Uncertainty in Global Application Services with Load Sharing Policy

Mark Burgess and Sven Ingebrigt Ulland

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

Abstract. With many organizations now employing multiple data centres around the world to share global traffic load, it is important to understand the effects of geographical distribution on service quality. The Domain Name Service is an important component for global load balancing. Using controllable simulations, we show that wide area sharing can play an important role in optimization of response times when traffic levels exceed that which can be supplied by a local infrastructure. We compute the probability of being able to meet Service Level Objectives as a function of DNS caching policy (Time To Live), so that service providers can account for DNS error margins in Service Level Agreements.

1 Introduction

Meeting Service Level Objectives (SLO) is a crucial goal for online businesses, especially in the application services sector. Even if no formal Service Level Agreement (SLA) has been made, Service Level Objectives (SLO) are set by clients' demands: users will typically defect from a slow web-site after only a few seconds of waiting in order to look for an alternative[1], hence performance is directly related to profit.

One of the benefits that networking offers is the ability to use distributed resources to one's advantage. When local resources fail to cope with demand, external resources can be brought into play simply by redirecting requests to another location. That location could be a few centimetres away, or several thousand kilometres across a wide area network. However, in wide area, globalized services there are more links in the chain between server and customer where uncertainties and delays can creep in, and each of these components becomes a focus of independent interest for the performance analyst.

The role of the Domain Name Service (DNS) has been of particular interest in the matter of load sharing and redirection of traffic. On the one hand, this is an obvious place to overload existing functions for the purpose of load balancing; on the other hand it is a fragile bottleneck in Internet services with low security properties and relatively poor performance. Ten years ago, network service capacities were orders of magnitude poorer than today so the delays incurred by DNS lookups was less noticeable. Today, the DNS appears as a key bottleneck which contributes a significant fraction to service round-trip times.

In this paper, we examine the predictability of wide area load-sharing, based on the Domain Name Service, and attempt to identify strategies for minimizing its impact on Service Level Objectives.

2 Global network services

Global load sharing is a subtle topic, because it makes cost trade-offs based on quite different currencies. Many layers are involved in directing a service across the globe, and each of these can add to the uncertainty in response time, and to the delay experience by the end user. One must therefore try to unravel the various components.

Why would we not simply centralize a service for easy management? The reasons for this include security, fail-over redundancy, traffic congestion management or even power saving (or cost saving under different tariffs)[2]. The inhomogeneity of load that occurs during the course of a day often makes night-time processing favourable in a data centre[3], thus it might be a reasonable strategy to divert traffic to a geographically remote location (on the other side of the planet) in order to balance a heavy day-time load – this would assume that the routing and transport cost were acceptable[4, 5]. In other work, we have looked at how local methods can be used to predict the scaling of application services in a data centre. The ability to respond to a request depends on both the nature of demand[6] and the availability of supply[7].

There is a need then to be able to predict the performance of wide area redirections for coping with server traffic, and study them in relation to the benefits of more local strategies. We approach the question of application service performance by asking a simple question: how does the use of wide area methods for network redirection affect application services levels? i.e. What level of uncertainty does globalized load balancing add to service response times?

There is a large literature on application service modelling. Much of it is now five to ten years old, and some is contradictory. We consider the situation today.

3 DNS

The Domain Name System (DNS) is the global name lookup service in the Internet. It is a distributed, hierarchical and redundant database running on many thousands of servers world-wide, each responsible for one or more DNS zones. When a client issues a request for a URL, the browser will first try to resolve the hostname in the URL into an IP address, so that it knows where to send the request. This is where it is possible for a DNS server to influence the outcome of a query, effectively directing the client to a desired or lightly loaded site. However, the DNS approach to load balancing is not without challenges, as will be discussed shortly.

