Towards Continuous Availability of Internet Services through Availability Domains

Nicholas Bowen[1] Daniel Sturman[1] Tina Ting Liu[2]

[1]IBM T.J. Watson Research Center
30 Saw Mill River Rd.
Hawthorne, NY 10532

[2]265 CSRL, MC 228
University of Illinois at Urbana-Champaign
1308 W. Main St.
Urbana, IL 61801

E-mail: {bowen|sturman}@us.ibm.com ting@crhc.uiuc.edu

Abstract

The increasing number of Internet users has caused a dramatic increase in electronic commerce. This growth is outpacing technologies for dependability causing traditional views of high availability to come under question. In particular, Internet failures are a phenomenon external to the owner of a commerce site that must be dealt with, and therefore, geographically distributed servers are a basic availability requirement for e-commerce sites. Geographic distribution provides an opportunity to view users in different roles based on those distributed components they must access. This paper presents an approach based on partitioning on-line function into domains, each of which provides service to users in a specific role. Coordination between domains is eliminated as much as possible by exploting application-specific knowledge. Once partitioned, availability techniques may be applied to each domain independently. We argue such an approach is necessary to deal with the geographic distribution of system components imposed by the nature of the Internet and maps well onto real e-commerce deployments.

1. Introduction

The Internet continues to grow at a rapid rate in terms of both the number of users and the number of Internet sites. With the increased number of users has come an increase in the amount of commerce being done over the Internet. The business need for electronic commerce (e-commerce) has been so compelling that its growth has outstripped the development of technologies for dependability. As a result, e-commerce has become a two edged sword, bringing a huge opportunity for reaching new customers but also creating a huge marketing liability in the event of outages which often become highly publicized.

Electronic commerce presents new challenges for high availability. Much of the network is out of the control of the individual business so the scope of any high availability solution must include these effects. The characteristics of the workloads are extremely non-uniform over time with many reports of peak load to average load ratios between 5:1 to 10:1. Workload variance is quite dependent on the nature of the business. For example, online financial services often experience a surge when the market opens or when a major event that impacts the market occurs. As people's home become more integrated into the Internet, there will be a strong correlation between effective advertisement and web traffic. Since availability is a function of the web server being able to handle these peak loads, businesses often purchase additional capacity above the peak loads.

The classical thinking of a single clustered server to provide high availability and scaling needs to be rethought. Geographic distribution has become a standard approach to Internet availability. For example, in the Nagano Olympics, IBM used geographically distributed servers to deal with these issues [4, 5]. Large software companies such as Netscape, Microsoft, and Sun employ extensive use of "mirrored" web sites [12, 3]. The use of multiple sites provides the ability for a business to avoid outages due to local Internet failures - either real failures or load induced outages that could be caused by high traffic to the web site in question.

However, companies are starting to provide electronic business functions such as sales, support, order processing, and supply-chain automation. These functions require coordination between disparate components, not all of which may be simply mirrored to provide geographic distribution. Traditional views of high availability designs and availability analysis come under question in this new environment. First, the complexity of the end-to-end solution make an overall availability model very difficult. The notion of a

0-7695-0707-7/00 $10.00

availability measure for the web server (e.g., 99.9% availability) does not necessarily provide the true availability for all end users. Further, given that a single server may be involved in both B2C (Business to Consumer such as on-line sales), B2B (Business to Business such as supply-chain automation), and internal operations (such as order processing) then the availability must be mapped to these different environments. There are several important observations to make. First, Internet failures and load induced failures mean that geographically distributed servers are a basic requirement. Second, we must delineate the availability discussion with respect to the roles of various user segments (e.g., the B2B and B2C end users). Third, after these roles have been defined, their particular behavior may be leveraged to remove dependencies and enable geographic distribution.

In this paper, we present an approach to Internet availability based on partitioning on-line function into domains each of which provides service to users in a specific role. Coordination between domains is eliminated as much as possible. Where coordination cannot be eliminated, we use application-specific knowledge to convert synchronous operations into asynchronous operations. Once this partition is performed, availability techniques may be applied to each domain independently, according to acceptable expense and business need.

