Distributed Resource Allocation Mechanism for SOA Service Level Agreements

Kostas Katsalis*, Leandros Tassiulas* and Yannis Viniotis†
*Department of Computer & Communication Engineering
University of Thessaly and CERTH-ITI. 37 Glavani - 28th October Str, Deligiorgi Building, 382 21 Volos - Greece.
†North Carolina State University, EB II, Room 3108, USA
Email: kkatsalis, leandros@inf.uth.gr, candice@ncsu.edu

***Abstract*—Enterprise computing facilities, such as data centers or server farms typically employ service-oriented architectures (SOA) to support multiple, XML-based Web Services. They are typically architected in multiple computing tiers, in which one tier is used for, say, offloading the CPU-intensive XML processing onto a cluster of (potentially virtual) middle-ware appliances. Service differentiation in enterprise networks addresses the issues of managing the enterprise network resources in order to achieve desired performance objectives. In this paper, we define a dynamic algorithm that manages allocation of CPU time in the appliance tier. We evaluate the service differentiation capabilities of this algorithm via simulations.**

***Index Terms*—Service Differentiation, SLA, Middleware, Internet servers**

I. Introduction

In modern enterprise computing, web services and other SOA technologies experience exponential growth [1]; in such environments, complex applications, developed with SOA principles, are grouped in Service Domains (SD) that serve requests from all over the internet. Service Level Management (SLM) [2] is necessary, especially for business-critical or delay-sensitive applications. For a variety of reasons, such applications are architected in multiple computing tiers, in which one tier is used for, say, offloading the CPU-intensive XML processing onto a cluster of (potentially virtual) middleware appliances. These appliances secure, accelerate and route XML documents so enterprises can cost-effectively realize the full potential of SOA. Frequently referred as SOA appliances or middleware integrators, XML appliances implemented in hardware or software can perform many of the tasks of a typical SOA, such as an Enterprise Service Bus.

One major SLM issue is how to provide *service differentiation* to the various SDs; for example, how to allocate different amounts of CPU time to the SDs. Service differentiation becomes complicated because requests arrive in usually unpredictable traffic patterns and pose unpredictable demands for computing resources. One way to capture such unpredictability is to model system operation as a continuous alternation between two modes: "underload" and "overload" mode. Switching between these modes can happen at any arbitrary time. Service differentiation goals are typically expressed separately for each mode.

In this paper, we model an enterprise computing center as a three-tier architecture. In the first tier, we consider a number of (http) routers; the routers collect traffic from sources distributed all over the internet and forward it to the second tier, which comprises a cluster of XML appliances. After XML-related processing, the appliances route traffic to servers in the third tier, where the final processing of the requests is done. The service differentiation problem we study focuses on providing different amounts of CPU time to each SD in the XML appliance tier.

In this paper, we propose a new, dynamic, feedback-based control mechanism that achieves such differentiation. The mechanism is implemented as a software controller that controls the arrival rates of the traffic that enters every appliance in the cluster. The rates are controlled by performing open/close port operations at the http routers. Because of the complexity in the theoretical proof of the problem that we examine, in this paper we evaluate the performance of the controller only via simulations.

II. Related work

Service differentiation is a well studied issue in the literature with numerous mechanisms and algorithms presented for server optimization. A very interesting approach can be found in [3] where protection from overloads is also under consideration. The management of the Server Capacity is made by using entrance "tickets" and estimating new sessions for each class. In [4], SAA/SDA algorithm provides unqualified Service Differentiation, by activating each time one instance for domains that are above the target CPU utilization target and deactivating one instance for domains that are below the target. Three server-side mechanisms are proposed in [5] where more resource capacity is available for the high-priority processes by slowing down a background pool for low-priority requests. A combination of control theory and queueing modeling is presented in [6]. Server resources are allocated to achieve a specified average delay, given the currently observed average request arrival rate. Of course service differentiation is a very common problem also for DBMS systems, where request classifing, scheduling and performance monitoring can be used for providing differentiation [7].

