Web Server Performance Optimization

Popular websites are expected to handle huge amounts of requests simultaneously within a reasonable timeframe. On top of that, boosted by the recent developments in server-side scripting technology, many web servers also must perform a significant amount of server-side processing. This puts a tremendous burden on the availability of processing capacity. Consequently, for many popular websites, where usually plenty of core-network capacity is available, web servers tend to become performance bottlenecks. This is often illustrated by examples of badly performing websites that regularly appear in the newspapers. In this article, we propose new dynamic thread assignment policies that minimize the average response-time performance of web servers and we validate the performance of these policies.

Wemke van der Weij

heeft Operations Research and Management gestudeerd aan de Universiteit van Amsterdam. Zij is nu promovendus aan het Centrum voor Wiskunde en Informatica in Amsterdam, met als onderzoeksgebied 'Performance Analyse van Hiërarchische Wachtrijnetwerken'.

Sandjai Bhulai

is verbonden aan de Vrije Universiteit van Amsterdam. Hij is daar universitair docent op het gebied van Markov beslissingsketens, communicatienetwerken en call centers.
Dit artikel is een verkorte versie van het paper "Dynamic Thread Assignment in Web Server Performance Optimization" die verstuurd is ter publicatie.

Web servers are typically equipped with a number of thread pools to handle incoming requests. The thread pools are dedicated to process a specific processing step of each request [1]. The performance of web servers is therefore largely dependent on the thread-management policy used. Nowadays, most web servers implement either static policies, where the number of threads is fixed, or dynamic policies, where threads can be created (killed) if the number of active threads is above (below) some threshold value. Although the implementation of these thread policies is common practice, remarkably little is known about the implications of thread-assignment policies on the performance of the web servers [2], [3], [4]. Moreover, the commonly used thread policies do not take into account the complex interaction between the server threads that compete for access to a shared processor resource [6].

In this article, we propose new dynamic thread-assignment policies that minimize the average response-time performance of web servers. To this end, we model the web server as a two-layered tandem of multi-threading queues, where the active threads compete for access to a common hardware resource. The proposed policies are based on on-the-fly information about both the number of active threads and the probability distribution of the required service time per request; in practice, the required information is easily accessible. Our results show that the optimal dynamic thread-assignment policies yield strong reductions in the response times. To validate the model, we have tested the performance of our policies in an experimental setting on an Apache web server. The experimental results show that our dynamic thread policies indeed lead to significant reductions of the response time, which demonstrates the practical usefulness of the results.

Model description

In this section we model the problem of dynamic thread assignment in the context of a multi-layered queueing system with a shared resource. We consider a network of N queues in tandem with a single shared processor for serving arriving requests. Requests arrive according to a Poisson process with rate λ to the first queue. At each queue, threads can be spawned which may be assigned to a request. When a request is assigned to a thread at queue i, it receives service for a duration of S_i with mean β_i for $i = 1,\ldots,N$. However, during service, the request only gets a fraction $1/k$ of the total capacity of the server when the total number of spawned threads is k. Upon completion of service, the thread is terminated and the request proceeds to queue $i + 1$ if $i < N$, or it leaves the system otherwise. If a request is not assigned a thread,

the request joins an infinite buffer at the queue and waits until it is assigned a thread. In figure 1 the model is presented graphically.

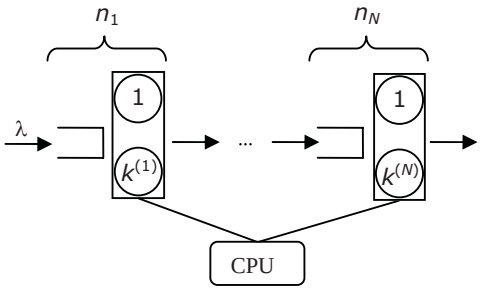

Figure 1: A two-layered queueing system

To obtain optimal thread-assignment policies that minimize the average expected response times, we model the system in the framework of Markov decision theory. Therefore, we model the service-time distribution of S_i by a phase-type distribution. In this article we only consider exponentially distributed service times of the requests in detail. For the other phase-type distributions we give the optimal policy without going into details.

