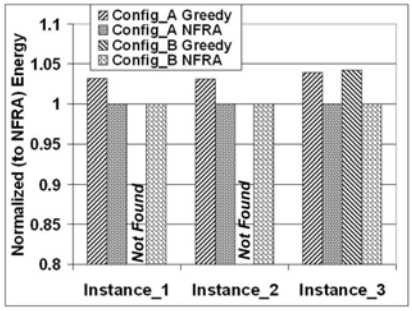

Fig. 8 Throughput Constrained Energy Optimization.

We can intuitively explain why greedy solution, when it works, is not as worse, as one would expect. Referring to the per request energy characterization of Fig. 7, the trend in per request energy across four different server pools remains similar for all three workload types. Hence, the choice of which server pool to use is not very different between Greedy and NFRA. However, as the heterogeneity of the server pools and workloads increases Greedy would perform worse, as we will shortly demonstrate. Second it is critical to note that NFRA always finds a feasible solution whenever such a solution exists. Hence, when resources are provisioned for near-peak demand then NFRA will find a feasible solution while Greedy will fail. When resources are scarce due server downtimes or failures and/or maintenance, Greedy may have to employ admission control even though enough resources are available. Note that, due to heterogeneous server clusters, there is no closed form equation that dictates the capacity requirement for a given set of clients and hence the existence of a feasible solution is non-trivial to check for Greedy.

4.4 Average Response Time Constrained Optimization

In this section we present the results for average response time constrained energy optimization and compare it with Greedy and Pseudo-opt approaches. Greedy works similar to the one used for throughput constrained optimization; once a pair (i, j) with the smallest e_{ij} is picked, the decision regarding how many requests of client j are assigned to server pool i is made using the $U_{ij,max}$ (eq. (6)) that that satisfies the average response time constraint.

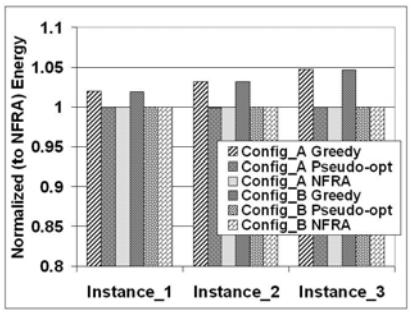

Fig. 9 Average Response Time Constrained Energy Optimization.

Fig. 9 shows the comparison of our approach with Greedy and Pseudo-opt. Here, Greedy always finds a solution and is within 5% margin of NFRA with NFRA being, on average, 3.3% better than Greedy. The reason behind such a small difference in energy optimization between Greedy and NFRA is, once again, the trend in energy consumption as mentioned in previous section. With respect to Pseudo-opt solution, NFRA results are within 0.1% bound of Pseudo-opt. Hence our slightly conservative resource allocation does not negatively impact profit much. However, the difference in execution time is dramatic. NFRA on average took about 0.004 seconds to complete resource allocation while Psuedo-opt takes 300 seconds, a 75,000X slowdown.

4.5 Stochastic Maximum Response Time Constrained Optimization

4.5.1 Energy Optimization

In this section we present the results for stochastic maximum response time constrained energy optimization. The Greedy approach implemented here is similar to the Greedy approach for average response time constrained energy optimization except that $U_{ij,max}$ is derived using eq. (11) that satisfies stochastic maximum response time constraint.

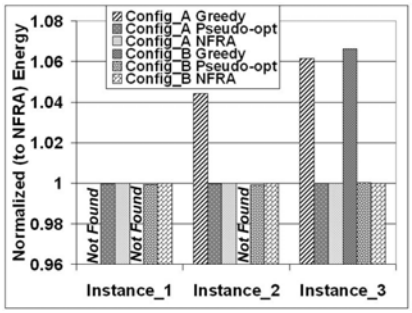

Fig. 10 Stochastic Maximum Response Time Constrained Energy Optimization.