2. Environment

The Internet provides a unique set of challenges for building a dependable system. Central to these challenges is the fact that there is no central administration of the Internet. Because any individual can only directly control a small portion of the total infrastructure, traditional end-to-end design and failure analysis techniques cannot be applied. For example, using N-way active replication techniques to ensure constant server availability provides limited additional dependability when the service provider contracted to provide network access does not provide a similar level of resilience.

It has been shown that network failures, on average, outnumber and tend to be of longer duration than server failures [9, 10]. Specifically, network related outages can account for over half of Internet server unavailability [9]. This problem is likely to be further exacerbated given the growth patterns of the Internet [6]. As a result, Internet applications must have the following characteristics:

- Geographic distribution: geographic distribution of Internet interfaces is essential not just for response time, but minimize the impact of regional network outages.
- Loose coupling between components: A consequence of geographic distribution is that high network latencies make traditional transactional consistency techniques impractical. Asynchronous communication, such as transactional messaging, is a preferred approach. Transactional messaging systems provide a transactional guarantee for senders that messages have been sent, and for receivers that messages have been received, but do not include the entire message flight as part of a transaction. Using a transactional messaging system, messages may be guaranteed to be delivered "exactly-once", but their delivery is asynchronous and may be delayed due to system failures en route.

  Where components cannot be geographically distributed, usually because of inherent synchronization, then other subsystems must be designed to operate as long as possible in the face of the components failure. Consequently, applications must be analyzed in terms of different roles such that, although one group may experience a failure, other groups may continue to operate. Thus, the system is never all-down, but may be down for users in particular roles in the organization.

3. System Design and Architecture

This section describes our proposed system design and architecture. First an example of the architecture is presented followed by a formal description of the architecture. We build off the observations about Internet services made in Section 2 and show, first with an example and then more generally, how highly available services may be built despite this environment.

3.1. Example

To illustrate our approach to factoring web applications, consider the example of an on-line store shown in Figure 1. The figure illustrates the traditional mechanism for deploying a scalable web application. A series of web servers handle requests from customers. The web servers process these order requests and modify two databases: one for order requests, and another tracking available inventory. Order handlers obtain lists of orders to be processed by accessing the databases directly.

Increased customer load is handled by increasing the number of web servers in the cluster (Database Management System (DBMS) capacity may also be scaled in a similar manner, but is not shown here). Increasing the number of servers also improves availability. However, as we will show in Section 4, Internet availability becomes the dominant factor in total system availability and, therefore, improving the availability of the web servers or DBMS alone has little impact on total system availability.

The monolithic nature of this system presents some limitations in terms of scaling and availability. In terms of scalability, static web requests can achieve a much higher raw

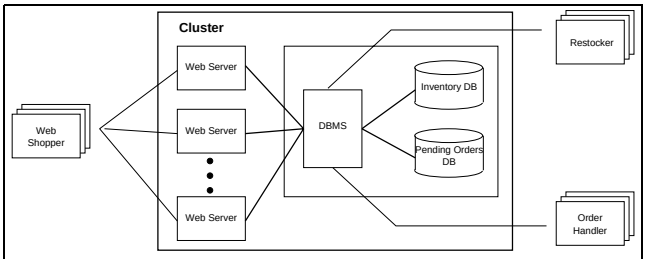

Figure 1. A traditional scalable web application deployed on a cluster.

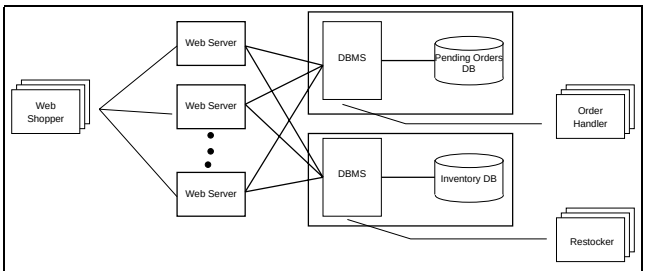

Figure 2. A *factored* version of the web application shown in Figure 1. Application characteristics are exploited to allow decoupling of components.

performance (e.g., units of work per second) that database oriented work. We calibrate this statement with some references to industry standard benchmarks. A Dell PowerEdge 6350 4-way SMP server has been benchmarked at 386 tpc-c transactions per second[1] [8]. The same hardware has been benchmarked at 13,000 SpecWeb requests per second. In this example, a comparable server is capable of performing 36 times more for static web requests than database transactions.

