Dynamic CPU Scheduling for QoS Provisioning

Kostas Katsalis
University of Thessaly, Greece
Email: kkatsalis@uth.gr

Georgios S. Paschos
LIDS Lab
MIT MA, USA
Email: gpasxos@mit.edu

Leandros Tassiulas
CERTH-ITI
University of Thessaly, Greece
Email: leandros@uth.gr

Yiannis Viniotis
Department of ECE
NCSU, USA
Email: candice@ncsu.edu

***Abstract*—Distributed, large-scale, enterprise applications are commonly supported in multi-tier data-center environments. In this paper, we study a scheduling problem for sharing CPU time in a cluster of servers among a number of enterprise customers. Such sharing is typically mandated by service differentiation requirements and QoS guarantees. Our main contribution is the formal definition of a CPU allocation/scheduling problem with respect to QoS quarantees and evaluation of scheduling policies that address the following design criteria: they have provable performance, they do not require a priori knowledge of service statistics and their overhead is adjustable. We provide the necessary mathematical framework for policies that satisfy the above criteria and evaluate proposed algorithms via theoretical analysis and extensive simulations.**

Keywords*—*Appliances, Application Delivery Control, Service Differentiation, Multitier Data Centers, Scheduling

I. Introduction

A typical, enterprise data center model is based on the *multi-tier architecture*[11] shown in Figure 1. The switch fabric consists of core, aggregation and access tier: core switches provide the interface for the flows entering or leaving the data center; aggregation switches refer to the aggregation of traffic to and from the access switches; and, access switches is the point of server attachment to the data center switch fabric. In addition, servers host applications are also "tiered". In Figure 1 we depict two such tiers, labelled as service and server tiers. Servers in the latter tier execute the main application processing; servers in the former tier are typically deployed to perform specialized preprocessing of user requests, such as SSL offloading, load balancing, firewalling, etc.

In this work, we study a resource provisioning problem that arises when a cluster of servers is accessed by a number of different *Service Domains* (SDs), i.e. classes of customers competing for CPU resources. Such customers are often large enterprises that initiate a large volume of requests for web service processing, authentication services or protocol bridging and integration functions. In a typical scenario, these large customers negotiate Service Level Agreements (SLAs) with the service providers. We focus on SLAs in which processing time in the service tier is provisioned/guaranteed to the service domains in situations of overload. Usually, in such systems overload periods are defined by congestion on the queueing services during peak usage intervals.

The designer of the data center can control incoming traffic in a variety of ways. In the context of the multitier data center architecture, we assume that a controller is placed in the switch fabric; the controller places the incoming requests into queues organized by service domain. It then decides how to schedule

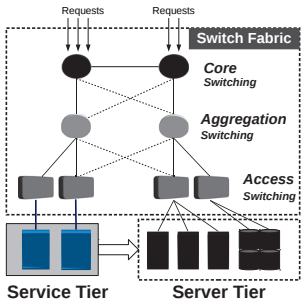

Fig. 1. Multitier enterprise data center architecture.

the requests from these queues to the servers residing in the service tier. This decision is based on a desired allocation of the CPU time to the service domains, as we will see in detail in the next section.

Our contributions are the following: First, we provide a mathematical formulation of the CPU provisioning problem. Besides services provided in data centers, the formulation is applicable to a broad family of scheduling problems where differentiation of resources must be provided, like for example in OS thread/process scheduling or multi-processing systems. Then, we formally define and prove the necessary conditions that will form a class of policies that are oblivious to service time statistics and fulfil the design criteria. We propose two policies and evaluate them based on their convergence speed, the required overhead and their insensitivity to system and statistical assumptions.

The paper is organized as follows: in Section II, we provide the motivation for this work, in Section III we describe the system model and the objectives, in Section IV, we define a class of scheduling policies that meets the objectives and outline two policies of this class. In Section V, we evaluate the performance of the proposed policies. We summarize previous related work in Section VI and conclude our study in Section VII.