Dynamic thread management

Let us first consider exponentially distributed service times. The motivation to study this distribution is that it accurately models simple HTTP requests, such as the GET request for an HTML document. For these service-time distributions the state of the system can be described by the tuple $(\vec{n},\vec{k})$, with n_i the number of requests in the system at queue i and $k^{(i)}$ the number of outstanding threads at queue i which equals the number of requests in service at queue i. The dynamic programming operator T for this system is as follows:

$$TV(\vec{n},\vec{k}) = \sum_{i=1}^{N} n_i + \lambda H(\vec{n}+e_1,\vec{k}) +$$
$$\sum_{i=1}^{N} \frac{k^{(i)}\mu^{(i)} H(\vec{n}-e_i+e_{i+1},\vec{k}-e^{(i)})}{k^{(1)}+\cdots+k^{(N)}} +$$
$$\left[1-\lambda-\sum_{i=1}^{N}\frac{k^{(i)}\mu^{(i)}}{k^{(1)}+\cdots+k^{(N)}}\right]V(\vec{n},\vec{k}),$$

with e_i the unit vector with all entries zero, except for the i-th entry for $i = 1,\ldots, N$, and with e_{N+1} the zero vector. The actions are modeled by $H(\vec{n},\vec{k}) = \min\{V(\vec{n},\vec{k}+ae^{(i)}) \mid i = 1,\ldots,N, a \in \mathbb{N}_0\}$. In the above equation, $V(\vec{n},\vec{k})$ is called the relative value function and this has the interpretation of the asymptotic difference in total costs from starting in the state $(\vec{n},\vec{k})$ instead of some reference state. The optimal decisions are a solution (in vector notation) of the optimality equation $g + V = TV$ [6]. The first term in the expression $TV(\vec{n},\vec{k})$ models the direct costs, i.e., the total number of requests in the system. Note that Little's Law [7] directly relates the total number of requests in the system to the average expected response times. The second term models the arrivals and the third term models the transitions from a request at queue i to queue $i + 1$. The last term is an uniformization constant.

From the relative value function for $N = 1$ it follows that all work-conserving policies have the same performance in terms of average expected response times. In this case the relative value function has the property that for $n > 0$, $V(n,0) \geq V(n,1)$ and $V(n,1) = V(n,1+a)$ for $a < n - 1$. Therefore, an optimal policy is to serve one request only. This policy is denoted by $\pi^{(1)}$. Policy $\pi^{(1)}$ turns out to be also optimal for $N > 1$. When the system starts from an arbitrary state, any policy that coincides with $\pi^{(1)}$ within finite expected time is optimal. However, the bias optimal policy (which is a more sensitive condition) is to move towards policy $\pi^{(1)}$ by spawning threads for all requests starting from the last queue and going backwards to the first queue at each time instant. For simplicity, we assume that we start with an empty system.

In general, the optimal policy is much more complex. In the exponential case all requests at the same queue have the same mean service time, but this is generally not the case. Due to this differentiation in service times at each queue, requests can overtake each other when threads are available. This leads to a general theorem for minimizing the average expected response times by dynamic thread-management rules. Define the tuple $(\vec{n},\vec{q},\vec{k})$, where entry n_i of the vector $\vec{n}$ denotes the number of requests in queue i and in service at queue i. Let $\vec{q}$ be the vector that denotes the number of requests in each phase in each queue, and let $\vec{k}$ be the number of outstanding threads in each phase in each queue. Then we have the following theorem.

Theorem: Let $\varphi_i(\vec{n},\vec{q},\vec{k})$ denote the decision rule that describes the thread-management rule at queue i, for $i = 1,\ldots, N$. Let φ_i be such that it spawns τ threads for requests in phase j at queue i given state $(\vec{n},\vec{q},\vec{k})$ when the τ phase j requests overtake in expectation requests in queue $j \geq i$ under decision rules $\varphi_{i+1},\ldots,\varphi_N$. Then policy $\pi = (\varphi_i,\ldots,\varphi_N)$ is optimal.

Numerical Experiments

In the previous section the optimal policies were given for phase-type distributed service types. In this section we compare the optimal policies with some commonly used policies in web server applications. We compare the theoretical gain of the different policies, and we also compare the theoretical gain with the gain observed

in experiments on an Apache web server. We compare the following policies:

- $\pi^{(1)}$ is the policy that spawns at most one thread,
- $\pi^{(\infty)}$ is the policy that spawns an unlimited number of threads,
- $\pi^{(*)}$ is the optimal policy derived from the relative value function, given in the previous section.