This difference in capability, coupled with the variability of workloads, creates significant challenges in scaling up the overall capability of web sites, i.e. the ratio between static web requests driven by browsers and DBMS activity driven by purchasing is nearly impossible to predict. This often leads to an application design where the database application is hosted on a different physical server than the web server. In this case, the application design uses a form of remote synchronous communications between the servers, *e.g.*, SQL's Distributed Relational Database Architecture (DRDA). The complexity of scaling this type of system often leads to a complicated system design where the availability of the web service is dependent on multiple physical servers and multiple software subsystems (e.g., web servers plus database servers). In an environment where geographic distribution is mandatory, these types of highly synchronous approaches become untenable.

Instead we propose the *factored* solution shown in Figure 2. Specific knowledge of the application allows us to geographically distribute components *without* reducing total system availability. In particular, we exploit the following characteristics of this application to convert synchronized interactions into asynchronous ones.

- The two database systems are used in different, non-overlapping, roles, and therefore maybe decoupled.
- Allowing some variability in inventory control allows the web servers to be decoupled from the inventory database. This may be done in one of two ways. Either small units of inventory may be periodically allocated to each web server or *business rules* may be derived through analysis of the business process that allow web servers to optimistically sell items much as airline flights may be overbooked today. In either case, the need for a transactional database operation is eliminated.

Asynchronous operations are carried out over the Internet using a transactional messaging system. Examples of such systems include IBM's MQSeries [2] and Microsoft's MSMQ [7]. Such systems guarantee the eventual delivery of a message despite transient network outages and node failures. Guaranteed messaging systems provide a programmer the simple semantics of "PUT a message on a named queue" or "GET a message from a named queue." Program control is returned to the programmer immediately even if there are failures in other parts of the system, including failures of the link, the remote system or the receiving application. These systems use database logging techniques to ensure these qualities of service. That is, if either the remote system or the link has failed when a program attempts to PUT a message to a remote application, the message will be transactionally logged on the local system. When the link recovers and the remote system reconnects, the message will be delivered. The message delivery protocol provides an "exactly-once" guarantee that the message does not get lost in transit.

The use of guaranteed message causes the programmer to restructure the application in a fairly dramatic manner as we will discuss in Section 3.2. However, once this is done the implementors have significant flexibility in the implementation of the system. For example, Figure 3 shows a system that has multiple front end systems and multiple back end systems. The front end structure provides increased capacity and allows the system to tolerate Internet

[1] 23,187.90 tpc-c/minute

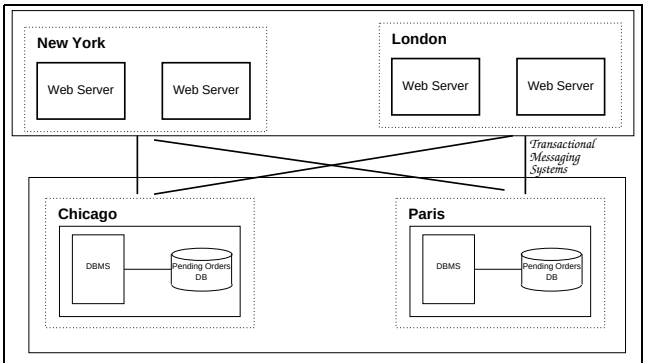

Figure 3. A web application partitioned into two availability domains. Each availability domain has a replicated sub-domain.

related outages. The multiple back end systems are for simple business reasons; that there is a requirement for a US based and European based distribution center. In addition to satisfying a business need, this back end structure could also be used to increase availability.