Our work differs from existing solutions in that we consider the underload/overload model of system operations, with different CPU utilization goals in each mode. The solution we propose is intuitive, has low overhead and requires no manual configuration.

III. Problem Statement

We consider a system that serves M SDs. The SLA problem we want to study is the following: allocate a given percentile P_i^u of the CPU resources in the appliance tier to SD i, when the system is in underload mode; the percentile changes to P_i^o, when the system is in overload mode. The architecture of the system we study can be seen in figure 1 . Multilayer switches (or http routers) are used to spread traffic from various SDs or service classes, in a cluster of XML appliances that is responsible for XML preprocessing. After preprocessing is done web requests are sent to a cluster of application servers to handle final processing. Server tier is not examined in this paper and we are only interested in the interaction between the appliance tier and the multilayer switches.

A. Design Requirements

In order to properly set the model on which the controller will operate, several requirements must be met by the appliance operation.

- Each appliance is capable of managing which service domain to serve and can accept configuration changes in runtime.
- Each appliance has no knowledge of the state of the other appliances and the service domains the other appliances are servicing.
- When a request enters the appliance the CPU service time it will need is unknown.
- We model each appliance as a G/G/1 queueing system, served by a *FIFO* CPU scheduler.

The FIFO assumption is realistic in computing centers that serve thousands or even millions of SDs. Even in less demanding situations, the FIFO scheduling assumption is a good system approximation for appliances that server more service domains than the internal queues the appliance supports.

Since we cannot rely on a CPU scheduler to allocate CPU time to competing SDs, the (only) alternative we have is to *control the rate at which traffic enters an appliance*. We propose a feedback-based control mechanism that is used to spread traffic from the router tier into the appliance cluster; the feedback is based on appliance CPU utilization. In a nutshell, this feedback is used at the router to increase/decrease the rate of traffic from a specific SD sent to an appliance. This action is known in the literature as a router-to-appliance "open or close port operation".

IV. Controller Design

In our model the controller periodically takes a decision every T_d seconds and every time a triggered change occurs between underload and overload mode. At the time of decision t_d the controller gathers runtime statistics through feedback

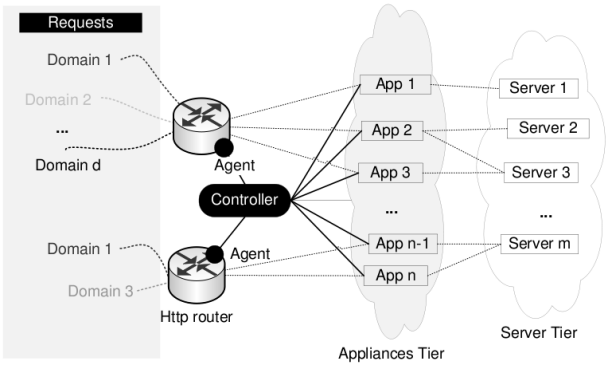

Fig. 1. *System Architecture*

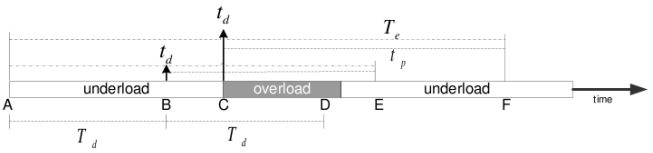

Fig. 2. Time model. *In moment B the controller makes a decision because period expires. The prediction it makes is for moment E. In time C the controller detects that the system is in overload mode and makes a prediction decision for time F.*

from all the appliances in the cluster and then perform calculations to make a prediction of the CPU utilization vector the cluster must work from this time on. CPU utilization and queue state in each appliance are the statistics of interest while the prediction is made for an interval of t_p seconds. The output of this calculations is utilization vector U_{PR}.

Since we don't know when the next mode change will occur, the t_p interval for our prediction can be short (e.g., on the order of one T_d interval) or long (e.g., on the order of the entire SLA observation period). T_e is the interval from the beginning of the observation to the time at which we make the prediction.

