RC23732 (W0509-112) September 30, 2005
Computer Science

IBM Research Report

SLA Based Profit Optimization in Multi-tier Systems

Danilo Ardagna
Politecnico de Milano
Italy

Marco Trubian
Università degli Studi di Milano
Italy

Li Zhang
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

SLA Based Profit Optimization in Multi-tier Systems

Danilo Ardagna
Politecnico di Milano
ardagna@elet.polimi.it

Marco Trubian
Università degli Studi di Milano
trubian@dsi.unimi.it

Li Zhang
IBM Research
zhangli@us.ibm.com

Abstract

Nowadays, large service centers provide computational capacity to many customers by sharing a pool of IT resources. The service providers and their customers negotiate utility based Service Level Agreement (SLA) to determine the costs and penalties on the base of the achieved performance level. The system is often based on a multi-tier architecture to service requests. The service provider would like to maximize the SLA revenues, while minimizing its operating costs. The system we consider is based on a centralized network dispatcher which controls the allocation of applications to servers, the request volumes at various servers and the scheduling policy at each server. The dispatcher can also decide to turn ON or OFF servers depending on the system load. This paper designs a resource allocation scheduler for such multi-tier environments so as to maximize the profits associated with multiple class SLAs. The overall problem is NP-hard. We develop heuristic solutions by implementing a local-search algorithm. Results are presented to demonstrate the benefits of our approach.

1. Introduction

To reduce their management cost, companies often outsource their IT infrastructure to third party service providers. Large service centers have been set up to provide services to many customers by sharing the IT resources. This leads to the efficient use of resources and a reduction of the operating costs. The service providers and their customers often negotiate utility based Service Level Agreements (SLAs) to determine costs and penalties based on the achieved performance level. The service provider needs to manage its resources to maximize its profits. Utility based optimization approaches are commonly used for providing load balancing and for obtaining the optimal trade-off among job classes for Quality of Service levels [6]. One main issues of these systems is the high variability of the workload. For example for Internet applications, the ratio of the peak to light load is usually in the order of 300% [3]. Due to such large variations in loads, it is difficult to estimate workload requirements in advance, and planning the capacity for the worst-case is either infeasible or extremely inefficient. In order to handle workload variations, many data centers have started to employ self-managing techniques [4, 5], such that resources are dynamically allocated among applications of different customers on the base of short-term demand estimates. The goal is to meet the application requirements while adapting IT architecture to workload variations. Usually, each request requires the execution of several applications allocated on multiple physical tiers. This paper focuses on the design of a resource allocator for multi-tier environments. The goal is to maximize the revenue while balancing the cost of using the resources. The cost usually includes the cost of energy, the cost of air conditioning of the data center [3] and software and hardware cost of resources allocated on demand. The overall profit (utility) includes the revenues and penalties incurred when Quality of Service guarantees are satisfied or violated. The resource allocator can establish the set of applications executed by each server, the request volumes at various servers and the scheduling policy at each server. The allocator can also decide to turn ON or OFF servers depending on the system load. Resource allocator policies are applied on multiple time scales. Optimum load balancing and scheduling are determined in real-time and applied every few minutes, while application allocation and servers status are evaluated with a greater time scale, i.e. every half an hour, in order to reduce system re-configuration overhead. We model the problem as a mixed integer nonlinear programming problem and develop heuristic solutions based on a local-search approach. The neighborhood exploration is based on a fixed-point iteration (FPI) technique, which iteratively solves a scheduling and a routing problem by implementing a gradient method. Experimental results are presented to show the benefits of our approach. The remainder of the paper is organized as follows. Section 2 introduces the overall system model. The optimization problem formulation is presented in Section 3. The experimental results in Section 4 demonstrate the quality and efficiency of our solutions. Conclusions are drawn in Section 5.