Fig. 10 shows the results using response time constraints specified in Table 1. As shown in the figure, Greedy is about 6% worse than NFRA, if Greedy is able to find a solution. Once again note that NFRA always finds a feasible solution while Greedy fails in some instances. Second, the reason why Greedy is only 6% worse than NFRA is due to similar trend in per request energy of the workloads across four server pools as mentioned earlier. Fig. 10 also shows the results of Pseudo-opt. As shown, once again, NFRA results are within 0.1% bound of the Pseudo-opt. Thus NFRA achieves results that are very close to optimal at much lower execution time overhead.

4.5.2 Profit Maximization

For profit maximization, the greedy approach sorts (i, j) pairs according to the per request profit in non-increasing order, where profit is obtained by subtracting e_{ij} values from R_j. Once a pair (i, j) is picked it greedily picks utilization value that, while satisfying response time constraint dictated by $U_{ij,max}$, maximizes per request profit, i.e., $U_{ij,lopt}$, and accordingly assigns client j's requests to server pool i. Fig. 11 shows the results for the profit maximization, where the result of Greedy is normalized to the result of NFRA. Here we set the edge splitting cardinality, K=8.

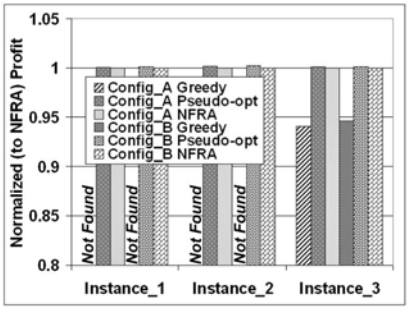

Fig. 11 Profit Optimization.

As shown in the figure, NFRA achieves, on average, 9.4% more profit than Greedy for *Instance_3* workload while Greedy fails to find any feasible solutions for *Instance_1* and *Instance_2* workloads. Furthermore, Fig. 11 also shows the Pseudo-opt results normalized to NFRA which is again within 0.1% bound.

The superiority of NFRA over Greedy may be highlighted with more heterogeneous setups when different client requests exhibit stark execution time differences on different server pools. To demonstrate this aspect we chose five different hosting center setups with two server pools consisting of *HP_A* and *HP_B* servers. The five hosting center combinations are (# of *HP_A* servers, # of *HP_B* servers): a) (30, 10), b) (25, 15), c) (20, 20), d) (15, 25) e) (10, 30). We chose only two clients, E-commerce and Support, due to their stark difference in T_{ij} values across *HP_A* and *HP_B* servers. Using the same setup we also demonstrate the importance of accounting for non energy proportionality. Here we modified our profit maximizing solution such that it does not scale per request energy, e_{ij}, for client j on server pool i. Hence the profit function Φ_{ij} of eq. (20) only accounts for the utilization dependent price, $\pi_{ij}{}^{k}$, while keeping the energy cost constant to e_{ij}, independent of utilization. Since the energy cost is constant we will not have a concave downward function (cf. Fig. 5). Hence in this new, energy proportionality oblivious, profit maximizing solution we must consider the full range of server utilization, i.e., (0, 1). We divided this whole range for each server pool and client pair in K=64 levels.

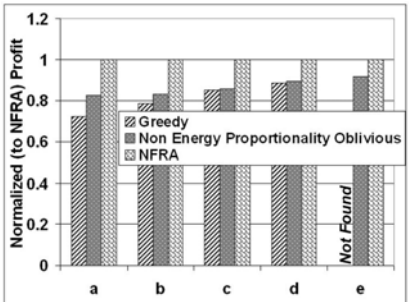

Fig. 12 Impact of Large Scale Heterogeneity.

Fig. 12 shows the results with NFRA performing 38.8, 27.3, 17.5, and 12.9% better than Greedy for configuration a, b, c, and d respectively. For configuration e Greedy does not find a solution. With respect to non energy proportionality oblivious solution, NFRA performs 17.4, 16.9, 14.4, 10.7 and 8.2% better for configuration a, b, c, d and e respectively. The results show that in the presence of large number of energy inefficient servers 1) NFRA outperforms Greedy and by a much larger margin 2) accounting for non energy proportionality becomes important.