After the utilization vector U_{PR} is calculated, we examine how we will spread the load in each appliance per SD. Load is spread probabilistically according to an instantiation matrix created by a Port Allocation Heuristic Algorithm. Also, because the system is distributed, each router in the system can be connected to a different appliances set. For this purpose each router holds a software agent that locally adapts the controller instantiation matrix independently from other routers, based on the appliances set that it is connected and the SDs that is servicing.

V. Controller Operation Algorithm

1) The system is continuously monitored for mode change or controller decision period expired events. When such events occur, the controller detects the mode of operation (underload/overload) and collects runtime statistics from the appliance tier.
2) The utilization vectors U_j^u (U_j^o if overload) are calculated for all SDs. U_j^u is the CPU utilization domain j received until that moment for underload operation.

3) Let q_{ij} denote the number of requests in the FIFO queue of appliance i for service domain j. In this step, we calculate the number of all the domain requests that are queued in all the appliance queues: $\sum_{i=0}\sum_{j=0} q_{ij}$ and the number of requests for service domain j in the virtual queue: $\sum_{i=0} q_{ij}$.
4) Then the desired utilization for domain j is calculated according to the equation:

$$U^u_{PRj} = \frac{T_e}{t_p} * P^u_j - \frac{T_e - t_p}{t_p} * \left(U^u_j + U^u_{qj}\right) \quad (1)$$

5) The output U_{PR} vector is used to produce an instantiation matrix with all appliance/SD pairs. In this step the controller "translates" the percentages to ports that routers must open for every appliance/domain pair.

A. Why and how we use the Queue term

In Eq. 1, the controller's prediction of the utilization is not based solely on the utilization domains had at the moment of decision t_d. The term U^u_{qj} is an estimate of and accounts for utilization a SD receives due to already queued requests.

$$U^u_{qj} = \left[\max\left(0.5, \frac{\sum_{i=0} q_{ij}}{\sum_{i=0}\sum_{j=0} q_{ij}}\right)\right] * U^u_j \quad (2)$$

The intuition behind this estimation term can be explained as follows: suppose that we have a large number of requests in the cluster queues and all requests belong to SD 1; suppose further that at the moment of decision we are "in" target for this domain then we must intuitively use this "future" cpu cycles usage as already taken to perform our prediction calculations. U^u_{qj} doesn't depend on average service time measurements, but only on straightforward queue state measurements. In Eq. 2, the first term is used to control how much important we consider the queue state at decision time, as an additional percentage to utilization U^u_j (or U^o_j).

B. How we calculate CPU utilization in the cluster

In Eq. 1, P^u_j is the target underload percentage for service domain j; in order to calculate the CPU utilization every domain got until the time of decision we use the formula:

$$U^u_j(t_d) = a * \left(U^u_{1j} + U^u_{2j} + ... + U^u_{ij}\right)/N + (1-a) * U^u_j(t_{d-1}) \quad (3)$$

In Eq. 3, $U^u_j(t_d)$ is the utilization that domain j achieved during underload periods (until time t_d); similarly, $U^o_j(t_d)$ is the utilization achieved during overload periods. t_{d-1} is the time the previous decision was made and variable a is the percentage of the total time the system worked for this mode. For example for underload: $a = \frac{\text{last underload period}}{\text{total underload period}}$. Finally,

$i = 1, \ldots, N$, where N is the total number of appliances in the cluster.

TABLE I
INSTANTIATION MATRIX

	SD 1	SD 2	...	SD M
App 1	p_{11}	p_{12}	...	p_{1M}
App 2	p_{21}	p_{22}	...	p_{2M}
...	...	...	...	...
App N	p_{N1}	p_{N2}	...	p_{NM}

C. Instantiation matrix creation

Instantiation matrix is created by translating the utilization vector U_{PR} to number of ports that must be open per appliance and per SD. For each SD the following calculation is performed: $p_j = R * U_{PRj}$ where p_j is the number of ports the router will open for SD j. The creation of the instantiation matrix is based on an iterative heuristic where we try to hold the same allocation as possible as in the previous step, in order to have the minimum port open/close operations. Also heuristic tries to have same number of open ports in all the appliances in order to eliminate idle state.

