Supporting Cluster-based Network Services on Functionally Symmetric Software Architecture*

Kai Shen
Dept. of Computer Science
University of Rochester
kshen@cs.rochester.edu

Lingkun Chu
Ask Jeeves Inc.
& University of California
at Santa Barbara
lkchu@cs.ucsb.edu

Tao Yang
Ask Jeeves Inc.
& University of California
at Santa Barbara
tyang@cs.ucsb.edu

ABSTRACT

Server and storage clustering has become a popular platform for hosting large-scale online services. Elements of the service clustering support are often constructed using centralized or hierarchical architectures, in order to meet performance and policy objectives desired by online applications. For instance, a central Executive node can be employed to make efficient resource management decisions based on a complete view of cluster-wide resource availability as well as request demands. Functionally symmetric software architecture can enhance the robustness of cluster-based network services due to its inherent absence of vulnerability points. However, such a design must satisfy performance requirements and policy objectives desired by online services.

This paper argues for the improved robustness of functionally symmetric architectures and presents the designs of two specific clustering support elements: energy-conserving server consolidation and service availability management. Our emulation and experimentation on a 117-server cluster show that the proposed designs do not significantly compromise the system performance and policy objectives compared with the centralized approaches.

Keywords

Cluster computing, peer-to-peer computing, network services, resource management, energy conservation

1. INTRODUCTION

The ubiquity of the Internet and various intranets has resulted in the widespread availability of on-demand services accessible through the network. Notable examples include document search engines [4, 17] and Web-based e-commerce applications [13]. In the scientific domain, online services ranging from numeric computation solvers [7] to DNA sequence pattern matching [6] have also enjoyed widespread usage. As global-scale service infrastructures (*e.g.*, Ninf [23] and OGSA [14]) become more mature, on-demand services are expected to continue gaining popularity in the future.

Due to its cost-effectiveness in achieving high availability and incremental scalability, server clustering has become the architecture of choice for hosting online services. This is especially the case when the system experiences high growth in service evolution and user demands. Despite its potential, there are significant challenges in providing service clustering support. Online services operate under certain performance and policy constraints. In particular, network clients seek services interactively and maintaining reasonable response time is imperative. As another example, maximizing resource utilization efficiency is an important performance goal for cluster-wide resource management [3, 26], along with other objectives such as quality-of-service support [34, 42] and energy conservation [8, 27].

Elements of the service clustering support are usually implemented using hierarchical architectures, where the complete system state is maintained at some central nodes. The availability of such complete system state is essential for making appropriate protocol decisions in order to achieve desired performance and policy objectives. The effectiveness of these approaches has been demonstrated under well-controlled environments, but customized enhancements (*e.g.*, hot-standby) are often needed to deal with abnormal situations, such as transient component failures, fluctuating service loads, and uncoordinated configuration changes.

Transient software and hardware failures occur frequently in large-scale distributed systems [25]. Previous studies have also shown that client request rates for Internet services tend to be bursty and to fluctuate dramatically over time, especially at the presence of extraordinary events [8, 10]. In addition, the capability of enhancing service features and adding servers without complex coordination is important for the manageability of large-scale service clusters. With the proliferation of fast growing network services, maintaining high robustness against these abnormal situations becomes increasingly important.

Earlier studies have demonstrated the feasibility of constructing highly robust distributed systems using symmetric ar-

*0-7695-2153-3/04 $20.00 ©2004 IEEE. This work was supported in part by the National Science Foundation grants CCF-0234346, CCR-0306473, and ITR/IIS-0312925.

chitectures with functionally equivalent software running at each node. These systems include distributed load balancing [5], serverless file system [2], and recent wide-area peer-to-peer networks [30, 32, 39]. However, it is not clear whether functionally symmetric software architectures can be employed in the construction of large-scale cluster-based network services for achieving high robustness. This is a challenging task because such a goal must be reached without compromising the performance and policy objectives desired by online services. In recognizing the diverse range of design objectives for different clustering support elements, we believe each clustering support element may require dedicated treatment. Our prior work has constructed effective cluster load balancing [35] and service differentiation support [34] using functionally symmetric architectures. This paper describes the designs of two additional clustering support elements: energy-conserving server consolidation and service availability management. We also examine their effectiveness in achieving respective performance and policy objectives.

The rest of this paper is organized as follows. Section 2 describes background information on cluster-based network services. Section 3 presents the proposed design techniques using functionally symmetric architectures. Section 4 describes our evaluation results based on emulation and experimentation on a 117-server cluster. Section 5 reviews related work and Section 6 concludes this paper.

2. BACKGROUND ON CLUSTER-BASED NETWORK SERVICES

Within a large-scale server cluster supporting online applications, internal service components are usually partitioned, replicated, and aggregated. *Partitioning* is introduced when the service processing requirement or data volume exceeds the capacity of a single server node. Service *replication* is commonly employed to improve the system availability and to provide load sharing. In addition, the service logic itself may be too complicated such that it needs to be partitioned into multiple service components. Partial results may need to be *aggregated* across multiple data partitions or multiple service components, and then delivered to external users.

Figure 1 illustrates such a clustering architecture for a document search engine [4, 17]. In this example, the service cluster delivers search services to end users and external partner sites through Web servers and XML gateways. Inside the cluster, the main search tasks are performed on a set of index servers and document servers, both partitioned and replicated. Each search query first arrives at one of the protocol gateways. Then some index servers are contacted to retrieve the identifications of top ranked Web pages matching the search query. Subsequently some document servers are mobilized to retrieve short descriptions of these pages and the final results are returned through the original protocol gateway. A large-scale service cluster typically consists of multiple groups of replicated service components. For instance, the replicas for partition 1 of the document servers in Figure 1 form one such group.

We also describe the clustering architecture for an on-demand computation service such as NetSolve [7]. A computational client submits the problem type (*e.g.*, BLAS [12, 20] or LA-

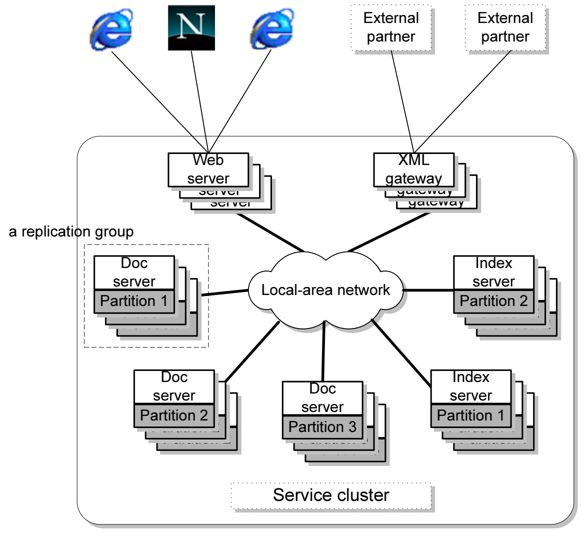

Figure 1: Document search engine.

PACK [1]), input data, and output method to a server that is equipped with the appropriate solver software. The server then performs the computation and responds to the client upon completion. Note that our work in this paper focuses on improving the robustness of the overall service architecture. We do not address the fault tolerance of individual program execution.