As mentioned earlier, the profit maximization problem that we formulated in section 3.5.3.2 is an approximation. Since we split the range $[U_{ij,lopt}, U_{ij,max}]$ into K levels, there is a quantization error. As we increase the value of K this error reduces and results improve. In order to understand the impact of K on approximation result we obtain the profit maximization results using different values of K for *config_A*. Fig. 13 shows the result with profit normalized to K=1. Across all three workloads, beyond K=4 the improvement in the solution is below 0.1%. However, as K increases, NFRA execution time increases super-linearly as shown on the secondary Y-axis, thereby increasing the overhead. Note that the improvement over different K values is also a function of server pool and client characterization. Hence the data shown in Fig. 13 may not hold across different server pool and/or client configurations, but the proposed iterative edge splitting approach will be able to quickly converge to a good solution given appropriate threshold bounds on incremental profit.

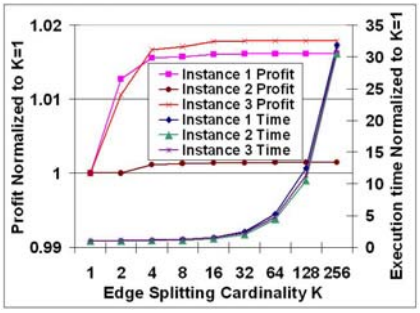

Fig. 13 Impact of Edge Splitting Cardinality on Approximation.

To understand how NFRA runtime scales across much larger setups we solved profit maximizing NFRA across four different setups, characterized by parameters (# of pools, # of clients, # of servers across all pools) given as follows: (12, 24, 600), (24, 40, 1200), (36, 56, 1800) and (48, 72, 2400). These setups were generated by scaling *config_A* and *Instance_1* workloads. In order to scale up the number of server pools we took the existing baseline *config_A* setup and generated new server pools by scaling up each of the server pool in terms of their frequency and power values. Furthermore *Instance_1* workload was similarly scaled up to generate T_{ij} and e_{ij} values for each of the new server pool by applying appropriate scaling factors. The details of these scaling factors are omitted here in the interest of space. However for the purpose of runtime scaling what matters is the number variables each new hosting center setup introduces rather than the precise values of T_{ij} and e_{ij}. In Fig. 14 we show how runtime scales on our Xeon dual core 1.86GHz, 2GB RAM machine. As shown in the figure, even with larger setup, run time for K=4 for the largest setup is about 2 seconds

while the profit is within 0.1% compared to K=16. However, with K=16 run time increases substantially (3X).

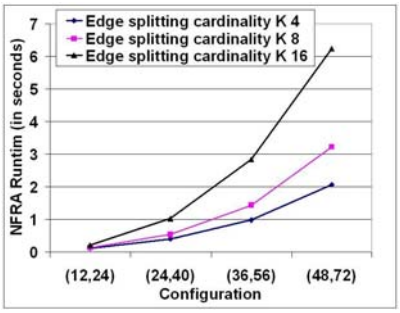

Fig. 14 NFRA Runtime Scaling.

4.6 Sensitivity Analysis

In this section we present the impact of variation, in various parameters, on the optimality of the solution and on the SLA constraints. For this purpose we perform sensitivity analysis, for profit maximizing NFRA, by introducing percentage variation in the arrival rates (λ_j) of all the clients. We introduce variation homogenously across all the clients for simplicity since otherwise exploration space will explode. Our sensitivity analysis was carried out for profit maximizing NFRA for *Instance_3* under *config_A*. First we obtain the optimal resource allocation, *opt*, without considering any variation with corresponding optimal profit denoted as $\boldsymbol{profit}_{opt}$. Then we apply different amounts of variation in arrival rate as show on X-axis in Fig. 15.