II. Motivation and proof of concept

The SLA we consider stems directly from the IT industry. It is common practice in such environments that the provider offers contracts with *different* charges/levels/types of service, based on whether the system is under "normal" or "overload" conditions. What constitutes normal and overload conditions depends on the service provider.

In typical contracts, metrics used for SLAs governing operations during normal load involve response times, availability, throughput, losses, etc. SLAs for overload conditions involve

978-3-901882-50-0

fewer and/or "simpler" metrics; the main idea is that under load stress, the system cannot effectively guarantee the same level of performance as the normal load metrics describe. The most widely used metric for overload conditions is CPU utilization; a typical SLA is: *"(during overload conditions) guarantee that service domain i gets a predefined percentile p_i of the appliance CPU capacity"*.

III. Problem Statement

In the context of the data center design shown in Figure 1, we are interested in allocating CPU time in all the servers of the service tier to traffic that arrives through the switch fabric. We formalize this problem, using an abstract system model.

A. System model

The system model is depicted in Figure 2 and consists of one controller, a set $\mathcal{M} = \{1, \ldots, m\}$ of servers, and a set $\mathcal{D} = \{1, \ldots, d\}$ of *service domains* (customer classes). [1] The controller maintains a queue for each domain that accommodates incoming requests in a First-Come-First-Serve (FCFS) manner. Since we focus on overload conditions, we assume saturated arrival conditions, i.e., each queue has always at least one request waiting to be serviced.

The controller obtains feedback signals from the servers at time instances when service is completed, selects one domain and forwards a request from this domain to the server that just finished servicing. Note that the controller cannot make selections at any other times, since preemption at servers is not allowed. The selection is done with the objective of satisfying the SLA we describe in Section III-C. In addition, we make the following assumptions for modelling simplicity and we discuss the impact of these assumptions in the next subsection: we assume that the feedback signals from any server to the controller is instantaneous; moreover, the transmission of a request from a controller queue to a server also takes zero time. The service times are generally distributed, with the same probability distribution and with finite mean for all the domains, where the value of the mean is *unknown*.

B. Justification of the chosen system and control model

The queues and the controller are located outside the service tier (i.e., in the core router): by keeping queues at the router side and making decisions at service completion instants, one makes sure that the control action is taken with the full system state known. If instead, we queued requests in the servers, the routing decisions would be made much earlier than the service instant and latest information about the aggregate CPU times could not be used. The instantaneous feedback and request transfer assumption would hold true in a practical deployment of our system, if a) a small, ping-pong buffering mechanism is used in the server and b) the server-controller interconnection is fast, compared to CPU processing times. Such a buffer would hold one to two requests at a time, ensuring that the server does not go idle while the controller decides the next allocation.

[1] In typical data center implementations, the controller can be housed in any device that terminates TCP/UDP connections (e.g., a core router, an http router/sprayer).

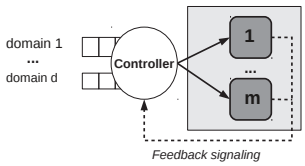

Fig. 2. System model.

C. The formal SLA statement (T-SLA)

The service differentiation problem we consider can be formally expressed by means of a Technical SLA (*T-SLA*): Let $i = 1, \ldots, d$, be the number of service domains utilizing the CPUs in the m servers of the service tier; let p_i be given strictly positive constants that sum up to 1. Allocate a percent p_i of the CPU capacity in the service tier to service domain i, using an appropriate scheduling mechanism at the controller.

D. Definitions for the mathematical statement

Let $\mathbb{N}_+, \mathbb{R}_+$ denote the sets of nonnegative integer and real numbers respectively. Consider the time instants $t_k, k \in \mathbb{N}_+$ at which a server becomes empty and can serve another request. Formally, we define the *action space* $\mathcal{Z} = \{1, 2, \ldots, d\}$ as the set of all domains. We say that the controller takes *action* $a(t_k) = i \in \mathcal{Z}$ at time t_k if it selects a request from domain i to forward to the server that became available at time t_k. A *scheduling policy* π (or policy π for simplicity) is a rule that determines the control actions across time.