Large-scale service clusters may contain hundreds or thousands of nodes. Providing clustering support for large-scale services can be challenging. Below we examine some service clustering support elements as well as their performance and policy requirements.

Cluster-wide resource management. Allocating cluster-wide resources for serving online requests is an important element for service clustering support. An effective resource management scheme needs to deliver high *performance* and satisfy certain *policy* requirements. Some common objectives for cluster-wide resource management include maximizing system throughput, minimizing average service response time, reserving resources [3], conserving energy [8, 27], providing service differentiation [42], or a combination of some of these [34].

Service availability management. Finding the group of nodes hosting a requested service component on the desired data partition is a fundamental building block in service clustering support. This task is complicated by random node failures, uncoordinated service enhancements, and server additions. *Performance* is important because the fulfillment of each external request typically requires a number of internal service invocations, each of which triggers a service lookup inside the cluster. In addition, it is usually desirable that newly added services or cluster nodes become visible to all existing nodes within a short amount of time.

3. SERVICE CLUSTERING USING FUNCTIONALLY SYMMETRIC SOFTWARE ARCHITECTURE

Functionally symmetric systems are distributed systems without any centralized control or hierarchical organization, and with functionally equivalent software running at each node. Such an architecture is inherently free of scaling bottlenecks. They also exhibit strong robustness in the face of random failures and even intentional attacks because a failure of one node is no more disastrous than the failure of any other. In addition to scalability and robustness against failures, functionally symmetric systems can be more agile in response to load fluctuations or other system state changes because adjustments could be made wherever situations arise without consulting with authorities [34].

Using functionally symmetric architectures to improve the distributed system robustness is not new. Similar ideas were used in systems such as the MOSIX distributed operating system [5] and the Serverless File System (xFS) [2]. Those earlier systems were deployed at relatively small scale (*e.g.*, several dozens nodes). Recent peer-to-peer work examined the construction of large-scale wide-area distributed systems [30, 32, 39]. In addition to employing functionally symmetric software architectures, the scalability of these systems is further supported through controlling various management overhead when the system size grows. Linearly increasing per-node management cost could result in excessive overhead when the system scales to a large number of nodes. Many peer-to-peer systems manage to limit various management overhead to logarithmic increases, including object lookup latency, routing table storage requirement, and system repair overhead in response to failures [28, 30, 32, 39]. We call this *slow-scaling overhead.*

Functionally symmetric architectures can similarly enhance the robustness of cluster-based network services. However, such a design must not compromise the unique performance and policy requirements desired by online network services. This is a challenging task because unlike hierarchical counterparts, protocol decisions in these systems are often reached collectively at all cluster nodes, each without complete knowledge of the global system state. In addition to robustness, scalability remains an important goal in our context. Since cluster-based network services typically grow to relatively modest sizes compared with global-scale distributed systems (1,000s *vs.* 100,000s), careful examinations are needed to assess the extent of scalability requirement. In particular, linear-scaling overhead can be tolerable when the unit of such overhead is small. This is especially the case considering the high performance of modern system area networks.

In the rest of this section, we illustrate the improved robustness using functionally symmetric software architecture (Section 3.1). We then describe our design of two specific clustering support elements using functionally symmetric architectures: energy-conserving server consolidation (Section 3.2) and service availability management (Section 3.3).

3.1 The Potential of Improved Robustness

We illustrate how functionally symmetric software architecture can improve the system robustness. Although more sophisticated techniques exist to model and measure system availability under component failures [22], a simple model is sufficient for our purpose. We consider an n-node service cluster. We assume *independent* node failures. Let the node mean time to failure be MTTF and the node mean time to repair be MTTR. For systems employing a central manager node, a failure of the central node renders the entire system unavailable. Therefore the overall system availability rate is $1 - \frac{\mathrm{MTTR}}{\mathrm{MTTF}+\mathrm{MTTR}}$. A system with a hot-standby for the central manager node can tolerate single node failure but may not survive concurrent node failures. Its system-wide availability is $1-(\frac{\mathrm{MTTR}}{\mathrm{MTTF}+\mathrm{MTTR}})^2$. For systems using functionally symmetric architecture, node failures only affect service capacity; they do not have direct impact on the overall system availability. Assume a small proportion of capacity reduction (denoted by ρ) can be tolerated, then the overall system availability rate is the proportion of the time when at most $\rho \cdot n$ nodes fail concurrently: $1 - (\frac{\mathrm{MTTR}}{\mathrm{MTTF}+\mathrm{MTTR}})^{\rho \cdot n}$.

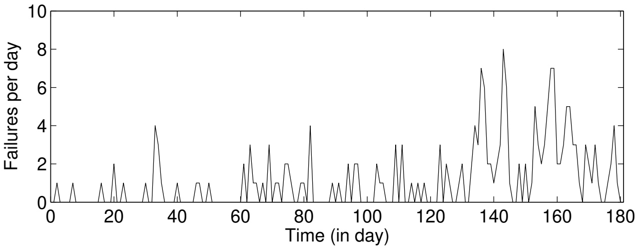

Figure 2: Daily failure count trace (over a period of six month) at a 300-node operational service cluster.

Below we attempt to quantify the system availability rates at a particular real-world setting. Our quantification is based on the daily failure count trace (over a period of six months) at an operational service cluster with about 300 nodes (shown in Figure 2). Only failures that cannot be repaired by automatic reboot (*e.g.*, hardware failure, disk overflow) are counted in this trace. The node MTTF is around 275 days for this operational period. We do not have accurate record for the failure repair time. We estimate the node MTTR as three hours based on the experience of the cluster management team. Based on the previous analysis, we can quantify the system *unavailability* rates under the three approaches: (1) 4.5×10^{-4} for centralized architecture; (2) 2.0×10^{-7} for centralized approach with a hot-standby; and (3) 8.3×10^{-21} for functionally symmetric architecture when $\rho = 2\%$. It is not hard to see that systems using functionally symmetric architecture are much more robust and such advantage would become more substantial for larger systems.

We should point out that the failures of the central node and the standby node may be positively correlated to each other. In other words, the failure of one of them may make it more likely for the other to fail also. This is because (1) they run the same software so they are subject to the same type of software-triggered failures; and (2) the failure of one increases service load on the other when load sharing is employed. Positively correlated failures reduce the effectiveness of hot-standby. Although the system availability may be improved by adding more standby nodes, this may require significantly more complex software, especially when strong consistency among standby nodes is required.

3.2 Energy-Conserving Server Consolidation

The richness of research issues for cluster-wide resource management lies in the wide range of resource management objectives online services may desire. A large body of previous work addressed various issues in request distribution and resource management for cluster-based server systems, focusing on quality-of-service support [3, 42], energy conservation [8, 27], or locality-aware distribution [26]. In order to achieve the desired performance and policy objectives, most of these studies relied on centralized or hierarchical architectures to manage cluster-wide resources in deterministic manners. For instance, a central *Executive* node can be employed to make efficient resource management decisions based on a complete view of cluster-wide resource availability as well as request demands (shown in Figure 3(A)).