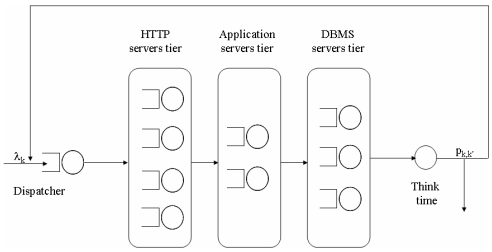

Figure 1. Network Queue Model

2 The System Model

The system under study is a distributed computer system consisting of M heterogeneous servers. There are totally K classes of request streams. Each class $k \in K$ request can be served by a set of server applications (application tiers in the following) according to the client/server paradigm. For simplicity assume that each class k request is associated with a single customer. The architecture comprises a requests dispatcher in front of servers that assigns incoming requests to individual server. The controller can also establish the allocation of application tiers to servers and the scheduling policy at each server. The service discipline under consideration is the Generalized Processor Sharing (GPS) class. The controller can also turn OFF and ON individual server in order to reduce the overall cost. For each class $k \in K$, a linear utility function is defined to specify the per request revenue (or penalty) incurred when the average end-to-end response time R_k, from multiple tiers, assumes a given value. $-m_k$ indicates the slope of the utility function ($m_k = u_k/z_k > 0$) and z_k is the threshold that identifies the revenue/penalty region (i.e., if $R_k > z_k$ the SLA is violated and penalties are incurred). Linear utility functions are currently proposed in the literature (see for example [2, 5]), anyway our approach can be extended in order to consider a broad family of utility functions. We only assume that the utility function is monotonically non-increasing, continuous and differentiable. Monotonic non-increasing utility functions are very realistic since the better the achieved performance by end users, the higher are the revenues gained per request by the Service Provider. The data center is modeled by a queueing network composed of a set of multi-class single-server queues and a multi-class infinite-server queue. The first layers of queues represent the collection of applications supporting requests execution. The infinite-server queues represent the client-based delays, or think time, between the server completion of one request and the arrival of the subsequent request within a session (see Figure 1). User sessions begin with a class k request arriving to the data center from an exogenous source with rate λ_k. Upon completion the request either returns to the system as a class k' request with probability $p_{k,k'}$ or it completes with probability $1 - \sum_{l=1}^{K} p_{k,l}$. Let Λ_k denote the aggregate rate of arrivals for class k requests, $\Lambda_k = \sum_{k'=1}^{K} \Lambda_{k'} p_{k',k} + \lambda_k$.

In next sections the following notation will be adopted:

K	:=	set of request classes;
N_k	:=	number of tiers involved in the execution of class k;
M	:=	number of servers at the data center;
C_i	:=	capacity of server i;
c_i	:=	time unit cost for server i ON;
$A_{i,k,j}$	:=	1 if server i can support the execution of application tier j for class k job, 0 else;
$\mu_{k,j}$	:=	max service rate of a capacity 1 server for executing processes at tier j for class k.

The routing matrix $[A_{i,k,j}]$ is usually obtained as a result of an optimization problem [7]. It is used to assign private servers to distinct Web sites for dedicated e-commerce transaction servers. It can also limit the number of Web sites assigned to servers due to caching issues [7]. Thus, the routing matrix is used to model the assignment of application tiers to individual servers. In practice, $c_i \propto C_i$, if power is the main cost associated with turning a server on.

The decision variables of our model are the followings:

x_i	:=	1 if server i is ON, 0 otherwise;
$\lambda_{i,k,j}$	:=	rate of execution for class k jobs at tier j on server i;
$\phi_{i,k,j}$	:=	GPS parameter at server i for executing processes at tier j for class k jobs;
$z_{i,k,j}$	:=	1 if process at tier j for class k jobs is assigned to server i, 0 otherwise.