There are many web sites that are using geographic distribution of servers to achieve scalability. This work advances the state of the art by formalizing an architecture that

- Independently considers the availability needs of various user communities based on their roles. That is, the Internet users have different requirements than the accountants or the order handler. Many current systems view the availability of the system as the reference point while we argue that the system should be decomposed into independent units.
- The end-to-end system has faulty components. We claim that the quest to create a single highly available system is fruitless, the structure and dynamics of the Internet bring out new conditions such as load surges and network failures that require new thinking.
- Geographic distribution is a fundamental requirement for scale and availability.

3.2. System Architecture

We now propose a new architecture for highly available web sites that is based on several key principles.

- A system is partitioned into availability domains to serve the availability needs of each unique user community.
- One must independently consider the roles of various users groups.
- An information architecture that maps the aggregate data of the enterprise into availability domains.
- A strong reliance on application behavior allows a decoupling of domains and the use of guaranteed messaging technology to interconnect the availability domains in the information architecture.

Availability Domains are defined as a collection of resources (both computation and information) that are solely required to satisfy the availability requirements for a particular collection of roles. That is, availability domains are defined so that availability analysis for a particular role should be limited to the implementation of a single availability domain. The system architecture consists of a collection of availability domains that are interconnected through a persistent, transactional messaging system.

Information Architecture In situations where the information of the enterprise maps directly into the availability domains, the result is a collection of completely independent systems connected by a transactional messaging system. Unfortunately, most real life systems cannot be partitioned in such a manner.

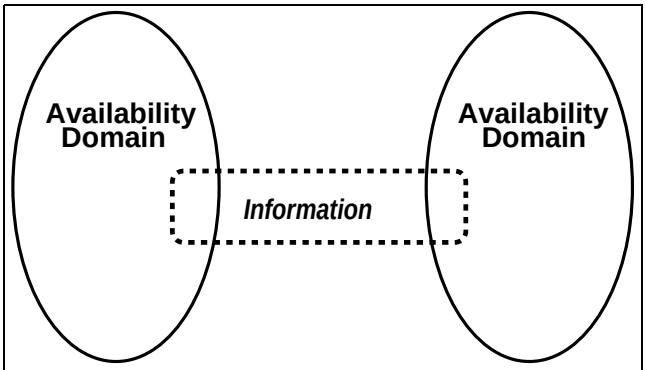

Figure 4. When information is shared across two availability domains, it creates an availability dependency that must be removed.

We define the Information Space as the total amount of data across all availability domains. An important issue is the mapping of the Information Space to availability domains. The easiest case is when the information can be partitioned to map completely inside an availability domain. For example, a particular set of static web pages is easily limited to a single availability domain: when two availability domains must share a common set of web pages, they

are easily copied and treated as two sets of resources. In the cases when a file system is shared by multiple availability domains, distributed file systems such as DFS [11] can be used. DFS provides a weak consistency guarantee of eventual consistency and therefore may scale over a campus or city. The complicated situation arises when operational data, such as that stored in a relational database, overlaps multiple availability domains as is shown in Figure 4.

In general, data with high consistency guarantees requires a greater amount of synchronization and, therefore, becomes more problematic when shared across availability domains. The dependencies of such synchronization result in availability dependencies across availability domains. In such cases, our methodology dictates that application-specific information be exploited to break the dependency, that is, convert a tightly synchronous interaction on into a loosely synchronous or asynchronous one. Synchronous remote access techniques violate our basic principles of no synchronous dependencies between availability domains, as discussed in Section 3.1.

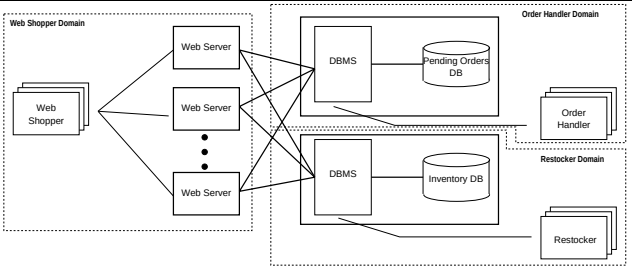

Figure 5. The *factored* version of the web application partitioned into three domains. The web shopper domain contains several independent web server sub-domains.