Due to this variation, fraction of the requests, denote by β_{new}, that will violate response time will be different from the fraction resulting from the optimal solution which has ignored variation. The results on primary Y-axis of Fig. 15 show that how far β_{new} is from the β stipulated in SLA. We denote this distance by percentage difference $\Delta\beta$. Furthermore we obtain new optimal resource allocation, opt_{new}, considering new arrival rates resulting from variation with corresponding optimal profit denoted as $\boldsymbol{profit}_{newopt}$. Note that the achieved profit by *opt*, as a result of variation, denoted by $\boldsymbol{profit'}_{opt}$, will be different from $\boldsymbol{profit}_{opt}$. The secondary Y-axis of Fig. 15 show how far $\boldsymbol{profit'}_{opt}$ is from $\boldsymbol{profit}_{newopt}$. Obviously, if the average arrival rate is less than the originally estimated λ_j value, $\Delta\beta$ is negative, i.e., we satisfy more requests than stipulated by SLA on time. Furthermore in the worst case $\boldsymbol{profit'}_{opt}$ is 10.6% lower than $\boldsymbol{profit}_{newopt}$ with average being 5.1%. As the average arrival rate increases, particularly beyond 4% variation, $\Delta\beta$ goes above zero for many clients, leading to SLA violation. For example, for client 1 at 6% variation we are violating β=0.05 (cf. Table 1) by 21%; probabilistically, 6.05% of the requests, instead of 5%, will violate the $\tau_{j,max}$. This implies that either we must make allowance for the expected variation during resource allocation for a given time epoch or based on dynamic monitoring of $\Delta\beta$ solve a new instance of NFRA instead of waiting till the end of the time epoch. Here time epoch refers to time interval length at which resource allocation decision are made. We carried out similar analysis for variation in μ_{ij} and e_{ij} obtaining similar results which are not presented here in the interest of space. Note that solving the new instance of NFRA may result in a resource allocation that demands client migration from previously allocated servers to new servers. Such migration if done frequently is undesirable. We can address this issue by comparing the resource allocation from the previous epoch with the new instance of NFRA. If the new solution needs substantial migration for a given client whose costs are prohibitive then we connect the client only to servers previously unallocated to this client and solve a new instance of NFRA with only the increment in client's request arrival rate.

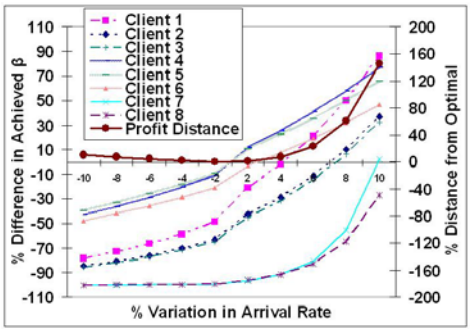

Fig. 15 Arrival Rate Variation.

If variations in the arrival rate are frequent and have large amplitude then we must solve a new instance of NFRA in order to stay close to the optimal solution, which may require turning ON or OFF more servers. Here we focus on increase in the arrival rates, which may require turning ON some new servers, either by 1) Reboot: unused servers are originally turned off and hence require reboot if allocated or 2) Wakeup: unused servers are put to system sleep state S3 and hence require wakeup when allocated. Sleeping servers still consume power, but much less than ON server, such as for DRAM refresh; OFF servers consume zero power for all practical purposes. To understand the impact of the frequency and amplitude of variation, in λ_j, on profit and penalties associated with turning servers ON, we carry out the following experiment: In order to capture frequency of variation we vary the length of the time epoch from 10 minutes to 60 minutes in the increments of 10 minutes, shown on X-axis in Fig. 16. At each such time epoch we apply 10, 15 and 20% increase in the arrival rates to capture the amplitude of variation and solve NFRA.

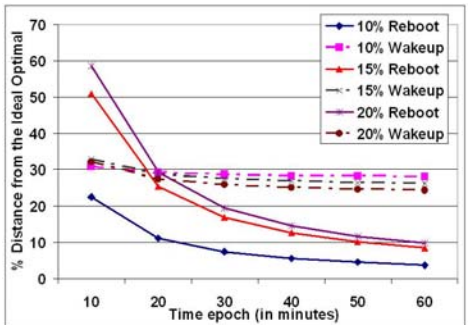

Fig. 16 Impact of Reboot/Wakeup Latency and Power.

The resource reallocation may require few more servers to be turned ON. The latencies associated with rebooting

the servers and wakeup from S3 are conservatively set to 90 [3] and 15 [23] seconds, respectively. During this reboot/wakeup latency, if the increase in arrival rate is such that the previously allocated resources are under provisioned then servers may run at 100% utilization. Requests allocated to such servers will be queued until served but will violate the response time. Hence we conservatively assume that we will pay penalty on all such requests. Once the servers are UP, either after reboot or wakeup, we start operating at the optimal point. We consider S3 state power to be 1/20th of the idle power [27]. The results in Fig. 16 show, for different time epochs lengths, how far the achieved profit will be from the theoretical optimum. The theoretical optimum here assumes that servers are turned ON instantly.

