Predictable Scaling Behaviour in the Data Centre with Multiple Application Servers

Mark Burgess and Gard Undheim

Oslo University College, Norway
mark@iu.hio.no

Abstract. Load sharing in the data centre is an essential strategy for meeting service levels in high volume and high availability services. We investigate the accuracy with which simple, classical queueing models can predict the scaling behaviour of server capacity in an environment of both homogeneous and inhomogeneous hardware, using known traffic patterns as input. We measure the performance of three commonly used load sharing algorithms and show that the simple queueing models underestimate performance needs significantly at high load. Load sharing based on real-time network monitoring performs worst on average. The work has implications for the accuracy of Quality of Service estimates.

1 Introduction

Network services are now a central paradigm in the management of commercial resources in a market framework. They are divided broadly into network level services (such as sale of subscriber lines) and application level services (such as Internet banking and other Web-based portals). Predicting the expected service delivery is an important prerequisite for selling services in a market place; these expectations are then codified in Service Level Agreements in order to place the relationships on a contractual footing[1]. In simplistic terms, a service provider's task is to deliver the promised service level in as lucrative a way as possible (the Service Level Objective). There is a clear need for models which can reliably predict the relationship between supply and demand.

Meeting the requirements set by SLAs, with variable demand and resources, has become a keen point of interest in the last year[2–4]. A common strategy for trying to determine this relationship is to opt for "over-provision", i.e. providing more resources than are needed, by some acceptable margin. This is a straightforward way of dealing with uncertainty, but it can be a relatively expensive way and, without a model for capacity, it is not clear exactly what margin is required for meeting the peaks in demand. Another approach would be to try to adapt the service scheduling strategy and resource pool to better adapt to changing requirements[3, 4]. In each case, there is a role for parallelism in the provision of the services both for the purpose of *load sharing* and *redundancy.*

In this paper, we consider how well one is able to predict the scalability of an application service by means of low level load balancing. We examine how the simple queueing models behave as estimators of local load balancing techniques, and we look at how response times vary as a function of traffic and number of servers. This information is of direct interest to data centre managers, especially when the hardware in a data centre in inhomogeneous, with varying processing capacity.

We quantify the expected uncertainty in service levels for different kinds of demand and equipment, so as to provide an estimate of service level margins, suitable for inclusion in a Service Level Agreement.

2 Experiment

Consider the following problem. Suppose our data centre resources are not fixed, but that we can add additional computing nodes in a cluster-like fashion, how can we adapt to changing service levels using models to predict the effects of changing the number of servers used in parallel?

We base our experiment on simulated traffic running on real hardware. The traffic is a stream of page-lookups to PHP-enabled web pages, generated using `httperf`. Each page involved CPU-bound iteration (spin delay); this was necessary to offer the server any load at all, as the server CPU hardware was comparable to that of the traffic generator.

There is some controversy in the literature about the exact arrival characteristics of Web traffic requests and how this affects the ability of a server to process them. We do not wish to involve ourselves in that question here (see [5] for a treatment of this matter); rather we wish to answer a far less ambitious question: assuming that the traffic follows the simplest, most predictable pattern, are we able use the corresponding models for load sharing to predict the performance of the balanced system? As we shall see below, this is far from being a straightforward question.

In our tests, we use a commercial Alteon 2208 load balancing switch as the dispatcher, to a back end of symmetrical IBM blade servers running Apache2 on GNU/Linux. These servers can be made identical, and they can be switched into power saving modes of lower CPU frequency (from 1.2 GHz to 2.8 GHz, stepwise). We make use of this feature to simulate the transition from homogeneous to inhomogeneous server hardware.

We would like to be able to predict the performance of servers in a data centre, with regard to

- Average delivery time of a job in the system.
- Change in average delivery time on adding new hardware.

This information is of direct use to a data centre manager adapting to demand, and it is also useful to the sales staff who decide when to promise specific service levels in their agreements.

Since we are dealing with random processes, exact prediction is not an option. A realistic picture is to end up with a probability or confidence measure e.g. what is the likelihood of being able to be within 80% or 90% of an SLA target value?

The shape of this probability distribution will also answer the question: what is the correct level at which to 'over-provision' a service in order to meet the SLA requirements. Fig. 1 shows the schematic arrangement of servers.