DNS servers come in a variety of flavours all of which allow the service to act as multiplexers, serving different looked-up values for incoming requests in a one-to-many mapping. The approaches differ in their 'back-ends', i.e. the amount of

internal processing they use to adapt to their situation. We shall consider both static and adaptive 'back-ends' in this paper.

Consider a single DNS query, and assume that its return value has not previously been cached anywhere. The following sequence of events ensues:

1. A client program passes a partially qualified hostname to a system interface, (normally called `gethostbyname()`.
2. The operating system *qualifies* the name by adding a local domain name, if none was specified, and redirects the request to the local resolver (typically a local DNS nameserver), asking for the A-record (or AAAA-record in IPv6) for the fully-qualified name – an A-record is a standard hostname-to-IP mapping. The request to the resolver is *recursive*, which enables a flag meaning "I only want the final answer."
3. If the local resolver does not have the answer, it performs *iterative* queries through the DNS hierarchy. First it asks one of the thirteen root DNS servers. These know which servers control the top-level domains like com, org and net. The local resolver caches the response.
4. Further, the resolver asks one of the top level DNS servers for further directions to the domain. Again, it caches the response.
5. When the iteration process reaches one of the lookup domain's DNS servers, this server will know the answer to the query, and reply with an IP address, e.g. 192.0.34.166. Yet again, the resolver caches the response.
6. The response is sent back to the client operating system, which also caches the response.
7. The IP address is returned to the browser, which in turn can contact the server and retrieve the content. In addition to the operating system caching the response, many clients will do so as well.

Note how this rather simple example introduces at least five levels of caching, not counting potential intermediate http proxies.

Consider a query to a balanced web site. A typical response could be as in this example:

```
;; QUESTION SECTION:
;cnn.com.                    IN      A

;; ANSWER SECTION:
cnn.com.             300     IN      A    64.236.29.120
cnn.com.             300     IN      A    64.236.16.20
cnn.com.             300     IN      A    64.236.16.52
cnn.com.             300     IN      A    64.236.16.84
cnn.com.             300     IN      A    64.236.16.116
cnn.com.             300     IN      A    64.236.24.12
cnn.com.             300     IN      A    64.236.24.20
cnn.com.             300     IN      A    64.236.24.28

;; AUTHORITY SECTION:
cnn.com.             600     IN      NS   twdns-04.ns.aol.com.
```

The nameservers return multiple A-records for the 'cnn.com' hostname – this list is known as a Resource Record Set (RR set). The addresses appear to be within the same provider network. This does not mean however that they are geographically close to each other. Clients typically traverse the RR set sequentially, starting from the top. That is, if the first address on the list does not work, the client tries the next one, and so on. This is a decision that the client makes.

The numbers in the second column of the response show the Time-To-Live integer value (TTL) in seconds. This value governs how long the answer can reside in a cache in any of the intermediate nodes. Once this value has expired, the cache discards the value and the client must obtain an 'authoritative' answer by iterative querying once again. This is a traditional strategy for off-loading DNS servers and reduce lookup latency by caching the IP address value of a DNS lookup for the specified lifetime. The relevance of this mechanism must be reevaluated in light of performance improvements over the intervening decades.

As long as the TTL is greater than zero, queries will be answered from a cache instead of being redirected to other servers. In the example, the A-records have a TTL of 5 minutes. A low TTL ensures that DNS servers are queried often, and hence are given the possibility to obtain fresh, up-to-date information about a domain which is making changes over relatively small time intervals. Low TTLs mean frequent repetition of the arduous look-up process and hence come at the cost of adding a considerable delay to the total service response time.

For example, the DNS can consume a significant part of the time it takes to fetch a web page, which might have several in-lined objects, including pictures and advertisements from many source domains. Shaikh et al note: "25% of the name lookups (with no caching) add an overhead of more than 3 seconds for the ISP proxy log sites, and more than 650 ms for the popular sites. respectively. It is interesting to observe that nearly 15% of the popular sites required more than 5 seconds to contact the authoritative nameserver and resolve the name. This is likely to be related to the 5-second default request timeout in BIND-based resolvers" [9].