We are interested in the gain defined by $(EW(\pi^{(s)})-(EW(\pi^{(*)}))/EW(\pi^{(*)})$, where $EW(\pi)$ is the average expected response time under policy π. We compare the gain for the parameter settings presented in Table 1. We consider $N = 2$, and exponentially distributed and hyper-exponentially distributed service times. If only $\beta_1^{(i)}$ is presented, the service times are exponentially distributed at queue i, and otherwise with probability $p_1^{(i)}$ the request has mean service time $\beta_1^{(i)}$ and with probability $1 - p_1^{(i)}$ mean service time $\beta_2^{(i)}$.

case	$p_1^{(1)}$	$p_2^{(2)}$	$\beta_1^{(1)}$	$\beta_2^{(1)}$	$\beta_1^{(2)}$	$\beta_2^{(2)}$
1			1.00		1.00	
2		0.91	1.00		0.55	5.45
3	0.91		0.55	5.45	1.00	
4	0.91	0.91	0.55	5.45	0.55	5.45
5			1.00		1.00	
6		0.91	1.00		2.75	27.25
7	0.91		0.55	5.45	1.00	
8	0.91	0.91	0.55	5.45	2.75	27.25
9			1.00		1.00	
10		0.91	1.00		0.55	5.45
11	0.91		2.75	27.75	1.00	
12	0.91	0.91	2.75	57.75	0.55	5.45

Table 1: Scenarios used for comparing the performance of policies

In Figure 2 the gains of using policy $\pi^{(*)}$ over policy $\pi^{(1)}$ and $\pi^{(\infty)}$ are presented. The gains of the Apache web server are also compared to the theoretically calculated gains. The figure shows that there are significant gains in the performance by applying the optimal policy. Furthermore, the figure shows that the observed gains in the experiment match the theoretical gains, which establish the practical usefulness of the model.

Conclusion

In this article we have considered the important problem of dynamic thread assignment in web servers such that the average expected response time is minimized. We have obtained the optimal policy for phase-type distributed service times. From the theoritical model it follows that the improvement in performance is significant. Validation of the gains in performance in an experimental setting shows the perfect match between theory and practice and illustrates the practical usefulness. For more details we refer to the websites of the authors where the full paper and related work can be found.

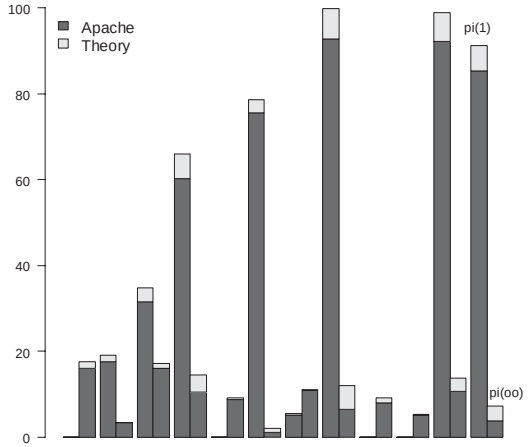

Figure 2: Comparison of gains: Apache vs. model

References

[1] Van der Mei, R.D., Hariharan, R. and Reeser, P.K. (2001). Web server performance modeling. *Telecommunication Systems*, **16**, 361–378.

[2] Harchol-Balter, M., Bansal, N. and Schroeder, B. (2000). *Implementation of SRPT scheduling in web servers*.

[3] Harchol-Balter, M., Bansal, N., Schroeder, B. and Agrawal, M. (2001). SRPT scheduling for web servers. *Lecture Notes in Computer Science*, **2221**, 11–21.

[4] Crovella, M.E., Frangioso, R. and Harchol-Balter, M. (1999). *Connection scheduling in web servers*, in Proceedings USENIX symposium on Internet Technologies and Systems.

[5] Hariharan, R., Ehrlich, W.K., Reeser, P.K. and Van der Mei, R.D. (2001). *Performance of web servers in a distributed computing environment,"* in Teletraffic Engineering in the Internet Era, J.M. de Souza, N. da Fonseca, and E. de Souza e Silva, Eds., 2001., pp. 137–148, also Proceedings 17th International Teletraffic Congress (Salvador, December 2001).

[6] Puterman, M.L. (1994). *Markov Decision Processes*. John Wiley & Sons.

[7] Tijms, H.C. (1994). *Stochastic Models: An Algorithmic Approach.* Chichester, England. John Wiley & Sons.