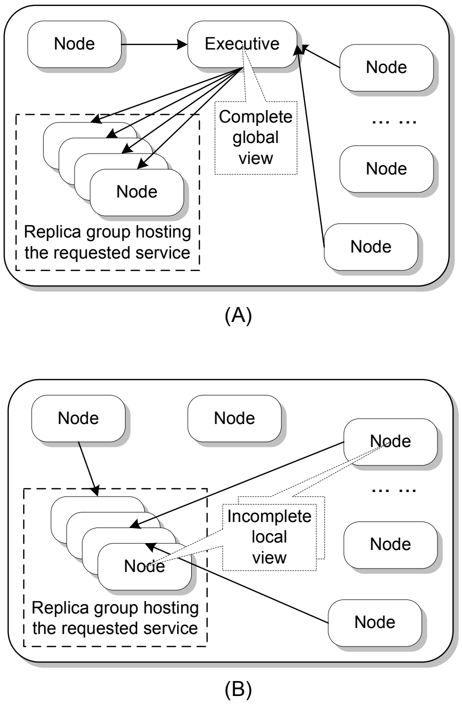

Figure 3: Resource management by a central Executive node (A) or using a functionally symmetric architecture (B).

As we argued earlier, functionally-symmetric architectures can achieve high robustness for large-scale systems. Under such an architecture (shown in Figure 3(B)), nodes make request direction and other resource management decisions based on a local-view of the system state, which is often incomplete. The main question for decentralized resource management is whether it can achieve performance and policy objectives desired by online applications. Our prior work has constructed effective cluster load balancing [35] and service differentiation support [34] using functionally symmetric architecture. In this paper, we describe resource management techniques for achieving energy-conserving server consolidation.

Client request rates for network services tend to be bursty and fluctuate over time. For example, the daily peak-to-average load ratio at Internet search service Ask Jeeves [4] is typically 3:1 and it can be much higher and unpredictable in the presence of extraordinary events. Figure 4 shows the total and non-cached search rate of a one-week trace we collected at Ask Jeeves search [4] via one of its edge Web servers. Note that this trace only represents a fraction of the complete Ask Jeeves traffic during the trace collection period.

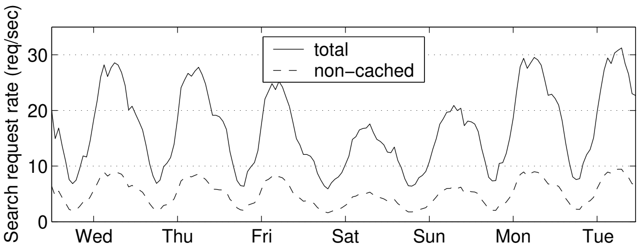

Figure 4: Search requests to Ask Jeeves search via one of its edge web servers (October 1-7, 2003).

In light of the load burstiness at large-scale service sites, it might be profitable to consolidate services to a handful of servers at load troughs while the remaining servers can be shut down to conserve energy consumption. Previous studies have addressed issues in energy management for Internet hosting centers [8, 27]. These systems rely on a centralized *Executive* node to monitor cluster-wide resource availability and request demands. An optimization problem is then solved at the Executive to decide how many servers should be powered up for servicing requests.

We describe a server consolidation scheme using a functionally-symmetric architecture. In this approach, each node dynamically maintains an up-to-date load assessment of itself, including the recent service response time and the request queue length. The request queue length is defined as the total number of currently active and queued service requests.

- For the response time, each node records the response time for requests completed in a recent observation window. The 95 percentile value is then used as the load assessment.
- For the request queue length, each node periodically checks its own load index. The stable load assessment is then maintained using an augmented exponentially-weighted moving average. More specifically, let s_t be the sampled load at t-th measurement. Then the load assessment after t-th measurement (denoted by l_t) can be calculated as:

$$l_t = \begin{cases} \alpha l_{t-1} + (1-\alpha)s_t & \text{if } s_t > l_{t-1}, \\ \beta l_{t-1} + (1-\beta)s_t & \text{otherwise.} \end{cases} \quad (1)$$
$$(\text{where } 0 < \alpha < \beta < 1)$$

The selection of the decaying factors α and β should be careful to maintain the smooth and stable reaction for both short-term load spikes and long-term load changes. Additionally, we let $\alpha < \beta$ so that the system can react quickly when load increases. In other words, Begin loop

1. Wait for a random delay between $[(1\text{-}\delta)\mathcal{P},\ (1+\delta)\mathcal{P}]$.
2. Purge expired information from the local view of the cluster-wide service availability (the expiration threshold must be larger than $\tau \cdot \mathcal{P}$).
3. Check whether any service availability broadcast has been received from other nodes in the past $\tau \cdot \mathcal{P}$.
   (a) If so, send the availability information of *locally-hosted* services to the source of the latest broadcast.
   (b) Otherwise, broadcast all known cluster-wide service availability information to other nodes in the cluster.

End loop

Figure 5: *Suppressive broadcast* for service availability management.

service performance has a higher priority than energy conservation.

A node makes a broadcast to all other nodes in its replication group and starts a power-down sequence when both the service response time assessment and the request queue length assessment crosses a low-load threshold for a certain period of time. We use a random-length grace period between 1-δ and 1+δ times the mean value. A random-length period is important to reduce the chance of simultaneous power-downs. To further eliminate such possibility, the node also waits for a short period after it announces its power-down intent to check if other replicas are making simultaneous attempts. If so, the power-down action is canceled and a new power-down attempt may be made after another random-length grace period if the local load assessment stays below the low-load threshold. The chance of consecutive collisions is very small after several attempts.

On the other hand, a live node will bring alive some dormant nodes when either one of its two load assessment indexes exceeds a high-load threshold for a certain period of time. We again use a random-length grace period to prevent all live nodes restarting dormant servers simultaneously. The Advanced Power Management (APM) tools for Intel-based systems allow a server to be remotely started at about one minute delay. Note that we are more cautious in the power-down process because it is our first priority to maintain acceptable response time for online services.

3.3 Service Availability Management

Service location lookup finds the group of nodes hosting a requested service on the desired data partition. This task is a fundamental building block in service clustering support. It is often complicated by random node failures, and by uncoordinated service enhancements and server additions. The performance of service lookup is important because the fulfillment of each external request typically requires a number of internal service invocations, each of which triggers a service lookup inside the cluster. With the common requirement of serving each online request within a fraction of a second, each service lookup must be completed within a few milliseconds or less. Under such a context, it is desirable to maintain an up-to-date local view of service availability at each node such that service lookups can be completed locally.

Membership management for asynchronous distributed systems has been extensively studied in distributed computing research [9, 21, 24, 31]. Protocols proposed in these studies are designed to satisfy strict membership management specifications for general distributed systems. These solutions tend to be complex in protocol handshakes when dealing with concurrent asynchronous events. In comparison, "soft state"-based system state maintenance[1] has been favored in many recent distributed systems [16, 36] owing to its simplicity and its ability to achieve high robustness in the face of node failures. Such techniques work especially well in cluster environments due to the reliability and high performance of system-area networks. Our discussion in this paper focuses on "soft state"-based membership management.