As we shall see, caching alleviates this problem considerably (indeed, the question of caching versus non-caching leads to a strongly bimodal behaviour), but with a different potential cost: global congestion.

From the above, we note that DNS can employ two basic mechanisms in multiplexing: it can cyclicly permute the RR set so that each new query is ordered differently. Since clients normally pick the first element from the list, this amounts to a continual Round-Robin shuffling of the server set. Second, the TTL value must be chosen to be a relatively low value to avoid too much re-use of old data which would bias the fair-weighted Round-Robin distribution.

Clearly, these methods are unreliable. There is much uncertainty. If we have three servers in the RR set and only every third query from a given client contains a service request, then all the traffic goes to the same server after all. We are also trusting clients to act predictably and not try to second guess the results

from the DNS server by performing their own shuffling: DNS implementations are not required to preserve the order of resource record sets. As for TTL values, multiple levels of caching make it a challenge to predict and control the actual TTL observed by the end user. The TTL policy is only a polite request to caches, not an enforcable mechanism.

A problem with low TTL values is in so-called *sticky sessions* in which a user is connected to a particular instance of a network service with a cookie of session identifier that is unique to one server (e.g. in a net bank or online retailer). If the next request to a persistent session were directed to a different server, the session would be lost. We shall not discuss this particular issue here, since its resolution is a story in its own right.

4 DNS latency

Service Level Objectives are performance wishes often set informally by clients and estimated by engineers. The service provider wants to guarantee that users will gain access to their services within a Service Level Objective. Users are known to give up on slow services within just a few seconds and take their custom elsewhere[1].

The uncertainty is the response time is rightfully a combination of the uncertainty margins in each of the independent causal factors between the client and the server. This includes the identification of the appropriate server through the DNS service. One would like to write:

$$\Delta t = \sqrt{(\Delta t_{\mathrm{DNS}})^2 + (\Delta t_{\mathrm{Routing}})^2 + \ldots} \tag{1}$$

Since the DNS is a network service, it is itself dependent on all of the other uncertainties in a network, so each DNS query invokes all of the latencies in the system even before a service has commenced. Alas, the inter-dependencies of a network make the Pythagorean formula above difficult to implement.

A data centre engineer would like to attempt to compensate for the lookup delay, or at least account for it in service promises to clients. Let us consider then, how much DNS server redirection could add to the margins needed for the 'over-provision' of services, according to this study? On a global scale there are more sources of possible delay which could add to the overall response time. It is plausible that the worst bottleneck would be shifted to some other component in the supply chain. Hence it seems possible that one might win performance improvements by making a more informed choice based upon monitoring of the available capacities along different alternative paths.

Consider the scenario in figure 1. We imagine a global organization with alternative data centres at different locations. Redirection to the appropriate data centre will occur by DNS multiplexing. There are two competing mechanisms in such a load sharing scheme: the desire to avoid bottlenecks at the dispatcher and the desire to avoid congestion at the servers themselves. If the dispatcher does not share efficiently, there might be congestion, but if the dispatcher struggles

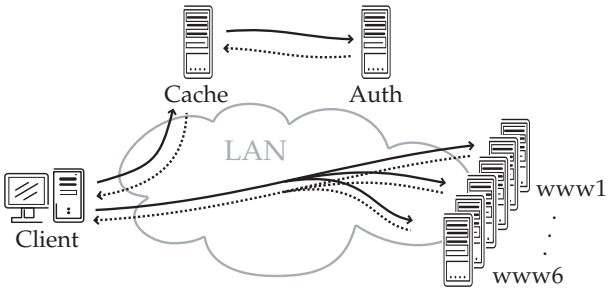


Fig. 1. A schematic illustration of the DNS load sharing scenario..

to share the load it could add to the service time itself. This is a classic problem in inventory management[10].