The analysis of multi-class queueing system is notoriously difficult. We use the GPS bounding technique in [8] to approximate the queueing system. Under GPS, the server capacity devoted to class k requests for tier j at time t (if any) is $C_i\phi_{i,k,j} / \sum_{k' \in \mathcal{K}(t)} \sum_{j=1}^{N_{k'}} \phi_{i,j,k'}$, where $\mathcal{K}(t)$ is the set of classes with waiting requests on server i at time t. Requests at different tiers within each class and on every server are executed either in a First-Come First-Serve (FCFS) or a Processor Sharing (PS) manner. Under FCFS, we assume that the service time for class k requests at server i has an exponential distribution with mean $(C_i\mu_{k,j})^{-1}$, whereas, under PS, service time of class k requests at cluster i follows a general distribution with mean $(C_i\mu_{k,j})^{-1}$, including heavy-tail distributions of Web servers. In the approximation, each multi-class single-server queue associated with a server application is decomposed into multiple independent single-class single-server queues with capacity greater than or equal to $C_i\phi_{i,k,j}$. The response times evaluated in the isolated per-class queues are then upper bounds on the corresponding measures in the original system. Under these hypothesis $R_{i,k,j}$, i.e., the average response time for the execution of the process at tier j of class k jobs at server i can be evaluated as $R_{i,k,j} = \frac{1}{C_i\mu_{k,j}\phi_{i,k,j} - \lambda_{i,k,j}}$ (see [8]). We adopt analytical models in order to obtain an in-

dication of system performance and response time as the authors in [4]. Note also that the approximation provided in [8] is asymptotically correct for high loads. In Section 4, resource allocation results will be validated by simulations considering also heavy-tail distributions for service time.
The average response time for class k jobs is the sum of the average response times at each tier computed over all servers, and is given by $R_k = \frac{1}{\Lambda_k}\left(\sum_{i=1}^{M}\sum_{j=1}^{N_k}\lambda_{i,k,j}R_{i,k,j}\right)$. Our objective is to maximize the difference between revenues from SLAs and the costs associated with servers ON in the inter-scheduler time period which can be expressed as $\sum_{k=1}^{K}\Lambda_k(-m_kR_k+u_k)-\sum_{i=1}^{M}c_ix_i$, which, after substituting R_k, becomes:

$$\sum_{k=1}^{K}(-m_k\sum_{i=1}^{M}\sum_{j=1}^{N_k}\frac{\lambda_{i,k,j}}{C_i\mu_{k,j}\phi_{i,k,j}-\lambda_{i,k,j}})+\sum_{k=1}^{K}u_k\Lambda_k-\sum_{i=1}^{M}c_ix_i$$

3. Optimization Problem

The overall optimization problem can be formulated as:

$$P1)\ \max\sum_{k=1}^{K}(-m_k\sum_{i=1}^{M}\sum_{j=1}^{N_k}\frac{\lambda_{i,k,j}}{C_i\mu_{k,j}\phi_{i,k,j}-\lambda_{i,k,j}})-\sum_{i=1}^{M}c_ix_i$$

$$\sum_{i=1}^{M}\lambda_{i,k,j}=\Lambda_k; \qquad \forall k,j \tag{1}$$

$$\sum_{k=1}^{K}\sum_{j=1}^{N_k}\phi_{i,k,j}\le 1; \qquad \forall i \tag{2}$$

$$\sum_{(k,j)\in\mathcal{B}_l}z_{i,k,j}\le 1; \qquad \forall i,l \tag{3}$$

$$z_{i,k,j}\le A_{i,k,j}x_i; \qquad \forall i,k,j \tag{4}$$

$$\lambda_{i,k,j}\le\Lambda_k z_{i,k,j}; \qquad \forall i,k,j \tag{5}$$

$$\lambda_{i,k,j}<C_i\mu_{k,j}\phi_{i,k,j}; \qquad \forall i,k,j \tag{6}$$

$$\lambda_{i,k,j},\phi_{i,k,j}\ge 0; x_i,z_{i,k,j}\in[0,1] \qquad \forall i,k,j$$