The first approach we consider employs a dedicated central node which maintains the complete cluster-wide service availability information as soft state. More specifically, each cluster node sends periodic refreshing messages containing local service availability information to the central node. Such information expires on the central node as soon as a certain number (called *expiration threshold*) of consecutive periodic messages are not received. The frequency of the periodic refreshing messages and the expiration threshold are chosen such that a failing or departing node can be removed from all live nodes' local view soon enough. In order to allow service lookups be performed locally, the central node periodically broadcasts the complete cluster-wide service availability information to all nodes. Although the dedicated central node receives a large number of messages in each period, the protocol overhead at a service node is very small (*i.e.*, the transmission of a periodic message and the receipt of another).

Our second approach, called *all-to-all broadcast*, employs a functionally symmetric architecture. In this approach, each node periodically broadcasts the local service availability information to all other nodes. This information is maintained at each node as soft state such that it has to be refreshed

[1] An operational definition of *soft state*: a source of soft state transmits periodic "refresh messages" to one or more receivers that maintain a copy of that state, which *expires* if the periodic messages cease.

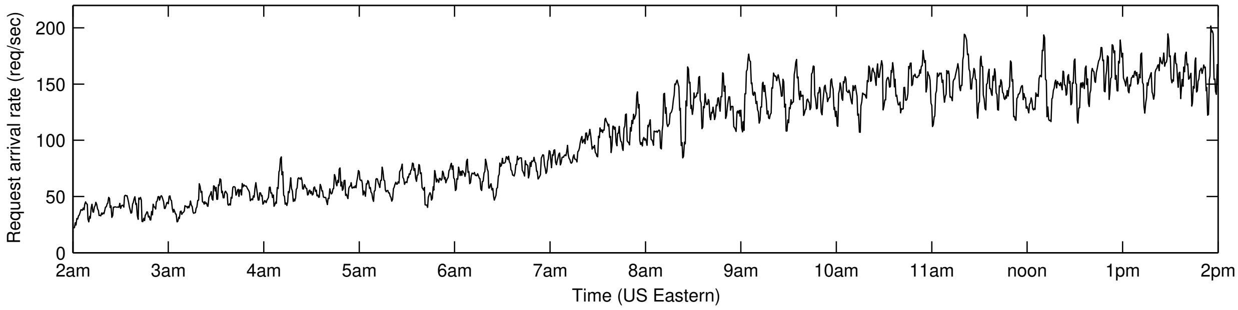

Figure 6: Request arrival rate for a 12-hour period with increasing service demands.

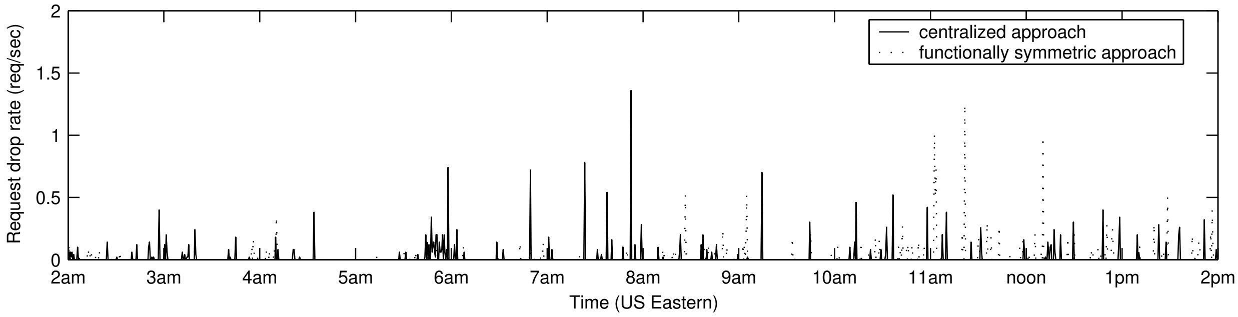

Figure 7: Comparison on the request drop rate for a 12-hour period with increasing service demands.

repeatedly to stay alive. To locate a service in the cluster, a node can simply iterate through local copies of service availability broadcasts received from other nodes in the cluster. At each service node, the bandwidth consumption and processing overhead for service broadcasts scale up linearly when the cluster size increases. Such linear-scaling overhead, although tolerable (quantitatively measured in Section 4.2), still represents a sizable performance penalty for large-scale network services.

In order to control the protocol overhead, we introduce another approach for service availability management called *suppressive broadcast* (illustrated in Figure 5). Like all-to-all broadcast, all nodes in this approach run functionally equivalent software. However, it differs from all-to-all broadcast in the following ways. First, each service broadcast contains the complete cluster-wide service availability information that the sender knows. Second, a node suppresses its own broadcast if it has received a service broadcast from another cluster node (called node X) in the past τ periods. If its own broadcast is suppressed, the node sends periodic messages containing local service availability information to node X. As long as the broadcaster (node X) is alive, the suppression scheme ensures that only a single service broadcast is sent out during each period in the entire cluster. When node X fails, another node in the cluster will take over after it detects the absence of τ consecutive service availability broadcasts. To reduce the chance of multiple nodes trying to take over simultaneously, each node employs a non-fixed period length which is set to be evenly distributed between (1-δ)$\mathcal{P}$ and (1+δ)$\mathcal{P}$, where $\mathcal{P}$ is the average period length. However, this approach does not eliminate the possibility of a simultaneous takeover. When that occurs, all nodes that made the attempts will be suppressed by each other's broadcasts. New takeover attempts will be made again until one node emerges as the broadcaster without colliding with other nodes' attempts. Note that the chance of consecutive collisions is very low after several attempts.

Since the broadcaster receives a large number of messages in each period, it consumes more bandwidth consumption and processing overhead compared with a regular service node. For load balancing, the broadcaster can voluntarily relinquish the role after a certain number of periods. It can do so by simply ceasing the service availability broadcast.

4. EXPERIMENTAL EVALUATION

We have made a prototype implementation of our proposed designs of energy-conserving server consolidation and service availability management. In this section, we show evaluation results on the effectiveness of our designs in meeting desired performance and policy objectives. We use different evaluation methodologies for the two clustering support elements. Due to the simplicity of the service availability management, our evaluation is based on a real-time emulation which allows us to assess large-scale systems that exceed the limitation of our experimental platform. For server consolidation, an effective evaluation requires the integration with a real online service application. For the purpose of evaluation, we incorporated our proposed design into the Neptune clustering middleware system [36] and we experimented with an online index searching application on a 117-server cluster. We believe the choice of a particular clustering middleware in this experimentation does not affect the applicability of our proposed design techniques.