The DNS Time To Live is key here, since a high TTL lowers the load on the DNS as a dispatcher, but at the same time increases the congestion on the servers. It behaves as a slider-knob trading off these two effects. Figure 2 shows what one might expect for the relationship between the TTL value set in the authoritative nameserver and the round-trip time of fetching web-objects from a hostname that is registered with several IPs in the authoritative nameserver.

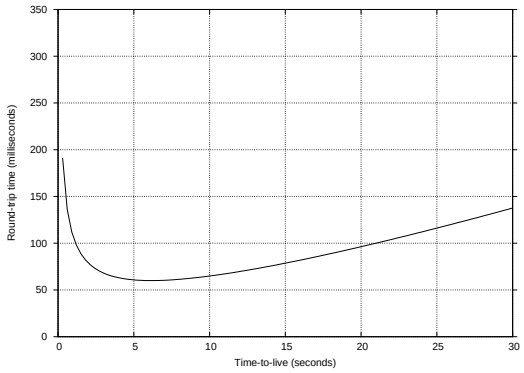


Fig. 2. Hypothesized cost of a DNS based load balancer at high traffic levels. The cost rises for low TTL as the load on the DNS server increases. The cost rises for long TTL as the individual servers start to become loaded inefficiently due to unfair weighting.

What is interesting about this form is the possibility that there exists an optimal value for the TTL parameter (the inventory re-order time). In previous work this TTL parameter has been set essentially by hand, without much insight into its functional role[8, 11]. We would like to investigate this causal role of

the TTL value more carefully below by testing DNS implementations against simulated traffic patterns.

5 Empirical study

Our experiments investigate the policies that can minimize DNS induced latency. Such latency can come from the inefficiency of the DNS service itself, from the relative congestion of alternative servers, and from transport uncertainties (which are unrelated to the DNS). We arrange for the latter to disappear by isolating a DNS load balancing scenario in the lab. Hence we are left with the interplay between dispatcher (the DNS response time) and server congestion (determined by the algorithm used by the dispatcher) which yields a final service time.

We deploy a client, the client's local resolver with cache (representing a local domain nameserver), and the remote domain's authoritative DNS server. We also have six web-servers, all configured similarly to answer requests for a single hostname, e.g. www.example.net. Wide-area network emulation is provided by the NetEm facility in the 2.6.16 linux kernel[13]. This is a part of the QoS framework there, and it allows us to specify delays and delay distributions for outgoing queues to simulate load. In the experiments on TTL vs RTT, the cache has a mean delay of 20ms, the authoritative has 300ms.

5.1 Effect of TTL on round-trip time, homogeneous servers

Running the `flood` tool to test DNS server response allows us to test our hypothetical inventory processing model for the combined lookup and service time. For the initial test, we make all of the servers identical in capacity and latency.

The plot in figure 3 shows the effect of TTL on static DNS server response-time for a full page load, that is, a DNS request for the hostname, TCP connection setup, sending HTTP request and finally receiving HTTP reply (connection tear-down is not included). The HTTP request used is trivial to serve and requires few CPU cycles per request.

For TTL 0, there is little variance in the results. We found that it was essentially impossible to overload a DNS server running on modern hardware with realistic traffic intensities. The uncertainty bars corresponds exactly to a controlled delay distribution which we specified for the authoritative nameserver (using NetEm). For TTL 1 and beyond, DNS requests are served both by the authoritative and caching nameserver. Since these two servers have very different delays set up for their outward queues, a request would either be served by the cache or the authoritative server. As TTL increases, the chance for a cache hit in the caching server increases proportionally. We see a flattening out of the tail for large TTL and no apparent rise in server congestion, since the load presented by our test page was low. Thus, this data represents primarily the behaviour of the dispatcher.