Definition 1 (Total service of domain i): Define the *total service of domain* i, $F_i^\pi(t) : \mathbb{R}_+ \longmapsto \mathbb{R}_+$, as the aggregate CPU time that was allocated to domain i until time t under policy π. Note that $F_i^\pi(t)$ is a summation of service time random variables.

Definition 2 (Utilization of domain i): The *utilization of domain* i under policy π up to time t, $U_i^\pi(t)$, is defined as

$$U_i^\pi(t) \doteq \frac{F_i^\pi(t)}{mt}, \quad \forall i \in \mathcal{D}, \quad t \in \mathbb{R}_+. \tag{1}$$

First, note that $\sum_{i \in \mathcal{D}} U_i^\pi(t) \leq 1$ for all t, π since $\sum_{i \in \mathcal{D}} F_i^\pi(t)$, the total service obtained by the domains, must be smaller or equal to the total time that the servers have been operating which is equal to mt. Moreover, $\sum_{i \in \mathcal{D}} U_i^\pi(t) \leq 1$ holds with equality if all servers CPU have no idle periods.

Definition 3 (Non-idling policies): A policy that satisfies

$$\sum_{i \in \mathcal{D}} U_i^\pi(t) = 1, \qquad \text{for all } t$$

is called a non-idling policy.

We define further the (long-term, steady-state, infinite horizon) metric of the allotted CPU to each domain i.

Definition 4 (Allotted CPU to domain i): The allotted percent of CPU capacity to domain i, under policy π is

$$\widetilde{U_i}(\pi) \doteq \lim_{t \to \infty} U_i^\pi(t) \tag{2}$$

$\widetilde{U_i}(\pi)$ is a long-term performance criterion achieved by policy π. We use the limit instead of $\liminf$ since we examine policies that we know *a priori* that will achieve steady state.

Formally, the objective of the (*T-SLA*) can be stated as follows: *Design a scheduling policy π such that for given percentiles p_i it achieves (in the long-run)*

$$\widetilde{U_i}(\pi) = \lim_{t\to\infty} U_i^{\pi}(t) = p_i, \qquad i = 1, 2, \ldots, d. \quad (3)$$

In the following we define and present a class of policies that can meet this objecitve.

IV. Proposed Policies

A "desirable" scheduling policy has several properties: (a) it achieves the *T-SLA*, (b) converges to the *T-SLA* fast, (c) is agnostic to the service statistics and, (d) requires a small number of calculations per time unit. Apart from (a) which is obvious, convergence (b) is crucial for achieving the goal within short periods. The knowledge of service statistics (c) and the decision load (d) are both related to the communication overhead and CPU costs and are very important considerations for practical systems.

To the best of our knowledge, no single scheduling policy exists that is superior in all these properties; our approach in this paper is to investigate policies that excel in some of the above criteria and give the designer the ability to trade off in order to satisfy the rest. In particular, we define a class of policies which have properties (a) and (c) and are able to trade off for (b) and (d).

A. The need for a dynamic policy

A (plain) Round Robin (RR) policy utilizes the idea of the *round* during which each service domain is served once. Let $m = 1$ the number of servers and the random variable $L_n = \sum_{k=1}^{D} S_k(n)$ be the duration of round n, where $S_k(n)$ is the service received by domain k during that period. By the Strong Law of Large Numbers (SLLN), the summations of service times for a given service domain will converge to the mean:

$$U_i^{\mathrm{RR}} = \lim_{m\to\infty} \frac{\sum_{n=1}^{m} S_i(n)}{\sum_{n=1}^{m} L_n} = \frac{\mathrm{E}[S_i(n)]}{\sum_{k=1}^{D} \mathrm{E}[S_k(n)]}$$

Since the targets p_i of *T-SLA* are arbitrary, in general the RR policy cannot satisfy the objective in eq (3). By similar arguments, the same conclusion holds true for static probabilistic policies and Weighted Round Robin policies in which weights are assigned statically. These schemes can only reach arbitrary defined goals in the case when the service process is known in advance. This gives rise to dynamic scheduling policies that act agnostically to service rates and adapt to changing conditions.