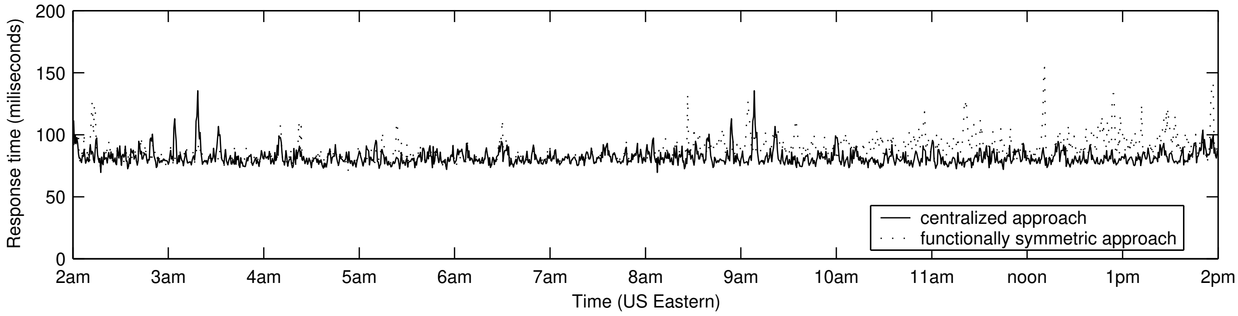

Figure 8: Comparison on the average service response time (for 30-second intervals) for a 12-hour period with increasing service demands.

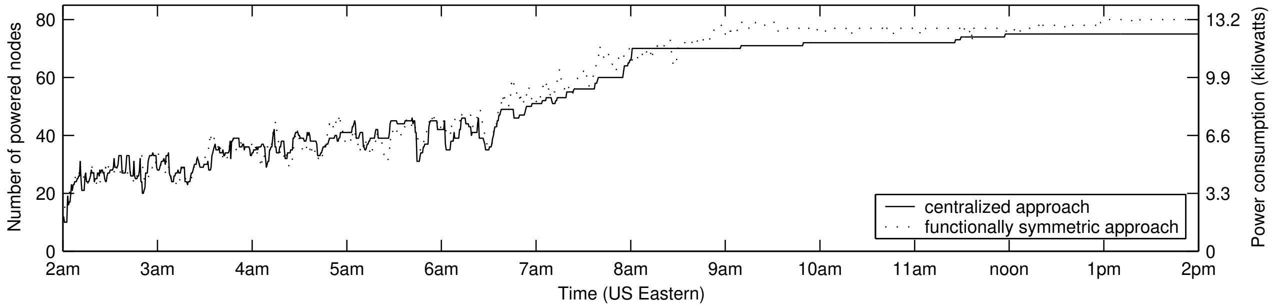

Figure 9: Comparison on the power consumption for a 12-hour period with increasing service demands.

4.1 Evaluation on Energy-Conserving Server Consolidation

We evaluate the effectiveness of the proposed functionally symmetric server consolidation scheme for conserving power consumption. We also examine whether it achieves this without affecting the system performance in terms of throughput and service response time. The experiments were conducted on a 117-server cluster. All servers run Linux kernel version 2.4.21. Each server has two 1.4GHz processors and 4GB memory. We ran a prototype search engine index service (shown in Figure 1) with 10 partitions, each of which has 8 replicas. Each partition has about 2GB data which can completely fit into the memory. Thus there is little disk activity after an index service is warmed up. An additional service was deployed to aggregate results from all partitions. We replicated this aggregation service into 32 replicas to make sure it is not the bottleneck. Four machines were used as testing clients to drive the system. One remaining machine was used as the Executive in the centralized approach. Server consolidation schemes were applied to the 80 index servers. Since online users usually demand interactive responses, requests that could not be completed at an index server within two seconds are dropped.

We first collected a 12-hour trace (2am$\rightarrow$2pm, US Eastern time) from an edge web server of Ask Jeeves search [4], which covers the trough and the peak of a one-day traffic. Queries were extracted from the trace to create a query trace file and the request arrival rates were calculated for every minute during the 12-hour period. Then we scaled the arrival rate up to 80% of the maximum throughput of the index system with 80 active servers. The maximum throughput was measured as the maximum request arrival rate at which the system could maintain a below-2% request drop rate. Finally, we cyclically played the query trace file using the extrapolated arrival rate for 12 hours. For every minute, the request arrival was modeled as a Poisson process with the extrapolated arrival rate. We periodically collected the aggregated throughput, the average response time, and the number of active servers at 30-second intervals.

We used the request queue length and the 95 percentile response time to assess the load of a server as we have discussed in Section 3.2. A server will try to power up a dormant machine if its queue length is larger than 8.00 or its assessed service response time is more than $120ms$. The decaying factors α and β are set as 0.85 and 0.95 respectively. A server will power down itself if its queue length is less than 0.80 and the response time is less than $70ms$. In the centralized approach, every index server sends its queue length and response time to the Executive every second. The Executive computes the average queue length and response time of all replicas. Then it uses the same strategies as the functionally symmetric approach with the following parameters: 5.50 and $110ms$ as the high-load watermarks for the queue length and the response time respectively; 0.75 and $70ms$ as the low-load watermarks. It uses the same decaying factors as the functionally symmetric approach. It takes around 90 seconds to power up a dormant server and initialize its index service. The power consumption of a sample server is measured as 165 watts when its processors are fully utilized.

Figure 6 shows the request arrival rate during the 12-hour period. The request arrival rate gradually grows from 22 re-

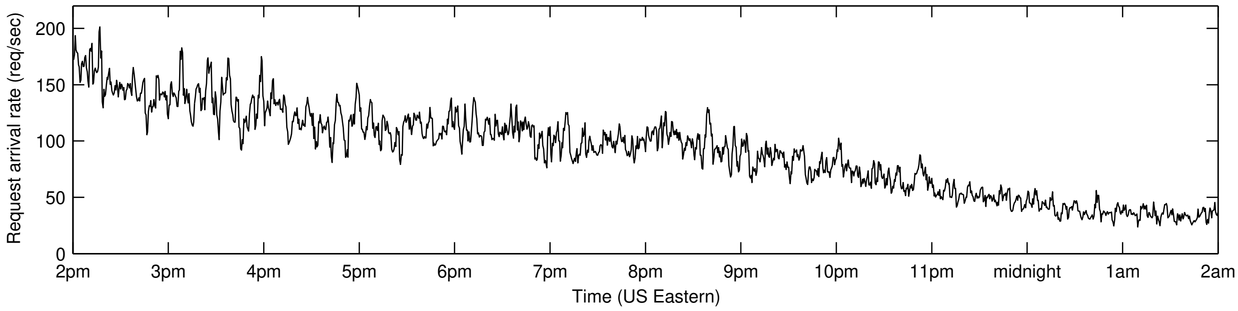

Figure 10: Request arrival rate for a 12-hour period with decreasing service demands.

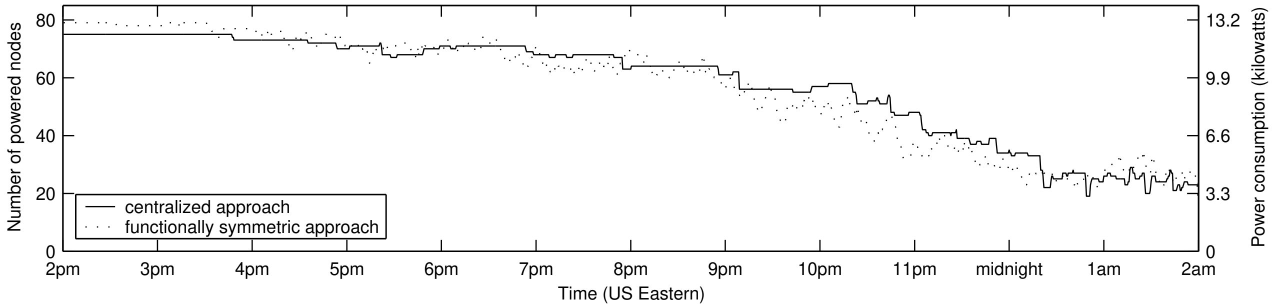

Figure 11: Comparison on the power consumption for a 12-hour period with decreasing service demands.

quests per second to 202 requests per second. Figure 7 compares the request drop rate between the functionally symmetric approach and the centralized approach. We can see that the functionally symmetric approach achieves similar low request drop rate as the centralized approach. In average, the mean drop rate for both approaches is around 0.01 request per second. This is less than 0.01% of all incoming requests (the average incoming request rate is 109.7 requests/second).