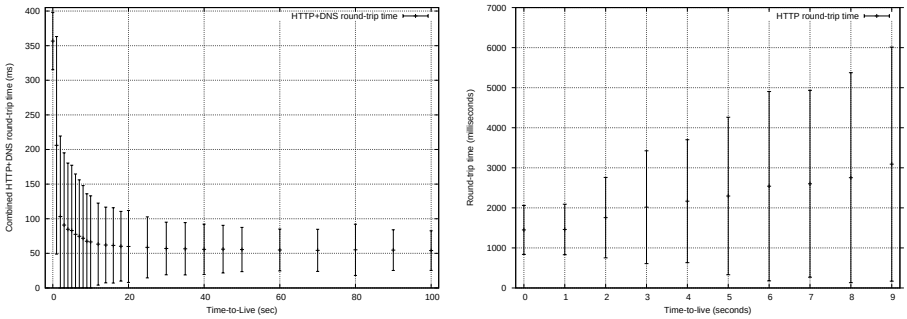


Fig. 3. Cost of lookup as a function of TTL for a 'static backend' DNS server. To be compared with the hypothesized form. Rather than rising at the end, it flattens out.. The second graph has no low TTL lookup cost up shows a rising inefficiency cost as server balancing fails..

Figure 4 shows how the DNS response time distribution is strongly bi-modal, clearly showing the influence of caching. The figure is dislocated so we can view the two peaks in more detail. We see that for TTL 0, there are no cache-hits on the caching server. But as TTL increases, so does the cache-hit rate. For TTL 100, we see a very high number of cached replies, and a close to zero amount of time-intensive authoritative requests.

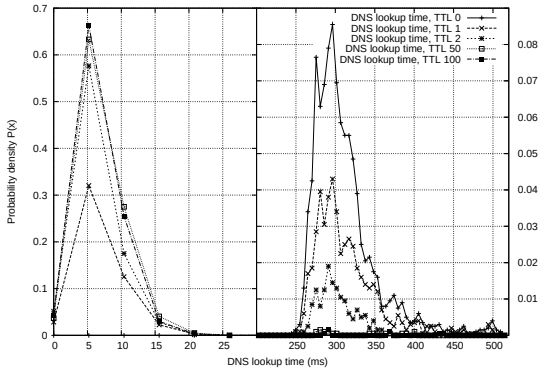


Fig. 4. The bimodal response time distribution for DNS showing the effect of caching..

The above experiment was repeated for a dynamic back-end DNS server: PowerDNS (with PostgreSQL as back-end). Here the database is loaded with the zone-data containing the resource records. PowerDNS is set up to query the database using a simple query, and appending `"ORDER BY random() LIMIT`

1", which returns one random IP address when requesting the hostname. The linearly increasing uncertainty in second figure 3 can be attributed to server loading due to poor entropy. Requests are queued and the system enters a thrashing phase. It increases with the TTL since a low TTL spreads the requests very efficiently among the servers, avoiding overload. As TTL increases, the probability of server load does too, and thus also the uncertainty. Thus we have measured both tails of our hypothetical curve for different traffic regimes.

5.2 Distribution entropy

The plot in figure 5 shows the cumulative frequency distribution of overall response times. What we observe is that for a zero TTL, most requests (90%) lie within the range of 0 to 1000 milliseconds per request. For TTL 9, we can see that approx 60% of the requests lie within the range of 0 to 2000 milliseconds per request. Also for TTL 9, we see that approx 90% of the requests lie within the range of 0 to 6000 milliseconds per request. Note that a long TTL seems to imply a longer wait, which warns against the very large TTL values suggested by authors several years ago. These results are of great interest to the Service Level Agreement architects. The efficiency of the load scheduler itself also de-