In Figure 5 we illustrate a business with three availability domains: webshopper, order handler, restocker. The webshopper role consists of the end users on the Internet. Order handlers process pending orders, bill the customer, and ship product. The restocker role periodically evaluates inventory and orders items from suppliers. The domains are designed to eliminate information dependencies between them to as great a degree as possible. For example, the webshopper is only dependent on the web cluster and failures in other availability domains will have no impact on the webshopper.

The other important aspect of defining availability domains is that each group has very different availability requirements. Although this is not a new observation, current monolithic systems are designed for the maximum of all availability requirements. In our example, the webshopper has the highest availability requirements with an objective of 24x7 dial tone availability. The order handler has much weaker availability requirements. In fact, we could envision that the order handler has a PDA with enough orders queued up that an outage of the main system for several hours would not have an impact. Role based availability requirements fundamentally improve the overall availability in a manner that would not have been possible building around a monolithic system structure.

Sub-Domains are defined as a building block component of an availability domain. These are independent units (they could be large scalable clusters) that can be easily added into an existing availability domain to provide additional capacity, increased availability, or geographical presence (e.g., in the case of a new distribution center). A subdomain can be the result of properly structuring the information architecture between the associated availability domains. In Figure 3, we show a case where the availability domain for the webshopper has been decomposed into two subdomains and placed in separate geographic locations. The design objective is to be able to achieve linear horizontal scaling when adding additional subdomains.

4. Analysis

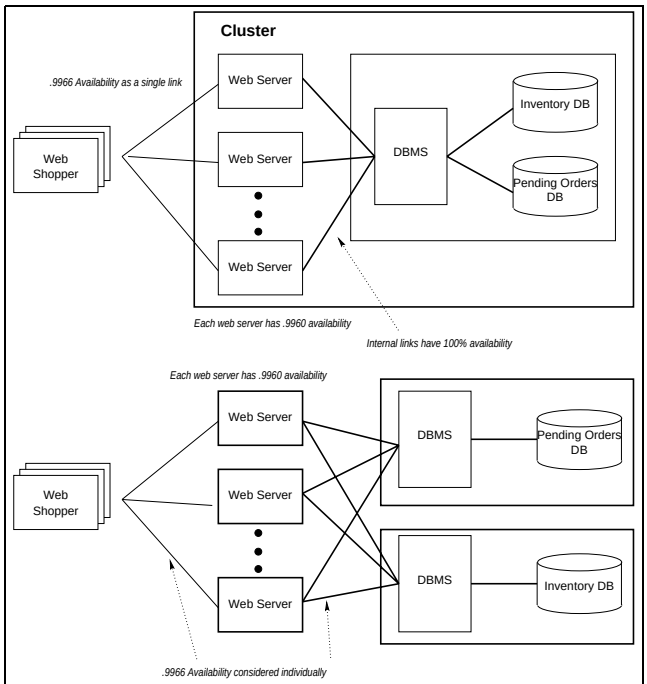

Figure 6. Parameters for analysis of the example shown in Section 3.1.

To demonstrate the effectiveness of our approach, we analyze the example discussed in Section 3.1. Two scenarios are modeled: web servers deployed in a centralized cluster, and web servers deployed in a geographically distributed configuration.

For these scenarios, we are primarily concerned with failures that cause web servers to be universally inaccessible to clients. That is, we are primarily concerned with a failure of the wide-area network, that portion of the network servicing our servers, or the failure of the servers themselves. We do not model the failure of those network resources servicing a particular client or group of clients. This assumption is justified in two ways *based on the nature of the webshopper role*. Our primary reason is that we assume a large enough number of clients that unavailability of any one client is not significant. Our secondary reason is that we are primarily concerned with presenting a high quality of service for *business* services. Consequently, client-end network failures are of less concern as they not only deny client's access to our servers, but to any competing servers.