Constraint family (1) entails that the traffic assigned to individual servers, and for every tier, equals the overall load predicted for class k jobs. Constraint family (2) expresses the bounds for GPS scheduling parameters. Constraint family (3) is introduced in order to assign distinct servers to subset of applications, where $\mathcal{B}_l$ is a subset of the indexes in $N_k\times K$. For example, a servlet engine can be executed with a Web server or an application server instance, vice versa application and DBMS servers are usually allocated to individual machines (i.e., eventually supporting multiple application or DBMS instances) for management and security reasons. The constraint family (4) allows to assign application tiers to severs according to $A_{i,k,j}$ and only if servers are ON. Constraint family (5) allows to execute requests at a server only if the corresponding application tier has been assigned to it. Finally, constraints family (6) guarantees that resources are not saturated. Note that $\sum_{k=1}^{K}u_k\Lambda_k$ is a constant and it has been dropped in the objective function.
Problem P1) is a mixed integer nonlinear programming problem. Even if the set of server ON is fixed, i.e. the value of variables x_i has been determined, the joint scheduling and routing problem is difficult since the objective function is neither concave nor convex. We have shown [1] by diagonalization techniques, that the eigenvalues of the Hessian of the cost function are mixed in signs.
The given problem can be solved by nonlinear commercial tools only for small size instances. For all instances of interest, an heuristic approach has to be considered. We have decomposed the problem into three sub-problems [1]. First, we evaluate the overall capacity to be provided to each tier. Then, tiers are assigned to servers. Finally, the routing and scheduling problems are solved by implementing a fixed-point iteration (FPI) technique, which iteratively solves a scheduling and a routing problem. The solution is then enhanced by a local-search algorithm that turns on and off servers and modifies the assignment of application tiers to servers.

4. Experimental Results

The effectiveness of our approach has been tested on a wide set of randomly generated instances. All tests have been performed on a 3 GHz Intel Pentium IV workstation. The number of servers has been varied between 40 and 400 (with steps of 40). Data centers up to 200 job classes have been considered and the number of tiers has been varied between 2 and 4. $A_{i,k,j}$ values were randomly generated, and every server was shared by at most five customers. Service times were randomly generated and for each test case the load was increased in a way that the utilization of data center resources varied between 0.2 and 0.8. N_k, m_k and z_k values have been randomly generated, z_k is proportional to the number of tiers N_k and to the overall demanding time at various tiers of class k job. m_k varied uniformly between 2 and 10. Usually the FPI converges very quickly and performs less than 10 iterations and execution time is always lower than 8 secs. Local search usually performs a greater number of iterations but the execution is stopped when the execution time achieves 30 min. Indeed, our implementation allows to compute optimum load balancing and scheduling in real-time, while application allocation and servers status can be evaluated with a greater time scale, i.e. every half an hour. As a typical example, Table 1 shows the average improvement which can be obtained from the initial solution (% IS column) and from the first FPI (% FP) by applying the local search approach for a test case with 400 servers. In this test case, 100 job classes are allocated on four tiers, and 100 servers are assigned at each tier by $A_{i,k,j}$. The column reports the overall execution time (in minutes). Results are