Note that at smaller time epoch lengths, e.g., 10 minutes, the latency associated with reboot is very high compared to wakeup from S3, hence we incur high losses under Reboot at high variations, i.e., 15% and 20%. Therefore under these circumstances Wakeup is better, as shown in Fig. 16. Although at low variation, 10%, Reboot works better since losses incurred are small during the latency period and for the rest of the time epoch unused servers consume no power. At larger epoch lengths Reboot works better since servers keep consuming power for the rest of the time epoch under S3 state for Wakeup. Hence if frequent and high variations are observed, Wakeup works better and Reboot works better otherwise.

5. Related Work

Due to increasing energy and cooling cost of the hosting centers many of the previous works have focused on energy management in hosting centers. Chase et al. in [1] proposed an economy based approach for energy optimization that monitors the dynamic variation in the workload and accordingly assigns servers so as to minimize energy while meeting SLA. In [2] authors proposed various independent and coordinated workload dependent DVFS techniques to minimize energy in server clusters. In [3] authors proposed queuing and control theory based performance constrained energy management framework for homogenous server clusters. Rusu, et al. in [4] proposed a QoS aware technique that dynamically reconfigures a set of heterogeneous clusters to reduce energy during the reduced load period. In [5] authors proposed power aware request distribution technique taking into account the startup and shutdown delay of the servers. Authors in [6] proposed an approach that exploits long-time-scale variation in the workload in order to reduce the resource requirement for energy improvement. In [7] Heath, et al. proposed a heterogeneous server cluster design for throughput constrained energy optimization. In a more recent work, Raghavendra et al. in [8] proposed a coordinated energy management approach by integrating various independent policies working at different levels of the hierarchy in the datacenter. Their framework attempts to synchronize decisions made at various levels. Authors in [9] proposed a power minimizing datacenter architecture by employing comprehensive online monitoring, live virtual machine (VM) migration and VM placement optimization. In [10] Govindan et al. proposed power management approach by under provisioning the resources and overbooking the power needs of the hosted workloads and distributing the power flexibly among hosted workloads. In another set of research ([15], [16], [17]) authors proposed energy management techniques for disk power management in servers with multispeed disks, dynamic speed control and storage cache management.

Along with the energy management in the hosting center profit optimization has also been a concern that has been addressed by many previous works. In [13] authors proposed to profile applications in terms of their resource needs and then employ resource overbooking to maximize the generated revenue while providing performance guarantees. Bennani [14] et al. proposed an analytical queuing model and combinatorial search based approach to resource allocation that optimizes some utility function, e.g throughput or response time etc. In [11] authors proposed an approach to maximize the total profit considering the prices and penalty paid by the clients being hosted on the hosting center, subject to SLAs. They used the fixed point iteration method to converge to optimal solution by solving separable concave resource allocation problems assuming an increasing, concave and differentiable price function. In another profit maximizing approach, Zhang et al. [12] proposed tabu search based heuristics that tries to maximize the total profit while considering the operational cost of the servers.

6. Conclusions

In this paper we proposed a generalized network flow based resource allocation framework called NFRA. We showed how non energy proportional systems affect the profits made by quantifying non energy proportionality of the systems with Energy Increase Factor (EIF). Using the proposed NFRA framework then, we solve resource allocation problems for throughput constraint, average response time constraint and stochastic maximum response time constraint with the objectives of energy minimization and profit maximization. We show that, while, given sufficient resources, heuristic based approaches may fail to find a feasible solution; NFRA finds a feasible solution. Furthermore using NFRA for profit maximization we show that our approach quickly converges towards optimal and finds a solution that is within 1% bound of the optimal.

References

[1] J. S. Chase, et al., "Managing energy and server resources in hosting centers", In Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, October, 2001, 103-116.

[2] M. Elnozahy, et al., "Energy-Efficient Server Clusters", In Proceedings of the Second Workshop on Power Aware Computing Systems, February 2002, 179-197.