We model the Internet connection to the cluster as a single network link, where the entire connection is either all up, or all down. The clustered servers are considered to be available when the network connection is available and at least one server in the cluster is available. For purposes of this analysis, we will ignore the availability of the DBMS (assume 100%). The availability of the cluster solution is shown in Equation 1.

$$A_{cluster} = A_{net}(1 - (1 - A_{server})^N) \quad (1)$$

Conversely, for the distributed case, the network connection to each distributed server is assumed to fail independently of all others. The distributed servers are considered to be available when there is at least one available server whose network connection is also available. The availability of the distributed solution is shown in Equation 2.

$$A_{dist} = 1 - (1 - A_{net} \times A_{server})^N \quad (2)$$

Values for 1 to 5 web servers in both scenarios are shown in Table 1 . We assume a network availability of 0.9966. Server availability was 0.9960. These values were derived from [9] by consolidating various classes of network failures and server failures.
The table illustrates that the limit of Equation 2 is 100% availability, but that the limit of Equation 1 is A_{net}.

The advantage of defining independent availability domains is more pronounced when you consider the availability interactions between the web servers and the DBMS. In the synchronous case, failure of the DB impacts the availability of a web server. This relationship is shown in Equation 3.

Servers	Cluster Availability	Distributed Availability
1	0.992614	0.992614
2	0.996584	0.999945
3	0.996600	0.999996
4	0.996600	0.999999997
5	0.996600	0.99999999998

Table 1. Sample values for the availability of the webshopper domain.

$$A_{cluster} = A_{net}A_{DBMS}(1 - (1 - A_{server})^N) \quad (3)$$

In the case of the factored solution, however, the DBMS availability may be assumed to be 100% since failures of the DBMS do not affect users in the webshopper role. Table 2 shows the reduced availability of the cluster solution for three web servers based on several values for DBMS availability:

DBMS Availability	Cluster Availability
0.9	0.89694
0.99	0.98663
0.999	0.99560
0.9999	0.99650
0.99999	0.99659

Table 2. Availability of the webshopper domain in a cluster based on DBMS availability.

The level of availability achievable through the distributed solution is only possible because the database was decoupled from the web servers, and this decoupling was possible only because application specific information was exploited. By using either business rules or pre-fetching of inventory (as discussed in Section 3.1), we are able to provide clients in the web shopper role with a highly available service.

We now examine the order handler role to see if we can improve its availability in a similar manner. For this role, things are more complicated: geographically distributing a database is fundamentally more difficult than the relatively stateless web servers. However, we can exploit the following facts:

1. We "own" the application for the order handlers. Consequently, we can impose more of the burden for decoupling on the order handlers than we did for the web shoppers.

2. There are fewer order handlers and each makes requests against the database less frequently. Thus, intermittent failures are much less of a problem than long duration failures.

Based on these observations, we propose an application-specific solution. A process at the order DB periodically pushes work out the various order handlers (via a transactional messaging system). Each order processor stores work assignments in a local database so that they are not lost due a crash failure. Order processing completion is sent back to the DB and the DB uses this information to measure each order processor's service rate. Enough work is advanced to each order processor to allow for failure and recovery of the DB. A balance must be struck between continued availability for the order processor (having enough work to do) and centralized control and load balancing order processing jobs to minimize order processing time. For example, each incoming order could be immediately farmed out to an order processor, thereby providing optimal resilience to DB failures. However, if an order processor crashed in this case, those jobs would be marooned until the order processor recovered. Availability for this role is measured as the ability for the order processors to keep working and, to a lesser degree, adequate response time to orders placed.

5. Conclusion

In this paper we have presented an approach to building highly available, scalable web services. This approach is built around the observation that, fundamentally, traditional availability techniques are poorly suited to these applications because the Internet cannot be made more reliable by any single application developer. Instead, developers must use geographic distribution to improve service availability.