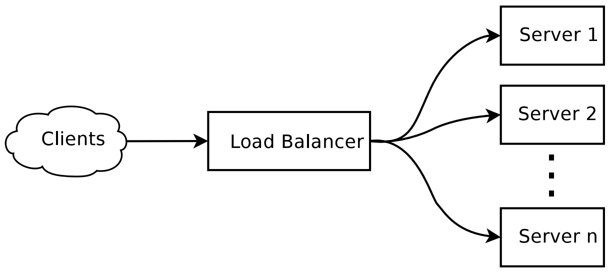

Fig. 1. The topology for low level load sharing using a commercial dispatcher.

3 Queues and service prediction

Queueing theory is the domain of experts, not of data centre engineers. Indeed, in engineering terms, only the simplest queueing models are really used as predictors[6], because the more complex models require input that is not available to the average engineer. What is the distribution of traffic at a given moment? Does it vary throughout the course of a day or a week? These are questions that are not easily answered and, even if they are, those answers are not easily used in an advanced queueing model to give a simple result. We need therefore simple models that give approximate rules of thumb.

Recent work by Sauvé et al discusses the matter of trying to convert low level knowledge of a system into SLA margins, in a multi layered model using the cost of the equipment and services to add an element of risk analysis to the problem[2]. They use the simplest kind of queue model to estimate capacity – this is understandable given the complexity of the problem. However, they do not evaluate whether the model actually has any predictive power. Another group has considered the idea of combining queueing models with an adaptive control framework to allow service capacity to change with demands[3]. This work is rather interesting from a design point of view, but it does not answer the basic questions that an engineer might ask, such as how to deal with the varying demand in lieu of this new technology.

The basic queueing models we consider are, in Kendall notation, the $M/M/1$ queue and the $M/M/n$ queue for n servers[7]. The M stands for 'memoryless', i.e. discrete time Poisson inter-arrival time traffic. This is the 'simplest' case for queues, since Poisson traffic has simple analytical properties. It is known that Web traffic is rarely this well-behaved in practice, however, we use it as a source with the rationale that, if we cannot predict Poisson traffic, then we certainly cannot deal with more long-tailed traffic patterns. There is evidence to suggest that it is good enough to compensate for this model with additional uncertainty[5].

The total quality of service in a system must be viewed as the combined qualities of the component parts[8]. It is a known result of reliability theory[9] that low level parallelism is, in general, more efficient than high level parallelism, in a component-based system. Thus a single $M/M/n$ queue is generally superior in performance to n separate $M/M/1$ queues (denoted $(M/M/1)^n$). The $M/M/n$ queueing model is already significantly more complex than the $M/M/1$ model. A natural question is then: how to actual measurements of performance tally with either of these models, given a Poisson traffic source?

4 Server Performance

For calibration, we begin by considering the response of a single server to increasing load, as a control. The graph in fig. 2 shows a rising CPU utilization on the left of the graph, and a response time which stays low until we reach approximately 100 requests/second.

Notice that the response time eventually levels out, as packets are simply dropped above the threshold. Above 110 requests/second the response time stays between 1200 and 1400 ms. From these results it is clear that the server can handle somewhere between 100 and 110 requests per second, and that this task is heavily CPU bound (this is a result of our experiment design, but we believe that it is representative for many application tiered web services).

We also see that the CPU utilization increases linearly until around 105 requests/second, then it drops. This is a telling result since the server should be heavily utilized above this rate if the system is working efficiently. Looking at the standard deviation of the results obtained for CPU load, we see that it is very high when the request rate is above 110 requests/second. This implies that the server is unable to deliver the maximum level of service when requests per second exceed the maximum threshold due to thrashing at the server side. This is a strong indication that we want to avoid this region at all costs.

The response time also flattens out when the load exceeds approximately 105 requests per second. Above this rate the server starts to refuse connections instead of accepting all of them. Data show that the error rate above 100 requests per second increases proportionally with the increasing request rate. This again indicates that the server can't process request at any higher rate.