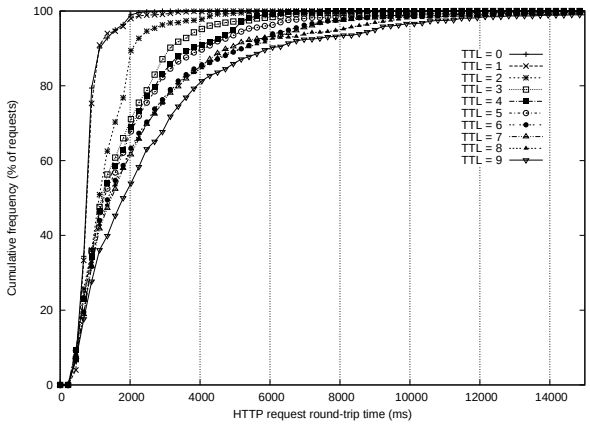


Fig. 5. Cumulative frequency plot of service times show with what certainty we can specify a response time as a function of TTL..

pends on the TTL. Ideally a load balancer will maximize the entropy of the connections histograms[14]. Figure 6 shows how well BIND distributes requests evenly among servers in a resource record set for a single client.

In the"Cyclic B" case, with a TTL of three seconds we see a more uneven response caused by BIND resetting the order of the resource record set each time the TTL expires. This causes a greater uncertainty which the simple round-robin

algorithm does not cope well with: an unfair weighting is induced on the server record distribution.

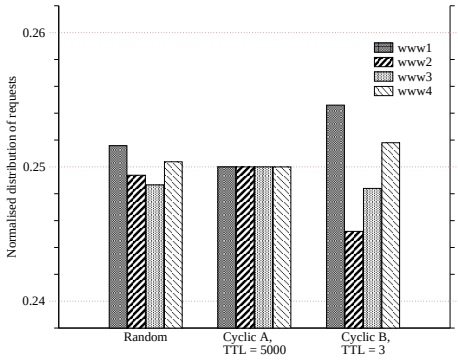


Fig. 6. Entropy of load balancing by DNS for one client..

The variations are very small, however, and do not pose any risk of over-utilisation for server 'www1' unless there is critically high traffic, in which case a non-linear instability could be seeded by this lack of parity. However, seen in the context of requests arriving from a source of many clients in diverse domains, there could be sufficient entropy of clients to even out this behaviour.

6 Comparable work

Several researchers have examined load balancing using DNS with varying conclusions. Cardellini et al survey proposed and commercially available balancing schemes, including constant and adaptive TTL schemes for DNS, dispatcher-based packet rewriting, and server-based mechanisms[8]. They find that both constant TTL with server and client state information, and the adaptive TTL scheme perform better than stateless round-robin approach.

Bryhni et al also compare a set of load-balancing implementations, with a focus on dispatcher-based systems[11]. The round-robin DNS is discussed with TTLs of 1 and 24 hours. Their trace-driven simulation results show that a dispatcher-based design running a round-robin algorithm yields the best distribution of load and amongst the lowest response times observed for that particular scenario.

In another paper Cardellini et al present a more thorough examination of web-server load balancing using DNS, and introduce HTTP redirection as a potential remedy for the otherwise coarse-grained nature of DNS[15]. They claim superior performance of redirection mechanisms over classic DNS-only balancing.

Shaikh et al, show that lowered TTL values must be carefully chosen to balance page responsiveness against excessive latency observed by the client[9].

The authors recognise that, to allow a fine-grained and responsive DNS-based server selection scheme, the TTL should be set to zero or a very low value, however this can lead to two orders of magnitude of extra delay, according to the paper. Other authors also explore the effectiveness of lowered TTL values. Jung et al [16], Teo [17], Park [18].

7 Discussion and Conclusions

DNS load balancing is a somewhat controversial topic. We have examined the behaviour of the DNS implementations with regard to their caching policy in order to find the expected uncertainty in meeting Service Level Objectives.

We find that today's DNS servers easily cope with high request volumes, in high levels (notwithstanding denial of service attacks). Caching policy does not impact directly on performance from the viewpoint of the server. However, the response time of the DNS service is relatively high by comparison to other services, due to the iterative nature of queries.