Figure 8 presents the service response time over the 12-hour period. For both approaches, the response time is effectively maintained at a low level when the service demand increases. The mean response time is measured as $87.3ms$ for the functionally symmetric approach, and $81.8ms$ for the centralized approach.

Figure 9 demonstrates the effectiveness of energy conservation for both approaches. The left Y axis shows the number of active servers during the test period, and the right Y axis shows the estimated power consumption rate of all the index servers. The mean number of active servers is 59 for the functionally symmetric approach, and 56 for the centralized approach. In other words, the centralized approach saved 47 kilowatts in average during the 12-hour test while the functionally symmetric approach saved 42 kilowatts, which is around 90% of the energy saved in the centralized approach. It is worth noticing that the curve of the centralized approach is smoother than that of the peer-to-peer approach. This is because each node in the peer-to-peer approach makes independent protocol decision according to its local view, which can result in conflicting decisions occasionally.

We also compared the functionally symmetric approach with the centralized approach when the service demand decreases. Figure 10 shows the request arrival rate for the 12-hour period (2pm→2am, US Eastern time) with decreasing service demands. We use the same protocol parameters as in the previous experiments. Our experiments show that the functionally symmetric approach also achieves similar performance compared to the centralized approach in this setting. The mean request drop rate during the 12-hour period is still maintained at 0.01 request per second for both approaches. The mean response time is $87.2ms$ for the functionally symmetric approach and $82.4ms$ for the centralized approach. Figure 11 compares the energy conservation for the both approaches. The average number of active servers is 58 for the functionally symmetric approach and 57 for the centralized approach.

Overall, our experiments show that the proposed functionally symmetric design perform very close (with at most 10% difference) to the centralized approach on all three performance and policy objectives: low service response time, low request drop rate, and energy conservation.

4.2 Evaluation on Service Availability Management

Protocol overhead for service availability management can be very significant for large-scale systems. In this section, we measure the overhead of the centralized approach, all-to-all broadcasts, and suppressive broadcasts at various cluster sizes. The purpose of this evaluation is to see whether our proposed functionally symmetric approach can support service availability management without incurring excessive overhead for large-scale clusters. Since our purpose is not

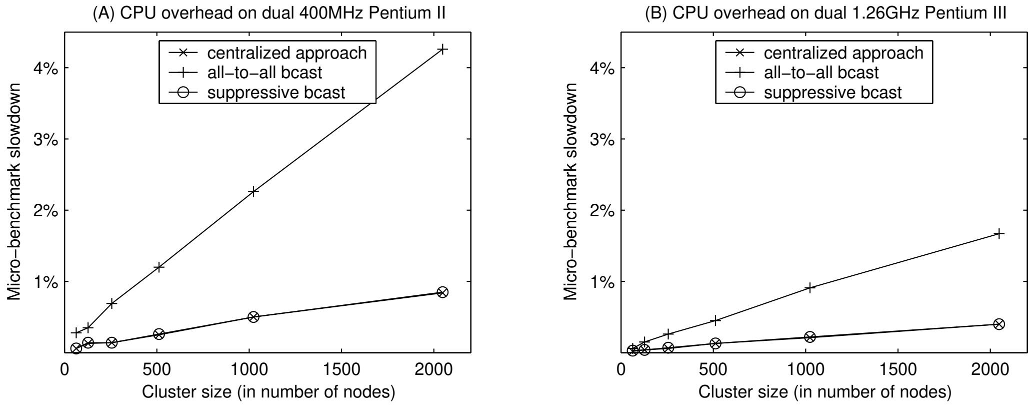

Figure 12: The slowdown of a CPU-bound micro-benchmark due to the overhead of service availability management.

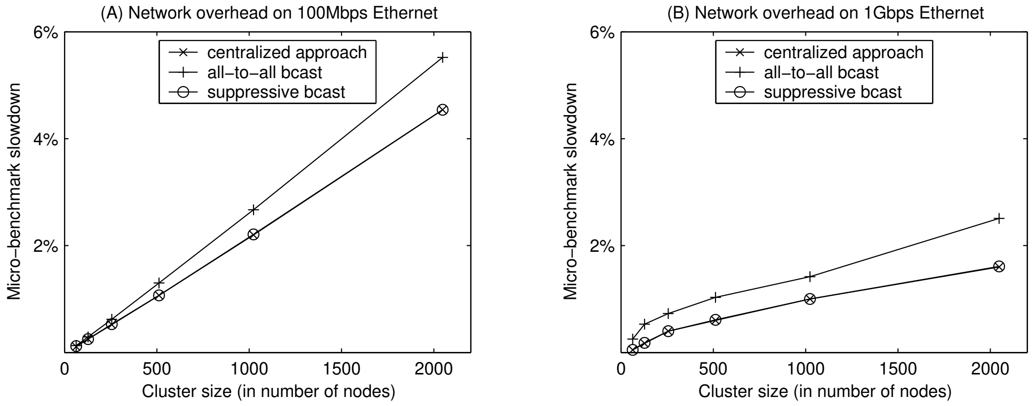

Figure 13: The slowdown of a bandwidth-bound micro-benchmark due to the overhead of service availability management.

to propose the best service availability management scheme, we do not compare with classical membership management schemes that are not based on soft state broadcasts [9, 21, 24, 31].

For the service availability management techniques in our study, the main overhead at each node includes the incoming link bandwidth consumption and the CPU consumption on message processing. In our experiments, the service availability announcements are made through UDP/IP broadcasts and the average broadcast frequency is one per second (*i.e.*, $\mathcal{P} = 1$ second) for all service availability management schemes. In order to evaluate large clusters, we use an emulation technique in this study. For an N-node cluster, we simulate $N-1$ nodes using real-time discrete-event simulation while one target node runs the real code. Our overhead measurement then focuses on the target node. For instance, the target node receives about $N-1$ UDP service availability announcements each second under the all-to-all broadcast approach. For the centralized approach, we only measure the overhead of a regular service node. Note that the protocol overhead at the central node is much higher.