The reason for the results shown in fig. 2 is the queue length configured on the Apache web-server. Apache only spawns a limited number of child processes to handle incoming requests, and when all these processes are busy it needs to queue up waiting requests. This queue has a maximum length, in our case defined by the Apache `ListenBacklog` directive. The errors start to increase when the server has filled up its queue; 120 requests/second gives an error rate of 5.91 packets, and when the request rate increases above this we see that the increase in errors is almost the same as the increase in requests per second.

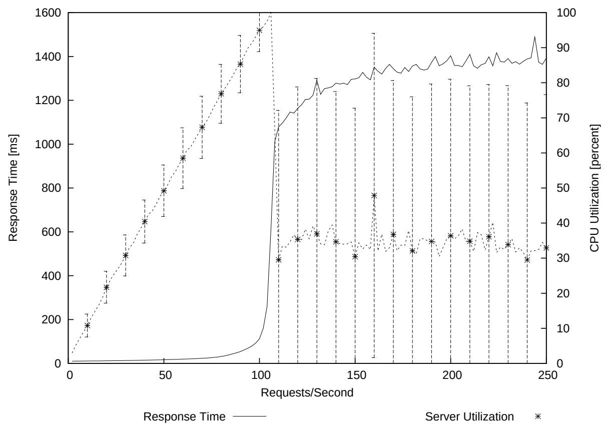

Fig. 2. *Response time and CPU utilization of a single server as a function of requests per second, using Poisson distribution with exponential inter-arrival times.*

5 Load sharing

Our load balancing arrangement uses a dispatcher at the bottleneck to distribute load to the back-end servers, using one of three different sharing algorithms:

– Round Robin: the classic load sharing method of taking each server in turn, without consideration of their current queue length or latency.
– Least Connections: the dispatcher maintains state over which back end server currently has fewest on-going TCP connections and channels new arrivals to the least connected host.
– Response Time: the dispatcher measures the response time of each server in the back end by regularly testing the time it takes to establish a connection. This has many tunable parameters.

In elementary queueing theory, there is no simple way of dealing with the issue of inhomogeneous servers. A simple question is therefore whether the simplest classical queueing models provide sensible predictive power for the data centre engineer in either a homogeneous or non-homogeneous case, on average.

6 Queueing Models as Sharing Predictors

We consider how queueing models perform compared to results obtained from our lab experiment. We use a c script to calculate response times using both $M/M/1^n$ and $M/M/n$ queues. From the previous sections we have calculated the performance of a single web-server, and we use this result in our queueing simulation. Below is a list of parameters used for expected response times using queueing theory:

- μ This is the variable we calculated when doing performance tests against one server. It is the processing capability of the server, also called the *service rate* and is often expressed in completions per millisecond. We take $\mu = 0.10413$from the fact that each server is able to process 0.10413 requests per millisecond.
- λ This is the *arrival rate*, and is expressed in arrivals per millisecond
- **n** Number of servers available. In our experiments we had at most 5 available servers to balance the load between.
  The *max_rate* specifies the limit at which we stopped simulation. This is usually equal to $n{\cdot}\mu$ since the queueing algorithms do not produce well-defined results when the servers are overloaded. In figure 3 max_rate equals $2 \cdot \mu = 0.20826$.