Applications are divided into availability domains, each of which independently provides an available service to a particular role of participants using the Internet service. Synchronization between availability domains must be reduced to a minimum to avoid availability dependencies between domains. In some cases, this naturally falls out from the application, but in other cases application-specific knowledge must be exploited to reduce a tightly synchronous interaction to a loosely synchronous or asynchronous one.

This work is complementary to other approaches to building web services such as the WebOS [14]. WebOS provides operating systems services such as naming, persistent storage, and security to applications on wide-area networks. Having universal underlying services of this nature would simplify the implemenation of availability domains.

This work is an initial step towards Internet service availability and, as such, opens many questions. A better understanding and model of Internet availability would be particularly useful. Specifically, a study comparing failure patterns based on geography would be especially useful in evaluating our approach. Such a study would provide a better understanding of the performance implications of creating availability domains and using persistent messaging techniques for communication. This analysis becomes more complex when the trend to augment traditional point-to-point messaging with many-to-many communication [1] is taken into account.

Within this work we have illustrated our ideas with one, fairly typical, example. We prototyped the factored application described in this paper. The prototype uses IBM WebSphere and DB2, with asynchronous transaction messaging provided by MQSeries. In the future, we intend to use this system evaluate our ideas against a wider number of real-world deployments. Evaluation against upcoming practices is also of interest. For example, push-based information distribution is particularly promising for certain application areas [13]. However, Internet failures may be particularly damaging to such applications. The ability to make such services highly available will significantly impact the speed of their adoption.

References

[1] G. Banavar, T. Chandra, R. Strom, and D. Sturman. A case for message oriented middleware. In *Proceedings DISC '99*, 1999.

[2] B. Blakeley, H. Harris, and R. Lewis. *Messaging & Queuing Using the MQI*. McGraw-Hill Series on Computer Communications. McGraw Hill, New York, New York, 1995.

[3] V. Cardellini, M. Colajanni, and P. S. Yu. Dynamic load balancing on web-server systems. *IEEE Internet Computing*, 3(3):28–39, May-June 1999.

[4] J. Challenger, A. Iyengar, and P. Dantzig. A scalable and highly available system for serving dynamic data at frequently accessed web sites. In *Proceedings of ACM/IEEE Supercomputing '98 (SC98)*, Orlando, Florida, November 1998.

[5] J. Challenger, A. Iyengar, and P. Dantzig. A scalable system for consistently caching dynamic web data. In *Proceedings of IEEE INFOCOM'99*, New York, New York, March 1999.

[6] R. Govindan and A. Reddy. An analysis of internet inter-domain topology and route stability. In *Proceedings of INFOCOM '97*, volume 2, pages 850–857, 1997.

[7] A. Homer and D. Sussman. *Professional MTS and MSMQ With VB and ASP*. Wrox Press Ltd., 1998.

[8] I. International. Ideas top performers - tpc-c. http://www.ideasinternational.com/benchmark/tpc/tpcc.html.

[9] M. Kalyanakrishnan, R. K. Iyer, and J. U. Patel. Reliability of internet hosts: A case study from the end user's perspective. In *Proceedings of the International Conference on Computer Communications and Networks*, Las Vegas, Nevada, 1997.

[10] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental study of internet stability and backbone failures. In *Proceedings fo*

the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing (FTCS-29), pages 278–285, Madison, Wisconsin, June 1999.

[11] E. Levy and A. Siberschatz. Distributed file systems: Concepts and examples. *ACM Computing Surveys*, 22(4):321–374, December 1990.

[12] D. Mosedale, W. Foss, and R. McCool. Lessons learned administering netscape's internet site. *IEEE Internet Computing*, 1(2):28–35, Mar.-Apr. 1997.

[13] V. Technologies. ebusiness: Extending the enterprise. White Paper, 1999. http://www.vitria.com.

[14] A. Vahdat, T. Anderson, M. Dahlin, E. Belani, D. Culler, P. Eastham, and C. Yoshikawa. Webos: Operating system services for wide area applications. In *Proceedings of the Seventh IEEE Symposium on High Performance Distributed Computing*, July 1998.