We assess the overhead of the service availability management by running a service application on the target node and measuring the application performance penalty caused by the employment of the management protocol. In order to assess the CPU overhead and the network bandwidth overhead separately, we use one CPU-bound micro-benchmark and another micro-benchmark whose performance is bounded by the incoming network bandwidth. Figure 12 shows the slowdown of the CPU-bound micro-benchmark due to the overhead of each of the three service availability management schemes. Figure 13 shows such slowdown for the bandwidth-bound micro-benchmark. Experiments were conducted on Linux servers with two different CPU configurations (dual 400MHz Pentium IIs and dual 1.26GHz Pentium IIIs) and two typical network link bandwidths (100Mbps Ethernet and 1Gbps Ethernet). All Linux servers are equipped with the kernel version 2.4.20. We assume the information about locally available services at each node is about 256 byte large. Therefore, each broadcast message has 256 byte payload in average for all-to-all broadcasts. Since the broadcast messages in suppressive broadcasts contain the complete cluster-wide service availability information, their sizes are much larger at around $N \times 256$ bytes. Note that most operating systems place an upper bound on the size of a UDP segment. We broke a large broadcast message into multiple

UDP segments of 8KB each. We do not include the overhead of middleware or application-level message processing in this emulation, so only the minimum kernel-level protocol processing overhead is considered in the CPU overhead.

For all-to-all broadcasts, results in Figure 12 and Figure 13 show a linear increase in application slowdown, with up to 5.5% and 4.3% slowdown for the network-bound micro-benchmark and the CPU-bound micro-benchmark respectively on 2048-node clusters. Such an overhead may be excessive for application services running on large-scale systems. Suppressive broadcasts improve the system scalability by significantly reduce the overhead for large clusters. Its protocol overhead is very close to that of the centralized approach at all settings. This is because typically only a single large broadcast message needs to be processed during each period under both suppressive broadcasts and the centralized approach. Compared with all-to-all broadcasts, suppressive broadcasts save up to 80% on the CPU overhead and up to 36% on the network resource consumption. The performance difference is less significant for the network overhead because the total application-level traffic volume is the same between the two schemes. Although suppressive broadcasts produce fewer number of broadcast messages, each one of them is much larger. The main performance advantage comes from savings on the lower-level (*e.g.*, network and link-layer) network overhead such as packet headers due to fewer number of messages in suppressive broadcasts.

Overall, our experiments demonstrate that the suppressive broadcast approach, based on a functionally symmetric software architecture, can satisfy specified policy objectives without incurring excessive overhead for large-scale cluster-based network services (0.22% CPU overhead on dual 1.26GHz Pentium III and 1.00% network overhead on 1Gbps Ethernet for a 1000-node service cluster).

5. RELATED WORK

We discuss the related work to this paper in the following categories.

Software infrastructure for cluster-based network services. Previous studies investigated techniques for providing software infrastructure supporting online network services [15, 18, 41]. While many of these studies discussed system availability issues in the wake of server failures, there is still a lack of systematic understanding on how to construct robust service clustering support against random server failures, fluctuating service loads, and uncoordinated service enhancements and server additions. With the proliferation of fast growing network services, achieving high robustness becomes increasingly important. This paper promotes the concept of constructing robust cluster-based network services using functionally-symmetric architectures.

Earlier decentralized distributed systems. Decentralized distributed management has been studied in the past. For instance, the MOSIX distributed operating system employs a decentralized load balancing scheme based on probabilistic algorithms [5]. xFS is a serverless file system that eliminates centralized servers by distributing the functionality of the server among the clients [2]. Those earlier systems were deployed at relatively small scale (*e.g.*, several dozens nodes). Additionally, our work differs from these systems by targeting online service clustering support and by employing distributed management schemes that do not compromise the unique performance and policy objectives desired by online services.

Wide-area peer-to-peer systems and applications. Recent peer-to-peer projects have studied the design and implementation of a number of important wide-area systems and applications, such as distributed hash tables [30, 32, 39] and storage systems [11, 19, 33]. Many of these systems demonstrate great potential in providing a higher degree of robustness than hierarchical approaches for the construction of global-scale distributed systems. In addition to the non-hierarchical nature of these systems, they often employ non-deterministic approaches paired with slow-scaling overhead to achieve high scalability and robustness. Despite these successes, it is not straightforward to apply peer-to-peer design principles in improving the robustness of cluster-based network services. First, online service applications often demand distinct performance requirements and policy objectives. Those requirements are often treated as secondary goals in wide-area peer-to-peer systems. Second, we believe that maintaining slow-scaling overhead is not a significant issue due to better network performance and relatively smaller sizes of local-area server clusters.

Cluster-wide resource management. A large body of previous work has addressed issues in request distribution and resource management for cluster-based server systems, focusing on quality-of-service support [3, 34, 42], energy conservation [8, 27], or locality-aware load distribution [26]. Most of these studies rely on centralized or hierarchical architectures to manage cluster-wide resources. For instance, demand-driven service differentiation (DDSD) provides a centralized server partitioning approach to differentiating multiple classes of service requests [42]. As another example, Chase *et al.* proposed to support optimal energy-conserving server consolidation through solving a constrained optimization problem at a central Executive node [8]. We argue that decentralized architectures have the potential to provide better robustness and scalability. In particular, we constructed a functionally symmetric resource management framework supporting energy-conserving server consolidation. Our performance evaluation demonstrates its effectiveness in achieving desired resource management objectives.

Service availability management for cluster-based systems. Earlier distributed computing research examined membership management protocols in asynchronous distributed systems [9, 21, 24, 31]. Providing service fail-over and migration support for off-the-shelf applications is the focus of the SunSCALR project [38] and the Microsoft cluster service (MSCS) [40]. These solutions tend to be complex in protocol handshakes during membership changes, and even more so in dealing with concurrent asynchronous events. On the other hand, soft state-based system state maintenance has been used in many network protocols to achieve high robustness in the face of random link or node failures [29]. Motivated by these studies, a soft state-based non-hierarchical approach for maintaining cluster-wide service availability information was used in our prior work [37]. This scheme, however, incurs excessive overhead for large-scale systems.

Our contribution in this paper is the design and evaluation of a more scalable service availability management scheme based on functionally symmetric software architecture.

6. CONCLUDING REMARKS

Functionally symmetric software architectures have great potential in providing a high degree of robustness for building large-scale distributed systems. This paper examines utilizing such design architectures in providing clustering support for online network services. Our key goal is to achieve high system robustness without compromising performance and policy objectives desired by online applications. In particular, this paper presents functionally symmetric designs of two clustering support elements: energy-conserving server consolidation and service availability management. Our emulation and experimentation on a 117-server cluster demonstrate the effectiveness of proposed techniques compared with centralized approaches.

Different clustering support elements desire a diverse range of performance and policy objectives. Our study in this paper is limited to two specific service clustering support elements and further work is needed to extend to additional clustering support. Nonetheless, we believe our experience serves as an important step for providing comprehensive network service clustering support using functionally symmetric architectures.

7. ACKNOWLEDGMENTS

We would like to thank Ask Jeeves, Inc. for allowing us to use their query traces and facilities. We would also like to thank anonymous referees and our shepherd, Satoshi Matsuoka, for their valuable comments and help.

8. REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, 3rd edition, 1999.

[2] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. Serverless Network File Systems. *ACM Trans. on Computer Systems*, 14(1):41–79, February 1996.