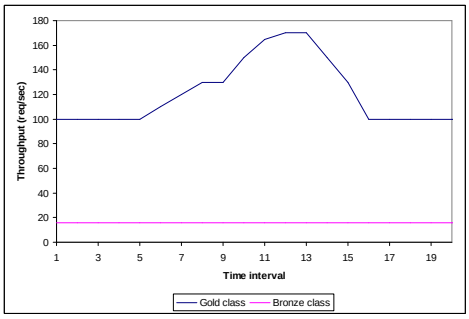

Figure 2. Throughput Variation

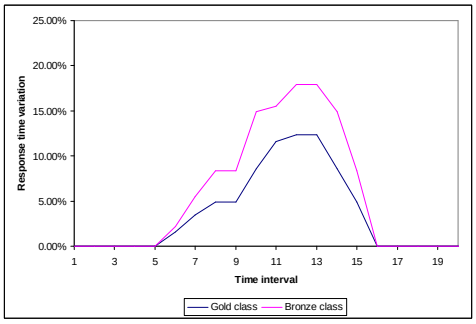

Figure 3. Response Time Variation

grouped by the average data center utilization. Tests have been run by considering homogeneous (only servers of capacity 1) and heterogeneous systems (half servers with capacity 1 and half of capacity 2) where the load was evenly shared among different tiers, or more realistically, in the second case higher tiers were the system bottleneck. Test results show that the optimal solution for heterogeneous systems adopts a lower number of servers of the corresponding homogeneous case using mainly servers of higher capacity. This is expected since servers of grater capacity give better performance and hence better revenues despite their higher cost. We have validated by simulation our solutions based on analytical models. We generated arrival streams with different classes of interarrival times to investigate the effects of non-Poission arrivals. We also simulated job service times from the log-Normal and Pareto families, with the same mean and standard deviations. The regenerative simulation runs until a minimum number of regenerative cycles have been reached and the collected statistics from the servers and queues all reach the desired confidence level (95%). We have observed consistent results from different simulation runs. In the plots reported in Figures 2 and 3 a gold and a bronze job class are considered ($|m_{gold}| >> |m_{bronze}|$). The service time distribution is Pareto. In the control time interval the gold class load increases by 70% (Figure 2) and the bronze class response time has a greater increase (about 20%) than that of the gold one (about 12%).

5. Conclusions

We proposed an allocation controller for multi-tier data center environments which maximizes the profits associated with multi-class Service Levels Agreements. The cost model consists of a class of utility functions which include revenues and penalties incurred depending on the achieved level of performance and the cost associated with servers. The overall optimization problem which considers the set of servers to be turned ON, the allocation of applications to servers and routing and scheduling at servers as joint control variables, is NP-hard and we developed a heuristic solution based on a local search algorithm. Future work will introduce strict QoS performance guarantees, i.e., deadlines for the response times.

Table 1. Improvement of the Optimization Steps

	Homogeneous			Heterogeneous		
U	% IS	% FP	Time	% IS	% FP	Time
0.2	257.3%	229.1%	30	300.1%	47.9%	30
0.3	221.7%	90.2%	26	33.6%	18.0%	30
0.4	78.5%	54.2%	22	41.8%	13.8%	25
0.5	120.1%	60.2%	28	29.1%	5.2%	13
0.6	75.1%	39.2%	18	22.0%	2.1%	20
0.7	100.2%	19.3%	12	15.3%	1.1%	15
0.8	21.3%	15.4%	5	115.1%	2.3%	10

References

[1] Ardagna, D., Trubian, M., Zhang, L. 2004. *On Maximizing SLA in multi-tier Web Applications*. Politecnico di Milano Technical Report n. 2004.34.

[2] Bennani, M., Menascé D. 2004 *Accessing the Robustness of Self-Managing Computer Systems under Highly Variable Workloads*. In Proc. of ICAC 2004. On-Demand

[3] Chase , J. S., Anderson, D. C. 2001 *Managing Energy and Server Resources in Hosting Centers*. In Proc. of the 18th ACM symposium on Operating systems principles. Operational

[4] Liu, Z., Squillante, M. S., Wolf, J. 2001 *On maximizing service-level-agreement profits*. In Proc. of the 3rd ACM conf. on E-Commerce.

[5] Urgaonkar, B., Shenoy, P. 2004. *Sharc: Managing CPU and Network Bandwidth in Shared Clusters*. IEEE Trans. on Parallel Distrib. Syst. 15,1, 2-17.

[6] Walsh, W., Tesauro, G., Kephart, J., Das, R. 2004. *Utility Functions in Autonomic Systems*. In Proc. of ICAC 2004.

[7] Wolf, J., Yu, P. S. 2001. *On balancing the load in a clustered Web farm*. ACM Trans. on Internet Technology, 1,2, 231-261.

[8] Zhang, Z. L., Towsley, D., Kurose, J. 1995. *Statistical analysis of the generalized processor sharing scheduling discipline*. IEEE Journal on Selected Areas in Comm., 13,6, 1071-1080. Workload 63-77.

[9] Zhang, L., Ardagna, D. 2004 *SLA Based Profit Optimization in Autonomic Computing Systems*. In Proc. of ICSOC 2004.