[3] Y. Chen, et al., "Managing server energy and operational costs in hosting centers", In Proceedings of the ACM SIGMETRICS international Conference on Measurement and Modeling of Computer Systems, June 2005, 303 – 314.

[4] C. Rusu, et al., “Energy-Efficient Real-Time Heterogeneous Server Clusters”, In Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium, April, 2006, 418 – 428.
[5] K. Rajamani, et al., “On evaluating request-distribution schemes for saving energy in server clusters”, In Proceedings of the 2003 IEEE international Symposium on Performance Analysis of Systems and Software, March 2003, 111 – 122.
[6] S. Ranjan, et al., “QoS-driven server migration for Internet data centers,” Tenth IEEE International Workshop on Quality of Service, 2002.
[7] T. Heath, et al., “Energy conservation in heterogeneous server clusters”, In Proceedings of the Tenth Symposium on Principles and Practice of Parallel Programming, June 2005, 186 – 195.
[8] R. Raghavendra, et al., “No "power" struggles: coordinated multi-level power management for the data center”, SIGARCH Comput. Archit. News 36, March 2008, 48-59.
[9] L. Liu, et al., “GreenCloud: a new architecture for green data center”, In Proceedings of the 6th international Conference industry Session on Autonomic Computing and Communications industry session, June 2009, 29-38.
[10] S. Govindan, et al., “Statistical profiling-based techniques for effective power provisioning in data centers”, In Proceedings of the 4th European Conference on Computer Systems, April 2009, 317-330.
[11] Z. Liu, et al., “On maximizing service-level-agreement profits”, In Proceedings of the 3rd ACM Conference on Electronic Commerce, October, 2001, 213 – 223.
[12] L. Zhang, et al., “SLA based profit optimization in autonomic computing systems”, In Proceedings of the 2nd international Conference on Service Oriented Computing November, 2004, 173 – 182.
[13] B. Urgaonkar, et al., “Resource overbooking and application profiling in shared hosting platforms”, SIGOPS Oper. Syst. Rev. 36, Dec. 2002, 239 – 254.
[14] M. N. Bennani, et al., “Resource Allocation for Autonomic Data Centers using Analytic Performance Models”, In Proceedings of the Second international Conference on Automatic Computing, June 2005, 229-240.
[15] E. V. Carrera, et al., “Conserving disk energy in network servers”, In Proceedings of ICS, June 2003, 86 - 97.
[16] S. Gurumurthi, et al., “DRPM: dynamic speed control for power management in server class disks”, SIGARCH Comput. Archit. News 31, May 2003, 169 – 181.
[17] Q. Zhu, et al., “Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management”, In Proceedings of the 10th international Symposium on High Performance Computer Architecture, February 2004, 118 – 118.
[18] K. D. Wayne, “Generalized Maximum Flow Algorithms”, PhD thesis, Department of Operations Research and Industrial Engineering, Cornell University, 1999.
[19] P.M. Vaidya, “Speeding up linear programming using fast matrix multiplication”, Technical Memorandum, AT&T Bell Laboratories, Murray Hill, NJ, 1989, 332-337.
[20] G. Bolch, et al., Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications. John Wiley, New York, 1998.
[21] L. A. Barroso, et al., “The Case for Energy-Proportional Computing”, Computer 40, 12 (Dec. 2007), 33-37.
[22] Verma, A.et al., “On admission control for profit maximization of networked service providers” In Proceedings of the 12th international Conference on World Wide Web. WWW '03. ACM, 128 - 137.
[23] Horvath, T. et al.,“Multi-mode energy management for multi-tier server clusters”, In Proceedings of the 17th international Conference on Parallel Architectures and Compilation Techniques, PACT '08, 270-279.
[24] SPECWeb2009: http://www.spec.org/web2009
[25] http://www.energystar.gov/index.cfm?c=prod_development.server efficiency#epa. Report to congress on server and data center energy efficiency. Retreived Oct 2009.
[26] lpsolve: http://lpsolve.sourceforge.net/
[27] http://terranovum.com/projects/energystar/standby_v_hiber.html
[28] http://h20000.www2.hp.com/bc/docs/support/SupportManual/c03161908/c03161908.pdf