We focus on dynamic RR policies where the idea of a round is used again, but the number of times each domain is served is controlled in a dynamic way; round by round a decision is made as follows: 1)At the beginning of a new round a list of domains, accompanied by weights w_i for every domain i, is selected by the controller.2) All the domains in the list are scheduled for server CPU service that many times as the weight indicates. 3) When the round is over, a new list as well as new weights are calculated.

B. Class Π of dynamic policies achieving the SLA

In what follows, we define $t_n \in \mathbb{R}_+$, to be the time instance when round n begins, where $n \in \mathbb{N}_+$. We will use index i for the set of service domains $\mathcal{D} \doteq \{1, \ldots, d\}$ and index j for the set of servers $\mathcal{M} \doteq \{1, \ldots, m\}$. Recall that S_i is a random variable describing the service time of domain i, with $\mathrm{E}[S_i] < \infty$. At each round n, $w_i(t_n)$ requests of domain i are serviced, with the number selected by the scheduling policy. The length of round n is denoted by L_n and given by

$$L_n = \sum_{i\in\mathcal{D}} \sum_{k=1}^{w_i(t_n)} S_i(k),$$

where $S_i(k)$ are independent random variables identically distributed with S_i. For all the non-idling policies operating in rounds it holds that at least one service domain i with $w_i(t_n) > 0$ participates in each round and thus for any round n we can write: $L_n \geq \min_i S_i \quad a.s.$ Next, we further constraint the set of policies:

Definition 5: A bounded-round policy satisfies

$$\mathrm{P}(L_n < \infty) = 1.$$

Our first analysis result states that bounded-round policies achieve steady state.

Theorem 1 (Convergence): The limit of $U_i(t)$ as $t \to \infty$ of a bounded-round policy π exists almost surely.

The proof of this theorem is based in the following recursive form, where $Z_i^{\pi}(n)$ is the total service that domain i received within the n^{th} round[2] :

$$U_i^{\pi}(t_{n+1}) = \frac{t_n}{t_{n+1}} U_i^{\pi}(t_n) + \frac{Z_i^{\pi}(n)}{t_{n+1}}, \quad (4)$$

Consider now the following set of policies:

Definition 6 (Fair policies): A policy that assigns $w_i(t) = 0$ whenever $U_i^{\pi}(t_{n+1}) > p_i$ is called fair.

First note that a fair policy is a non-idling policy for the case of saturated queues. We prove this by contradiction. Assume a fair policy that is also an idling policy. Assuming $\sum_{i\in\mathcal{D}} U_i^{\pi}(t) < 1$ for some t, there exists a time period $[t_0, t_0 + \delta], \delta > 0$, which contains an empty round starting at t_k, i.e., $w_i(t_k) = 0, \quad \forall i \in \mathcal{D}, t_k \in [t_0, t_0 + \delta]$. This implies that $U_i^{\pi}(t_k) > p_i, \quad \forall i \in \mathcal{D}$. Summing up, we get

$$\sum_{i\in\mathcal{D}} U_i^{\pi}(t_k) > \sum_{i\in\mathcal{D}} p_i = 1,$$

which is a contradiction. Thus, let us consider the set of policies Π, which contains all bounded-round, fair policies. As we have shown, if $\pi \in \Pi$, then π is also non-idling.

Theorem 2 (Optimality of Π): The bounded-round, fair policies achieve the T-SLA.

Proof: Consider any bounded-round fair policy $\pi \in \Pi$. From Theorem 1 we conclude that the limit $\lim_{t\to\infty} U_i^{\pi}(t)$ exists. Assume that for some domain i we have

$$\lim_{n\to\infty} U_i^{\pi}(t_n) = \tilde{u}_i > p_i$$

[2]Due to page limitations, the proof is omitted from this version of this work.

and $p_i > 0$. Then pick $\epsilon < \tilde{u}_i - p_i$, the limit guarantees that there exists $n_0(\epsilon)$ such that for all $n > n_0$ we will have

$$U_i^{\pi}(t_n) > \tilde{u}_i - \epsilon > p_i$$

and thus, $w_i(t_n) = 0$ for all $n > n_0$. The latter, however, implies that $\lim_{n\to\infty} U_i^{\pi}(t_n) = 0$ with probability 1 which is a contradiction. Thus, $\lim_{n\to\infty} U_i^{\pi}(t) \leq p_i$.