Round robin load balancing in DNS service works adequately with high levels of entropy, but are more likely to become unstable under high traffic conditions for low TTL. Low level load balancers favour simple round-robin load-sharing at low to medium intensity[7]; there one has very low latency routes between the dispatcher and server and the cost of looking for improvements outweighs any benefits. In global routes, a DNS server can benefit from a knowledge of the round-trip time when load balancing. Unfortunately, there is not a clear correlation between the time measured by the client, and that measured by the DNS server load balancer, so this does not work well.

The uncertainties inherent in wide area load sharing mean that a DNS load balancing strategy is not a substitute for low level load-sharing mechanisms. Failover redundancy is a main reason for having multiple data centres, but this is not the same as load balancing by DNS. Cumulative frequency plots indicates the additional round-trip time with corresponding uncertainties for requests. This shows us what one can expect to achieve in an agreement 80% or 90% of the time. Pre-sorting sites, e.g. by 'picking the site in your country' etc preempts DNS weaknesses.

DNS cannot be avoided, but is it the right tool for load balancing? Clearly it is not. However, it is nearly the only viable *interface* for load-balancing on a global scale (IPv4 anycast is another). DNS, with the solid backing of a dynamical back-end (based on a database with state-information gathered from the servers, maybe proximity information from ARIN's IP-to-country mappings, time zone info, etc), is a very powerful tool for global server load balancing.

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

References

1. 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*.
2. M. Burgess and F. Sandnes. A promise theory approach to collaborative power reduction in a pervasive computing environment. In *Springer Lecture Notes in Computer Science*, page to appear.
3. M. Burgess, H. Haugerud, T. Reitan, and S. Straumsnes. Measuring host normality. *ACM Transactions on Computing Systems*, 20:125–160, 2001.
4. B. Abrahao et al. Self-adaptive sla-driven capacity management for internet service. In *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006)*, pages 557–568. IEEE Press, 2006.
5. M. Burgess. Probabilistic anomaly detection in distributed computer networks. *Science of Computer Programming*, 60(1):1–26, 2006.
6. 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.
7. M. Burgess and G. Undheim. Predictable scaling behaviour in the data centre with multiple application servers. In *Proc. 17th IFIP/IEEE Distributed Systems: Operations and Management (DSOM 2006)*, volume submitted. Springer, 2006.
8. V Cardellini and M Colajanni. Dynamic load balancing on web-server systems. *Internet Computing IEEE*, 3(4):28–39, 1999.
9. Anees Shaikh, Renu Tewari, and Mukesh Agrawal. On the effectiveness of dns-based server selection. In *Proc. of IEEE INFOCOM 2001*, Anchorage, AK 2001.
10. H.L. Lee and S. Nahmias. *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, chapter Single Product, Single Location Models. Elsevier, 1993.
11. E Klovning H Bryhni and O Kure. A comparison of load balancing techniques for scalable web servers. 14(4):58–64, 2000.
12. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.
13. Stephen Hemminger. Network emulation with netem. In *LCA national Linux conference '05*, 2005.
14. M. Burgess. *Analytical Network and System Administration — Managing Human-Computer Systems*. J. Wiley & Sons, Chichester, 2004.
15. Valeria Cardellini, Michele Colajanni, and Philip S. Yu. Geographic load balancing for scalable distributed web systems. In *MASCOTS*, pages 20–27, 2000.
16. Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. Dns performance and the effectiveness of caching. *IEEE/ACM Trans. Netw.*, 10(5):589–603, 2002.
17. YM Teo and R Ayani. Comparison of load balancing strategies on cluster-based web servers. *Transactions of the Society for Modeling and Simulation*, 2001.
18. Kihong Park, Gitae Kim, and Mark Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *ICNP '96: Proceedings of the 1996 International Conference on Network Protocols (ICNP '96)*, page 171, Washington, DC, USA, 1996. IEEE Computer Society.