D. Agents operation

After the controller computes a global instantiation matrix for all service domain-appliances pairs, all local agents in each router are updated with this information. These agents manipulate the "global" matrix locally and independently from other routers. Then this agent in each router is used to perform two operations, Scheduling and Routing.

Scheduling Algorithm used: The router will probabilistically select domain j to schedule traffic to an appliance in the cluster with probability $P^s_{ij} = \frac{\sum_i p_{ij}}{R}$ where R is the total number of ports the router can handle and $\sum_i p_{ij}$ is the number of the ports allocated for domain j.

Routing Algorithm used: A request will be sent to appliance i with probability P_{ij} calculated as: $P^s_{ij} = \frac{\sum_j p_{ij}}{R}$. In doing so, we also provide a form of load balancing because traffic is probabilistically distributed among the appliances.

VI. SIMULATIONS

There is no theoretical proof, that our controller will work. We investigate through simulations three main questions:

(**Q1**) whether our proposed mechanism meets the CPU utilization goals in both underload and overload occasions,

(**Q2**) we compare it with the static port allocation approach,

(**Q3**) we present how controller parameters such as decision or prediction period etc affect our controller performance.

For our simulation purposes we built a custom, discrete-event simulator in C# language. For the sake of simplicity incoming traffic to service domain i is modeled as a Poisson process with arrival rate $\hat{I}_i$. In order to avoid side effects as resource starvation, in all simulations presented, we use a minimum number of ports for every SD no matter what the instantiation matrix is in each cycle of measurements. In our simulations this percentage is set equal to 10%. The rest of the ports are distributed according to the controller operation

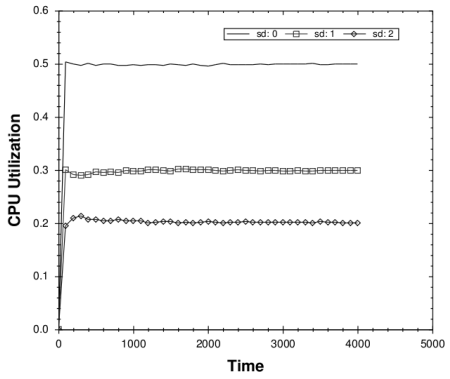

Fig. 3. Controller: $P^u = \{50\%, 30\%, 20\%\}$

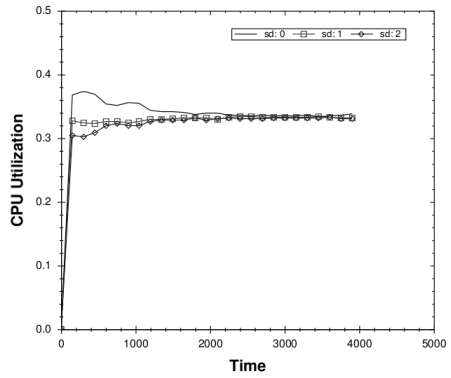

Fig. 4. Controller: $P^o = \{33\%, 33\%, 33\%\}$

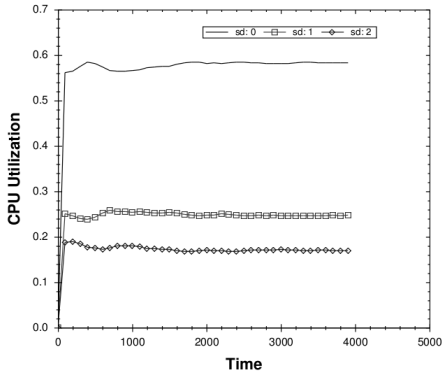

Fig. 5. Static allocation: $P^u = \{50\%, 30\%, 20\%\}$

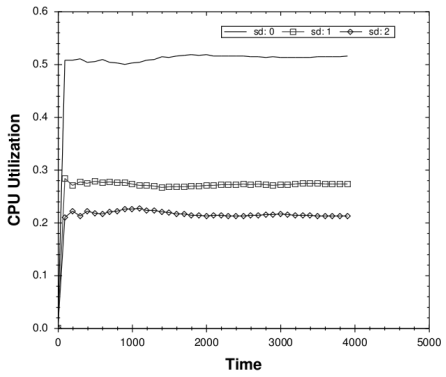

Fig. 6. Static allocation: $P^o = \{33\%, 33\%, 33\%\}$

described above. Also we consider that there is always enough traffic for each SD and that the SLA defined is feasible, meaning that the traffic for all SDs is enough to meet the target CPU utilization.