Then, partition the set of domains $\mathcal{D}$ into the subsets $\mathcal{D}_l$, $\mathcal{D}_h$ such that:

$$\begin{cases} \tilde{u}_i < p_i & \text{if } i \in \mathcal{D}_l \\ \tilde{u}_i = p_i & \text{if } i \in \mathcal{D}_h \end{cases}$$

Since we have shown that it is impossible to have $\tilde{u}_i > p_i$ and all limits exist, the above partition is valid. Assume then that the set $\mathcal{D}_l$ is non-empty, summing over the union we have:

$$\sum_{i\in\mathcal{D}_l\cup\mathcal{D}_h} \tilde{u}_i < \sum_{i\in\mathcal{D}_l\cup\mathcal{D}_h} p_i = 1,$$

which is a contradiction since $\sum_i U_i(t, BGP) = 1$ for all t. From the above discussion, we conclude that for all bounded-round fair policies

$$\lim_{n\to\infty} U_i^{\pi}(t_n) = p_i, \qquad \forall i \in \mathcal{D}.$$

■

In the following, we examine two policies that belong to the class of bounded-round fair policies defined above.

1) Below Goal Participates (BGP): Under the BGP policy, in round n (that starts at time t_n) all the service domains with utilization $U_i^{BGP}(t_n) \leq p_i$ have weights with $w_i(t_n) = 1$; the rest have zero weights. In other words, only the service domains that have allotted CPU time less than or equal to their *T-SLA* goal are given one slot; the rest are given none.

To face overhead limitations, $T - BGP$, a variation of the policy of BGP, is proposed. To avoid communication overhead we keep the calculated weights constant for a fixed period of time T. More formally, for an integer number of k rounds, weights remain the same, as long as $t_{n+k} < t_n + T$, meaning $w_i^n = w_i^{n+1} = ... = w_i^{n+k}$ for any domain i. In the time instant when for the minimum k, $t_{n+k} \geq t_n + T$ is true, k is reset to zero and new weights are calculated for the new round according to BGP policy.

2) Only Most Suffering (OMS): Under the OMS policy, each round is composed of only one domain which is served once. The selected domain is the one with the largest amount of missing service. More precisely, at a decision instant t_n, we set $w_i(t_n) = 0, i \neq j$ and $w_j(t_n) = 1$ where $j = \arg\min_{k\in\mathcal{D}}\{U_k^{\text{OMS}}((t_n)) - p_k\}$. As a policy, OMS resembles the family of *maximum weight* policies [2] and the Join the Shortest Queue policy; both are well-known optimal policies and can be thought of as a degenerate case of the Round Robin scheduling policies. Note, however, that OMS is a very dynamic policy and introduces a high communication and computation overhead; it must calculate a decision at every service completion instant. Similar to *T-BGP*, we define T-OMS, the $T - delayed$ version of the OMS policy, in order to provide a trade-off between overhead and performance.

Note that all four policies are clearly fair policies and moreover they are bounded-round policies since we have (almost surely):

$$L_n^{BGP} \leq \sum_{i\in\mathcal{D}} S_i, L_n^{T-BGP} \leq T + \sum_{i\in\mathcal{D}} S_i,$$
$$L_n^{OMS} \leq \max_{i\in\mathcal{D}} S_i, L_n^{T-OMS} \leq T + \max_{i\in\mathcal{D}} S_i.$$

Corollary 1: $BGP, T-BGP, OMS, T-OMS$ achieve the T-SLA.

V. Policy Evaluation

Besides the theoretical analysis of the previous section, we evaluate our policies based on extensive simulations, studying practical considerations and overhead issues. More specifically a) we demonstrate how statistical and system parameters affect convergence speed, and b) we examine trade offs that aim to reduce communication and processing overhead. All simulations were performed in a custom, discrete-event simulator written in Java. The main simulation model is a controller used to spread traffic from a set of d service domains to a cluster of m servers. Our investigation is based on extensive simulations that were performed but due to lack of space we present an indicative set.