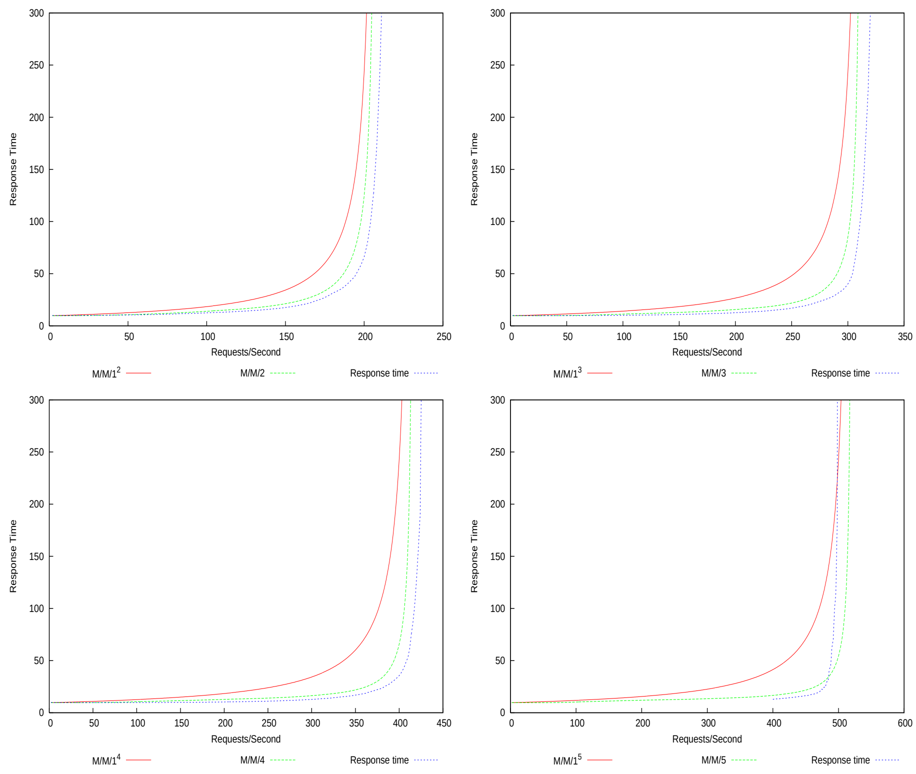

Fig. 3. *Theory vs. Experiment: This graph shows the results obtained from simulating load balancing with both $M/M/n$ and $(M/M/1)^n$ queues and response times from a real experiment.*

Figure 3 shows a comparison of queue model with experiment for 2-5 indentical servers, using the Round Robin algorithm to balance the load between two servers. We see that the experimental results for n servers follow those for the $M/M/n$ queue quite closely up to the point of about 90% saturation. This is an encouraging sign for theory – given that the servers truly are measurably identical. It is perhaps better than one would expect given the simplicity of the queueing model. However, there is an indication that the model worsens as the number of servers increases. The $(M/M/1)^n$ is overly pessimistic for all n at low traffic, but $M/M/n$ traces the actual behaviour with surprising accuracy.

At about 90% of the maximum expected capacity (as calibrated against the single server levels) the models begin to fail. A summary of requests per second, errors and network input/output in Table 1 reveals that errors first start to occur when the request rate exceeds this 88% mark. The actual 2,3,4-server solutions even out-perform the $M/M/2, 3, 4$ predictions across the whole spectrum of loads, even when entering the thrashing regime. This is likely due to the fact that each server in fact deals with requests using parallel processing threads, not a First Come First Served (FCFS) discipline as the queue model assumes[5]; this is more efficient than FCFS. Nonetheless, the closeness of behaviour is notable for two servers. As the total load rises in the 5 server case, this behaviour changes and the thrashing regime cuts in even more sharply. This could be a sign of a change in the performance of the load balancer itself.

No. of requests	Response Time	Std. Dev.	No. of errors
2	9.89	0.09	0
10	9.87	0.12	0
50	10.45	0.13	0
100	12.49	0.12	0
150	17.64	0.36	0
160	20.12	0.67	0
170	24.40	1.00	0
180	31.40	1.72	0
190	41.08	3.27	0
200	66.04	18.35	0
202	78.44	34.21	0.002
204	97.91	77.31	0.003
220	853.56	76.73	0.1

Table 1. *A summary of CPU utilization, request and error rates when using Poisson distributed inter-arrival times in a 2-server regime.*

Repeating the simulation for higher numbers of shows a similar story, but one which becomes progressively worse earlier. Figure 3 shows the comparison for $M/M/1^5$ and $M/M/5$. Here performance reaches approximately 490 out of the expected 520 requests per second (77%) before significant deviation sets in. Then at about 490 requests/second the experimental system reaches an instability and diverges drastically from the theoretical models. The fact that this occurs more quickly here is an anomaly; alas, we did not have more identical servers to attempt to extend the number to an even higher level, but this would have been interesting. Clearly, when theory and practice do not agree, some of the theoretical assumptions are incorrect. It is unlikely that the problem lies with the distributions. A clue as to the reason is found in the next section.

7 Scalability with increasing servers

If performance scales linearly when adding servers it means that we can easily predict the effect of adding more servers. This would be a useful property when considering QoS and maintaining SLA agreements. In figure 4 we show the addition of servers at four traffic levels intended to saturate the same number of servers.