One switch is connected to 3 XML appliances that form the cluster we investigate. The traffic arrived in http router comes from 3 different domains while the SLA defines the underload CPU utilization goal as $P^u = \{50\%, 30\%, 20\%\}$ and overload CPU utilization goal as $P^o = \{33\%, 33\%, 33\%\}$. In order to properly evaluate the mechanism in a stressful environment, we decided to choose different service rates and different arrival rates for each domain while the instantiation matrix is initialized with equal number of ports for all service classes and all equally shared for all the appliances. The total number of ports is set to 1000 while the prediction period in this set of simulations is equal to the controller decision period T_d.

In figures 3-6, we provide an answer to questions **Q1** and **Q2**. The static allocation we simulated is different for the underload and overload modes. This means, for example, that, when the system is in underload mode, $1000 * 0.5 = 500$ ports are open for SD1 and $1000 * 0.333 = 333$ ports are open when the system is in overload mode.

As we can see in Figures 3 and 4, the SLA target is clearly met while the system is in underload and in overload mode. Also in comparison to the static allocation shown in Figures 5 and 6, our mechanism clearly performs better. In fact our mechanism is even sensitive and adaptable in arrival rates that change in time while the static allocation solution seems to work better only in one occasion, where arrival rates are equal and service rates are equal for all classes of traffic. Static allocation fails to keep up with the fact that SLA defined for overload conditions is different than in underload, it takes no history of CPU utilization under consideration and its performance is highly dependaple on the arrival and service rates of the SDs.

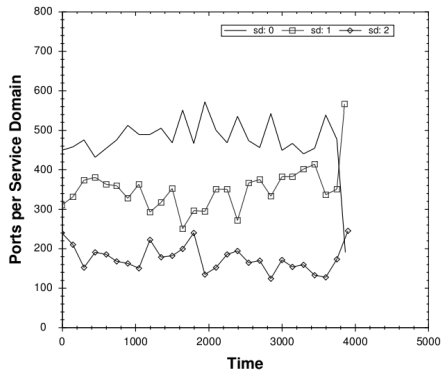

Fig. 7. Ports per Service Domain allocation

Two very interesting observations can be deduced by Fig.7

and Fig.8 where we can see the port allocation done in the instantiation matrix per domain and per appliance. As we can see the number of open ports for each SD changes each time the controller must take a decision. Because of the 10% portion of ports that all classes share, we can see that there is always a minimum of ports open to deliver requests for every SD. The iterative heuristic used for the Instantiation Matrix creation tries to share equal number of requests to each appliance and this load balancing behavior can be seen in Fig.8.

Figure 9 is used to answer question **Q3**, and in particular to show how the prediction period affects the controller performance. Long term predictions lead to goal aberrance in both underload/overload modes. Our simulations showed that controller meets the goal when predictions are made for medium to short prediction periods while by reducing the decision period by a factor of 4 the controller converges faster to the desired goal.

In figure 10, we study a distributed environment where two routers are connected to two different sets of three appliances each, all servicing requests for three SDs. Again, different arrival and service rates where selected for every SD. Here underload mode is presented but goal is reached also in overload conditions.

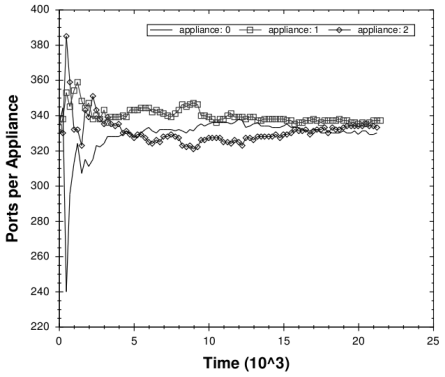

Fig. 8. Ports per appliance allocation.