A. Effect of system and statistical parameters on convergence

In the first series of simulations, we present results for the BGP policy. The effect of system and statistical parameters is the same for OMS. Also, note that for all the simulations presented, time refers to simulation time units and is a dimensionless quantity. In that sense, time units can be considered as seconds.

1) Statistical parameters effects: The statistical parameters we studied concern the service process in the servers. We performed extensive simulations in studying how the rate of convergence depends on (a) the service rates, (b) the service distribution variations, and, (c) the service probability distribution itself. For the simulation scenario presented, $d = 4$ and p_i objectives are defined as $\vec{p} = (10\%, 20\%, 30\%, 40\%)$.

The performance of the algorithm for all the domains can be seen in Figure 3-a , where for all the domains $\mu = 1$. As we can see, the algorithm achieves the desired utilization.

In Figure 3-b, we present results where all the domains have the same service rate, ranging from 0.1 to 100. In the case where all the service domains have the same service time, increasing service rates increases the convergence rate; more rounds are carried out in the same time and convergence occurs faster.

Figure 3-c is indicative of how the variance between the service rates for any domain i affects performance. We compare cases where the service rate distribution "helps" the domain which needs more service, and cases where the service rate distribution "helps" the service which requests less CPU power. When for domains i, j, $\mu_i > \mu_j$ then the service time is less per request for domain i and if the target is high, domain must participate in more rounds than in the case when the target is low. In order to control the variance of the service time distribution for every domain i we use the formula

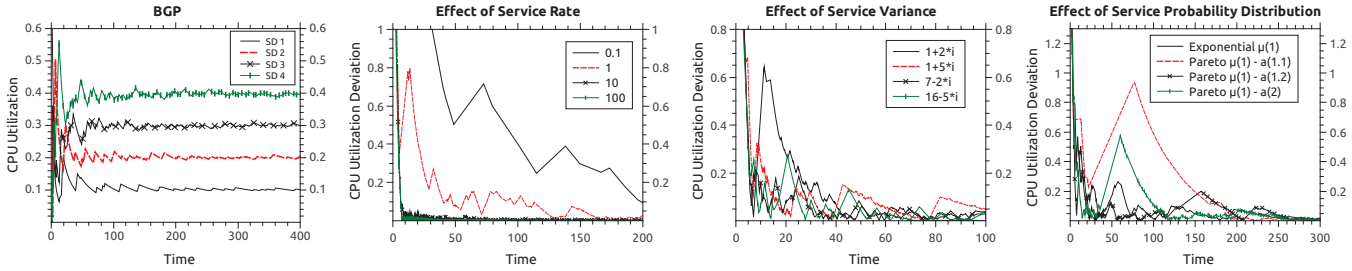

Fig. 3. Effect of statistical parameters. In the vertical axis of (b),(c) and (d), $\sum_{i}^{d} |U_i^{BGP}(t) - p_i|$, the total deviation from the goal vector is noted.

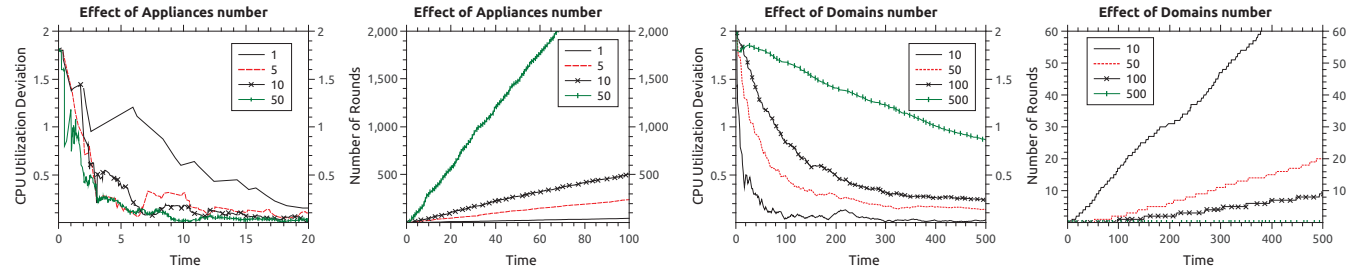

Fig. 4. Effect of system parameters. In the vertical axis of (a) and (c), $\sum_{i}^{d} |U_i^{BGP}(t) - p_i|$, the total deviation from the goal vector is noted.