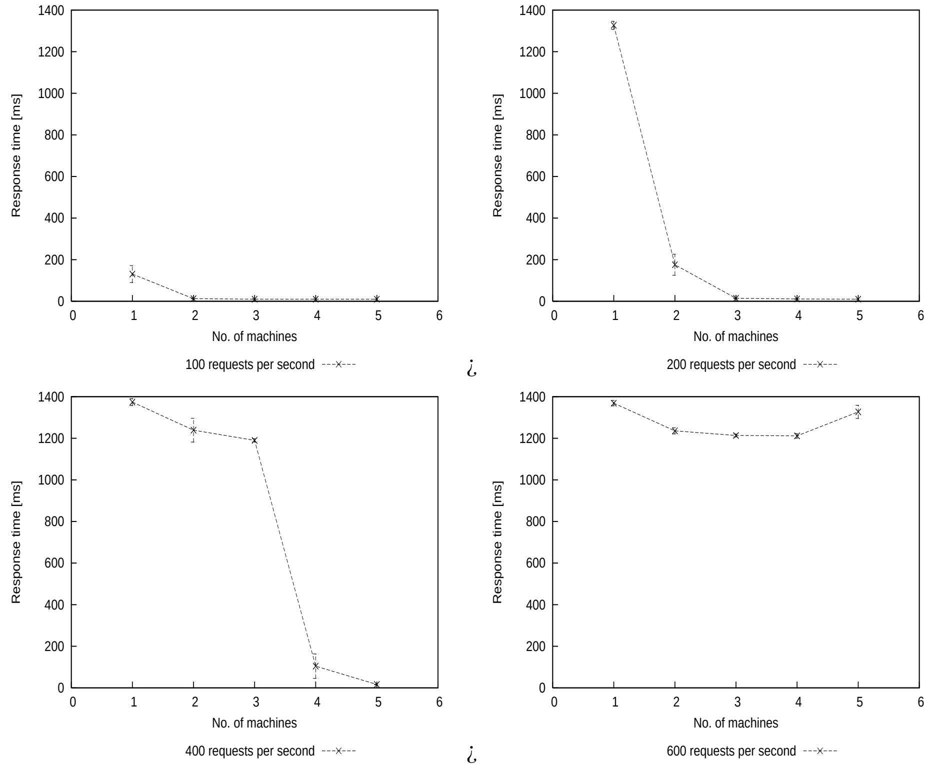

Fig. 4. *How response rates scale when adding extra servers. For low traffic, adding a new server results in a discontinuous improvement in response time. At very high load, adding a server seems to reduce performance, leading to the upward curve in the last figure as the limitations of the bottleneck dispatcher become apparent.*

In the first graph we see that the response time is kept low in all scenarios. This fits with the assumption that even a single server should be capable of handling 104 requests per second. The next graph shows that all scenarios except the one with a single server are capable of handling 200 requests per second. This fits the scaling predictions. With 400 requests per second, in the third graph, four servers just cut it. In the fourth graph we request 600 connection per second, and we see that the response rate is high for all scenarios. These results make approximate sense, yet the scaling is not exactly linear; there is some overhead when going from having 1 server to start load balancing between 2 or more servers. We explore this further below. Figure 5 shows how the rates scale with number of servers. Note the discontinuities in points of maximum curvative in the response functions after one server and four servers. the response is not exactly linear.

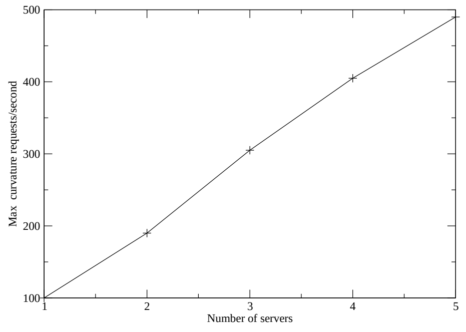

Fig. 5. Maximum processing capacity as a function of number of servers.

8 Load sharing performance

The measurements above have been made with the round robin sharing strategy. It makes sense that this would agree maximally well with the $M/M/n$ model when all of the servers are indentical. In any real data centre, the likelihood of having perfectly identical hardware is near zero given the rate at which technology advances. Equipment bought even a few weeks later could have dramatically different specifications.