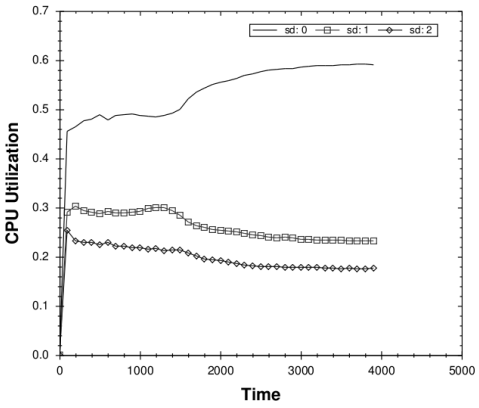

Fig. 9. Controller: $P^u = \{50\%, 30\%, 20\%\}$

Other parameters that affect controller operation is the use of threshold-tolerance parameters, the queueing term presence

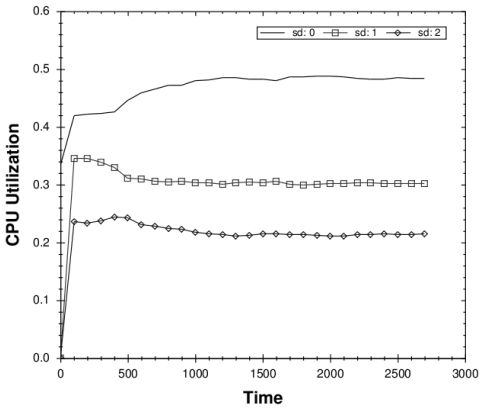

Fig. 10. Controller: $P^u = \{50\%, 30\%, 20\%\}$

in eq 1 and the formula used in eq 2 etc, but because of lack of space this parameters will be presented in future work.

VII. Conclusion And Future Work

In this paper, we have studied a problem of providing Service Differentiation to multiple Service Domains in distributed, multitiered SOA environments. The SLAs under investigation were expressed in terms of CPU Utilization SLAs in the appliance tier, where different utilization targets are defined when the system enters underload/overload conditions. We presented an intuitive, low-overhead controller that does not rely on CPU schedulers to achieve the SLA goals. Instead, it uses local agents in each router to increase/decrease the rate of traffic that is sent to the appliances; moreover, it is based on metrics that need no explicit knowledge of arrival or service statistics. More complex SLAs and end-to-end performance investigation of our approach are planned as our future work.

References

[1] M.N. Huhns, M.P. Singh, "Service-oriented computing: key concepts and principles", pp. 75 - 81, Internet Computing IEEE, Volume: 9 Issue:1, Jan-Feb 2005.

[2] L. Lewis, P. Ray, "Service level management definition, architecture, and research challenges", vol.3, pp. 1974 - 1978, GLOBECOM '99

[3] D. F. Garcia, J. Garcia et al, "A QoS Control Mechanism to Provide Service Differentiation and Overload Protection to Internet Scalable Servers", IEEE Transactions on Services Computing, vol. 2, no. 1, January-March 2009.

[4] M. Habib, Y. Viniotis, et al, "A Service Differentiation Algorithm for Clusters of Middleware Appliances", ICSOFT - International Conference on Software and Data Technologies, Sofia,Bulgaria, 26-29 July 2009.

[5] Lars Eggert and John Heidemann, "Application-level differentiated services for Web servers", World-Wide Web J., vol. 2, no. 3, pp. 133-142, Aug. 1999.

[6] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher, "Queueing Model Based Network Server Performance Control", Proc. 23rd IEEE Real-Time Systems Symp. (RTSS '02), pp. 81-90, Dec. 2002.

[7] Yue and H. Wang, "Profit-Aware Admission Control for Overload Protection in E-Commerce Web Sites", Proc. 15th IEEE Int'l Workshop Quality of Service (IWQoS '07), June 2007.