$\mu_i = a + b * i$, where the values of a and b can be seen in the legend of the Figure 3-c.

Using a service probability distribution different from the exponential (with the same average service time for all the domains), results in a different convergence rate as we can see in Figure 3-d. For the case of service time following Exponential and Pareto distributions, the mean for all the domains is equal to 1 and the shape parameter a of the Pareto distribution is varied, to achieve different "tail" behavior. The closer a is to 1, the heavier the tail, generating very long service times. These appear as spikes in the figure and make convergence harder.

2) System parameter effects: We study the impact of the number of servers m and the number of service domains d on the convergence rate. In Figure 4-a, we present simulations for an environment where $d = 4$ and the number of servers varies from 1 to 50. In the vertical axis the total deviation from the goal vector is noted while we set the *T-SLA* to be $\vec{p} = (10\%, 20\%, 30\%, 40\%)$ and all the domains have exponential service times with $\mu_i = 1$. Increasing the number of servers leads to faster convergence and this effect can be explained by the behavior shown in Figure 4-b: for any given time, BGP completes more rounds when there are more servers in the system and it has more chances of adjusting. The number of domains also has a direct impact in the form and rate of convergence, as we can see in Figure 4-c where total deviation from the goal vector is noted in vertical axis; increasing the number of domains results in slower convergence. While increasing the number of domains a round takes more time to complete, less rounds are executed in the same period of time (Figure 4-d) and thus the policy exhibits slower convergence rates. The T-SLA for these simulations specified $p_1 = 10\%$ for one domain and equal p_i values for the rest.

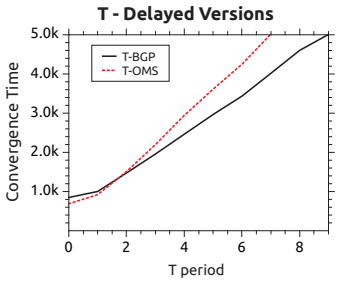

Fig. 5. T- Delayed versions.

B. Overhead and trading-off analysis

As we already discussed in subsection IV-B, one way to reduce communication and processing overhead is to keep the calculated weights constant for a fixed period of time before the service pattern is updated. The *T-Delayed* versions also converge but now this happens with a slower pace than before. We note that the one domain per round strategy of $T - OMS$ is rather penalized in this scenario; $T - BGP$ utilizes the idea of the round and appears superior. This can be seen in Figure 5, where a comparison of convergence speed is presented between the two policies. The simulation scenario is $d = 4$, target utilizations are $\vec{p} = (10\%, 20\%, 30\%, 40\%)$ and

$E[S_i] = 1s$, the same for all the domains. The convergence criterion was $U_i^{\pi}(t) = p_i \pm 0.05$ for all domains i for at least 200 successive rounds. As we can observe, only for small values of T, OMS is slightly superior than BGP. Note, that there exists a value of period $\widetilde{T}$ where $T-OMS$ and $T-BGP$ experience similar convergence time and above this value $T - BGP$ offers faster convergence than the corresponding $T-OMS$. In future work we investigate analytical expressions on convergence speed and we will try to provide analytical expressions of instant $\widetilde{T}$.

VI. Related work

In current commercial implementations, the assignment of service domains to servers is done in a static fashion by an administrator. The resource provisioning (*T-SLA*) problem has been addressed by the authors of [3] and [4] in a different context. In these works, the control used to achieve the targets in Equation 3 was exercised directly on arrival rates to the servers, not scheduling. For extensive work on scheduling algorithms the reader is referred to [5].