To investigate the effects of inhomogeneities on the balancing, we manually adjusted the CPU speed on some of the blades to create a stepwise inhomogeneous environment with the following processor capacities: 1.4, 1.75, 2.1, 2.45, 2.8 GHz. The result is shown in figure 6 and 7.

Figure 6 shows results from using Least Connected (LC), Round Robin (RR), and Response Time (RT) algorithms to balance load between 5 servers. The algorithms perform approximately the same under low loads. The LC algorithm

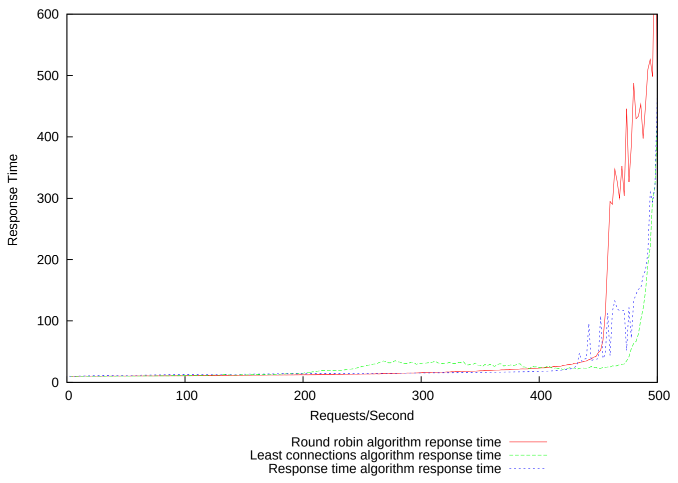

Fig. 6. *The performance of the RT, LC and RR algorithms in a homogenous server environment. RR is superior at low traffic intensity, LC survives to slightly higher levels, but a small instability around 250 req/s.*

performs well marginally longer than the others. The RT algorithm perform almost as well, but succumbs to noise before earlier.

For the inhomogeneous case, the results are more surprising. Here the Round Robin and Response Time strategies (based on actual measurements of the system performance) perform equally well. Both succumb to noise at about the same point. Remarkably the Least Connections approach (which uses state information directly in the dispatcher) survives the longest. We suspect that this is because, once the variance in the computing nodes blows up due to thrashing, it dominates the measurements of response times and they become so unreliable that the information make no difference and the performance of RT. RT is therefore no better than RR (random chance) in practice. The dominance of uncertainty matches the conclusions made in [5]. The fact that Least Connections works longer is probably a result of the greater reliability of the state kept by the dispatcher combined with the fact that the faster machines finish their equal tasks more quickly, thus maintaining an equilibrium. In real traffic, job lengths will be variable and we suspect that LC will not perform significantly better than random chance..

9 Conclusions

There is surprisingly little literature on dispatcher load balancing, in spite of the importance of this technology for business. Ref. [10] performs similar studies to ours and arrives at somewhat different conclusions. In [10, 11] they found the RR algorithm to be the worst under all conditions. Teo[10] also show results that indicate that RR, LC and RT converges at high loads. These results contradict

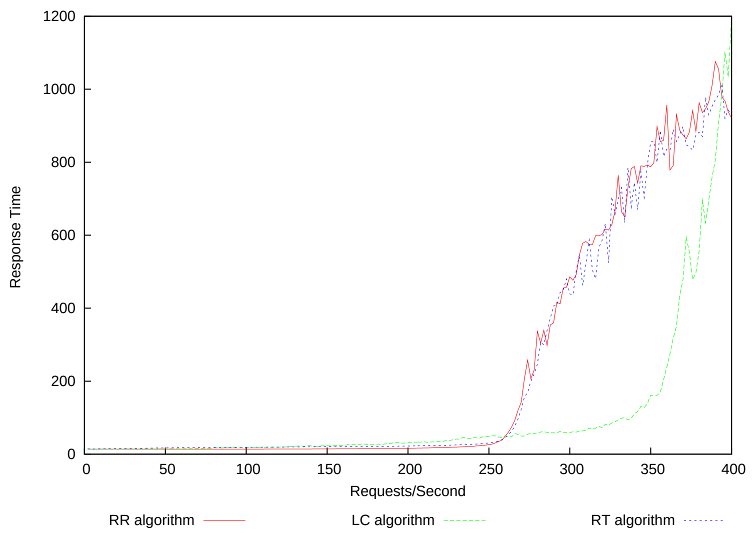

Fig. 7. *Performance of the RT; LC and RR algorithms in an inhomogeneous environment. Surprisingly both RR and RT measurements behave similarly under load, while RR is best at low loads. This indicates that there is no advantage to using RT. LC holds out longest here before saturating as it is able to utilize the extra capacity in the faster servers.*