[3] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster Reserves: A Mechanism for Resource Management in Cluster-based Network Servers. In *Proc. of the 2000 ACM SIGMETRICS Intl. Conf. on Measurement and Modeling of Computer Systems*, pages 90–101, Santa Clara, CA, June 2000.

[4] Ask Jeeves Search. http://www.ask.com.

[5] A. Barak, S. Guday, and R. G. Wheeler. *The MOSIX Distributed Operating System: Load Balancing for UNIX*, volume 672 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[6] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, B. A. Rapp, and D. L. Wheeler. GenBank. *Nucleic Acids Research*, 30(1):17–20, 2002.

[7] H. Casanova and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. *International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, 1997.

[8] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing Energy and Server Resources in Hosting Centers. In *Proc. of the 18th ACM Symp. on Operating Systems Principles*, pages 103–116, Banff, Canada, October 2001.

[9] F. Cristian and F. Schmuck. Agreeing on Processor Group Membership in Timed Asynchronous Distributed Systems. Technical Report CSE95-428, Dept. of Computer Science, UC San Diego, 1995.

[10] M. E. Crovella and A. Bestavros. Self-similarity in World Wide Web Traffic: Evidence and Possible Causes. *IEEE/ACM Trans. on Networking*, 5(6):835–846, 1997.

[11] F. Dabek, M. Frans Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area Cooperative Storage with CFS. In *Proc. of the 18th ACM Symp. on Operating Systems Principles*, pages 202–215, Banff, Canada, October 2001.

[12] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. A Set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, 1990.

[13] eBay Online Auctions. http://www.ebay.com.

[14] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 2002. http://www.globus.org/research/papers/ogsa.pdf.

[15] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-Based Scalable Network Services. In *Proc. of the 16th ACM Symp. on Operating System Principles*, pages 78–91, Saint Malo, October 1997.

[16] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *Proc. of the 19th ACM Symp. on Operating Systems Principles*, pages 29–43, Bolton Landing, NY, October 2003.

[17] Google Search. http://www.google.com.

[18] S. D. Gribble, M. Welsh, E. A. Brewer, and D. Culler. The MultiSpace: An Evolutionary Platform for Infrastructural Services. In *Proc. of the USENIX Annual Technical Conf.*, Monterey, CA, June 1999.

[19] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proc. of the 9th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 190–201, Cambridge, MA, November 2000.

[20] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic Linear Algebra Subprograms for Fortran Usage. *ACM Transactions on Mathematical Software*, 5:308–325, 1979.

[21] L. E. Moser, P. M. Melliar-Smith, and V. Agrawala. Process Membership in Asynchronous Distributed Systems. *IEEE Trans. on Parallel and Distributed Systems*, 5(5):459–473, May 1994.

[22] K. Nagaraja, X. Li, B. Zhang, R. Bianchini, R. Martin, and T. D. Nguyen. Using Fault Injection and Modeling to Evaluate the Performability of Cluster-Based Services. In *Proc. of the 4th USENIX Symp. on Internet Technologies and Systems (USITS '03)*, Seattle, WA, March 2003.

[23] H. Nakada, M. Sato, and S. Sekiguchi. Design and Implementations of Ninf: towards a Global Computing Infrastructure. *Future Generation Computing Systems*, 15(5-6):649–658, 1999.

[24] G. Neiger. A New Look at Membership Services. In *Proc. of the 15th ACM Symp. on Principles of Distributed Computing*, pages 331–340, Philadelphia, PA, May 1996.

[25] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do Internet services fail, and what can be done about it? In *Proc. of the 4th USENIX Symp. on Internet Technologies and Systems*, Seattle, WA, March 2003.

[26] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. In *Proc. of the ACM 8th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 205–216, San Jose, CA, October 1998.

[27] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Dynamic Cluster Reconfiguration for Power and Performance. In *Proc. of the Workshop on Compilers and Operating Systems for Low Power*, Barcelona, Spain, September 2001.

[28] C. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in A Distributed Environment. In *Proc. of the 9th ACM Symp. on Parallel Algorithms and Architectures*, pages 311–320, Newport, RI, June 1997.

[29] S. Raman and S. McCanne. A Model, Analysis, and Protocol Framework for Soft State-based Communication. In *Proc. of ACM SIGCOMM'99*, pages 15–25, Cambridge, Massachusetts, September 1999.

[30] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proc. of the ACM SIGCOMM*, pages 161–172, San Diego, CA, August 2001.

[31] A. Ricciardi and K. Birman. Process Membership in Asynchronous Environments. Technical Report TR93-1328, Dept. of Computer Science, Cornell University, 1995.

[32] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-scale Peer-to-Peer Systems. In *Proc. of the 18th IFIP/ACM Intl. Conf. on Distributed Systems Platforms*, Heidelberg, Germany, November 2001.

[33] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-Peer Storage Utility. In *Proc. of the 18th ACM Symp. on Operating Systems Principles*, pages 188–201, Banff, Canada, October 2001.

[34] K. Shen, H. Tang, T. Yang, and L. Chu. Integrated Resource Management for Cluster-based Internet Services. In *Proc. of the 5th USENIX Symp. on Operating Systems Design and Implementation*, pages 225–238, Boston, MA, December 2002.

[35] K. Shen, T. Yang, and L. Chu. Cluster Load Balancing for Fine-grain Network Services. In *Proc. of the Intl. Parallel & Distributed Processing Symp.*, Fort Lauderdale, FL, April 2002.

[36] K. Shen, T. Yang, and L. Chu. Clustering Support and Replication Management for Scalable Network Services. *IEEE Trans. on Parallel and Distributed Systems - Special Issue on Middleware*, 14(11):1168–1179, November 2003.

[37] K. Shen, T. Yang, L. Chu, J. L. Holliday, D. A. Kuschner, and H. Zhu. Neptune: Scalable Replication Management and Programming Support for Cluster-based Network Services. In *Proc. of the 3rd USENIX Symp. on Internet Technologies and Systems*, pages 197–208, San Francisco, CA, March 2001.

[38] A. Singhai, S.-B. Lim, and S. R. Radia. The SunSCALR Framework for Internet Servers. In *Proc. of the 28th Intl. Symp. on Fault-Tolerant Computing*, Munich, Germany, June 1998.

[39] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. of the ACM SIGCOMM*, pages 149–160, San Diego, CA, August 2001.

[40] W. Vogels, D. Dumitriu, K. Birman, R. Gamache, M. Massa, R. Short, J. Vert, J. Barrera, and J. Gray. The Design and Architecture of the Microsoft Cluster Service - A Practical Approach to High-Availability and Scalability. In *Proc. of the 28th Intl. Symp. on Fault-Tolerant Computing*, Munich, Germany, June 1998.

[41] J. Robert von Behren, E. A. Brewer, N. Borisov, M. Chen, M. Welsh, J. MacDonald, J. Lau, S. Gribble, and D. Culler. Ninja: A Framework for Network Services. In *Proc. of 2002 Annual USENIX Technical Conf.*, Monterey, CA, June 2002.

[42] H. Zhu, H. Tang, and T. Yang. Demand-driven Service Differentiation for Cluster-based Network Servers. In *Proc. of IEEE INFOCOM'2001*, Anchorage, AK, April 2001.