Interesting work on the topic of server provisioning with dynamic resource allocation in service networks is presented in [7] or more recently in [6]; in the latter, the minimization of the number of virtual servers allocated to the system is examined, while in the former the authors aim to differentiate services based on both their relative profitability and QoS requirements. The performance metric in both cases is the end-to-end delay. Utility maximization problems in server operations is presented in [9], where the goal is to maximize the time average value of the instantaneous utility subject to network stability.

Moreover, related work in CPU power management policies for service applications can be found in [8] where the authors propose a method for estimating CPU demand of service requests based on linear regression between the observed request throughput and resource utilization level. Finally, our approach is closely related to the techniques of stochastic approximation, like the ones developed in [10].

VII. Conclusions and future work

In this paper, we presented scheduling policies for provisioning the CPU resources in a server tier of a multitier data center among competing service domains. The objective of such provisioning is to guarantee to each domain a prespecified percentage of the CPU resource. We provided the necessary mathematical framework and proofs of convergence for a class of policies that does not require exact knowledge of service time statistics and has adjustable communication and computation overhead. Besides theoretical analysis, extended simulations were performed. In order to address policy overhead, we evaluated how periodic execution of control actions can affect policy performance and speed of convergence.

Directions for future work include (a) theoretical analysis of convergence rate for the class of bounded-round, fair policies, (b) extension of the controller actions to include interaction of the service and server tiers, and, (c) extensions of the policies for SLAs outside the overload regime. A basic assumption of our queueing model is saturated conditions, meaning that there will always be available requests for every domain. Extensions of BGP and OMS are possible in order to make these algorithms operable when the assumption for saturated arrivals is relaxed. Note, that if saturated arrivals cannot be assumed, this implies that a domain can potentially request less traffic than agreed on the SLA. This, requires an alteration of the SLA description, such that the SLA is satisfied when the minimum of p_i or requested resources is achieved.

Acknowledgements

This work is financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning of the National Strategic Reference Framework (NSRF) Research Funding Program: Heracleitus II - Investing in knowledge society through the European Social Fund.

References

[1] E. Nitto, et al, "A journey to highly dynamic, self-adaptive service-based applications" *Journal,Automated Software Engineering,* pp. 313-341, 3-4, December 2008.

[2] L. Tassiulas, A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks", *IEEE Trans. on Automatic Control*, 37(12), Dec. 1992.

[3] K. Katsalis, L. Tassiulas, Y. Viniotis, "Distributed Resource Allocation Mechanism for SOA Service Level Agreements", *IFIP-NTMS*, Paris, 7 – 10 February 2011.

[4] M. Habib, et al, "A Service Differentiation Algorithm for Clusters of Middleware Appliances", *ICSOFT*, Sofia,Bulgaria, 26 – 29 July 2009.

[5] J. Blazewicz, et al, "Handbook on Scheduling: From Theory to Applications", ISBN-10: 3540280464, first Edition, 2007

[6] Lama, P.; Xiaobo Zhou;,"Efficient Server Provisioning with Control for End-to-End Response Time Guarantee on Multitier Clusters", Parallel and Distributed Systems, IEEE Trans,vol.23, no.1, pp.78-86, Jan. 2012

[7] M. G. Kallitsis, et al, "Distributed and dynamic resource allocation for delay sensitive network services," *IEEE GLOBECOM*, pp. 1-6, Dec. 2008.

[8] Chun Zhang et al, "Leveraging service composition relationship to improve cpu demand estimation in SOA environments,", *IEEE SCC*, pp. 317-324 , Washington, DC, USA, 7-11 July 2008.

[9] Longbo Huang, Michael J. Neely, "Utility optimal scheduling in processing networks", Performance Evaluation, Vol. 68, Issue 11, 2011, p. 1002-1021

[10] Julius C.B. Leite, Dara M. Kusic, Daniel Moss et al, "Stochastic approximation control of power and tardiness in a three-tier web-hosting cluster". Autonomic computing (ICAC '10). ACM, New York, NY, USA, 41-50.

[11] Arregoces M. et al, "Data Center Fundamentals", Cisco Press, 2004