our results and the ones found by Cardellini[12], as we found the RR algorithm to have best performance at low intensities, and that the LC algorithm could sustain higher traffic intensity than RR and RT. We suspect that the results are dependent on the nature of the test data downloaded. Our results have been easily controllable, and yet we see the effects of uncertainty start to dominate behaviour. In this regime, none of the expensive technology does much better than random chance.

In terms of scalability, we see that adding servers is not a smooth operation, in contrast with ref. [10]. Even at high traffic loads, modern servers have such power that they lead to very discontinuous changes in the performance. If one fixes the traffic level, the addition of a single server can have a large impact on the processing time, assuming that the trunk service is not a bottleneck. One can move from saturation to a comfort zone fairly easily. In this regard, there is no reason not to have extra capacity available in a server centre. Even a single extra server on standby can make a significant impact if one can predict the rise.

Our results indicate that there is little reason to use anything other than RR as a load sharing algorithm. This is a rather depressing conclusion. One would hope that monitoring and analysis would improve performance, but there is no strong evidence of this here.

What does the data centre engineer learn from this study? We believe that the key message of this work is that dispatcher capacity is the key in server load balancing. The sharing algorithm plays a lesser role than the ability of the dispatcher to cope with the connections. A simple Round Robin sharing works approximately as well the more sophisticated methods, but as the total load increases, the efficiency of the sharing appears to waver. We are not able to confirm this last conclusion in our experiment, but we think this point is worth further study. Once again, Service Level Objectives are far from predictable close to maximum capacity. We see here an operating level of 80% of maximum as being a reasonable over-capacity in production.

We are grateful to Jon Henrik Bjørnstad and Sven Ingebrigt Ulland for helpful discussion. This work is supported by the EC IST-EMANICS Network of Excellence (#26854)

References

1. British Standards Institute. *BS15000 IT Service Management*, 2002.
2. J. Sauvé et al. Sla design from a business perspective. In *IFIP/IEEE 16th international workshop on distributed systems operations and management (DSOM), in LNCS 3775.*
3. W. Xu, X. Zhu, S. Singhal, and Z. Wang. Predictive control for dynamic resource allocation in enterprise data centers. In *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006)*, pages 115–126. IEEE Press, 2006.
4. X. Li, L. Sha, and X. Zhu. Adaptive control of multi-tiered web applications using queueing predictor. In *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006)*, pages 106–114. IEEE Press, 2006.
5. J.H. Bjørnstad and M. Burgess. On the reliability of service level estimators in the data centre. In *Proc. 17th IFIP/IEEE Distributed Systems: Operations and Management (DSOM 2006)*, volume submitted. Springer, 2006.
6. D.A. Menascé and V.A.F. Almeida. *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning.* Prenctice Hall, 2000.
7. M. Burgess. *Analytical Network and System Administration — Managing Human-Computer Systems.* J. Wiley & Sons, Chichester, 2004.
8. G.B. Rodosek. Quality aspects in it service management. *IFIP/IEEE 13th International Workshop on Distributed Systems: Operations and Management (DSOM 2002)*, page 82, 2002.
9. A. Høyland and M. Rausand. *System Reliability Theory: Models and Statistical Methods.* J. Wiley & Sons, New York, 1994.
10. YM Teo and R Ayani. Comparison of load balancing strategies on cluster-based web servers. *Transactions of the Society for Modeling and Simulation*, 2001.
11. Paul Barford and Mark Crovella. Generating representative web workloads for network and server performance evaluation. In *SIGMETRICS '98/PERFORMANCE '98: Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 151–160, New York, NY, USA, 1998. ACM Press.
12. V Cardellini and M Colajanni. Dynamic load balancing on web-server systems. *Internet Computing IEEE*, 3(4):28–39, 1999.