StarFish: highly-available block storage

Eran Gabber Jeff Fellin Michael Flaster Fengrui Gu
Bruce Hillyer Wee Teck Ng Banu Özden Elizabeth Shriver

Information Sciences Research Center
Lucent Technologies – Bell Laboratories
600 Mountain Avenue, Murray Hill, NJ 07974
{eran, jkf, mflaster, fgu, bruce, weeteck, ozden, shriver}@research.bell-labs.com

Abstract

In this paper we present StarFish, a highly-available geographically-dispersed block storage system built from commodity servers running FreeBSD, which are connected by standard high-speed IP networking gear. StarFish achieves high availability by transparently replicating data over multiple storage sites. StarFish is accessed via a host-site appliance that masquerades as a host-attached storage device, hence it requires no special hardware or software in the host computer. We show that a StarFish system with 3 replicas and a write quorum size of 2 is a good choice, based on a formal analysis of data availability and reliability: 3 replicas with individual availability of 99%, a write quorum of 2, and read-only consistency gives better than 99.9999% data availability. Although StarFish increases the per-request latency relative to a direct-attached RAID, we show how to design a highly-available StarFish configuration that provides most of the performance of a direct-attached RAID on an I/O-intensive benchmark, even during the recovery of a failed replica. Moreover, the third replica may be connected by a link with long delays and limited bandwidth, which alleviates the necessity of dedicated communication links to all replicas.

1 Introduction

It is well understood that important data need to be protected from catastrophic site failures. High-end and mid-range storage systems, such as EMC SRDF [4] and NetApp SnapMirror [17], copy data to remote sites both to reduce the amount of data lost in a failure, and to decrease the time required to recover from a catastrophic site failure. Given the plummeting prices of disk drives and of high-speed networking infrastructure, we see the possibility of extending the availability and reliability advantages of on-the-fly replication beyond the realm of expensive, high-end storage systems. Moreover, we demonstrate advantages to having more than one remote replica of the data.

In this paper, we describe the StarFish system, which provides host-transparent geographically-replicated block storage. The StarFish architecture consists of multiple replicas called storage elements (SEs), and a host element (HE) that enables a host to transparently access data stored in the SEs, as shown in Figure 1. StarFish is a software package for commodity servers running FreeBSD that communicate by TCP/IP over high-speed IP networks. There is no custom hardware needed to run StarFish. StarFish is mostly OS and machine independent, although it requires two device drivers (SCSI/FC target-mode driver and NVRAM driver) that we have implemented only for FreeBSD.

The StarFish project is named after the sea creature, since StarFish is designed to provide robust data recovery capabilities, which are reminiscent of the ability of a starfish to regenerate its rays after they are cut off.

StarFish is not a SAN (Storage Area Network), since SAN commonly refers to a Fibre Channel network with a limited geographical reach (a few tens of kilometers).

StarFish has several key achievements. First, we show that a StarFish system with the recommended configuration of 3 replicas (see Section 4) achieves good performance even when the third replica is connected by a communication line with a large delay and a limited bandwidth — a highly-available StarFish system does not require expensive dedicated communication lines to all replicas. Second, we show that StarFish achieves good performance during recovery from a replica failure, despite the heavy resource consumption of the data restoration activity. Generally, StarFish performance is close to that of a direct-attached RAID unit. Moreover, we present a general analysis that quantifies how the data availability and reliability depend on several system parameters (such as number of replicas, write quorum size, site failure rates, and site recovery speeds). This analysis leads to the suggestion that practical systems use 3 replicas and a write quorum size of 2.

In many real-world computing environments, a remote-replication storage system would be disqualified from consideration if it were to require special hardware

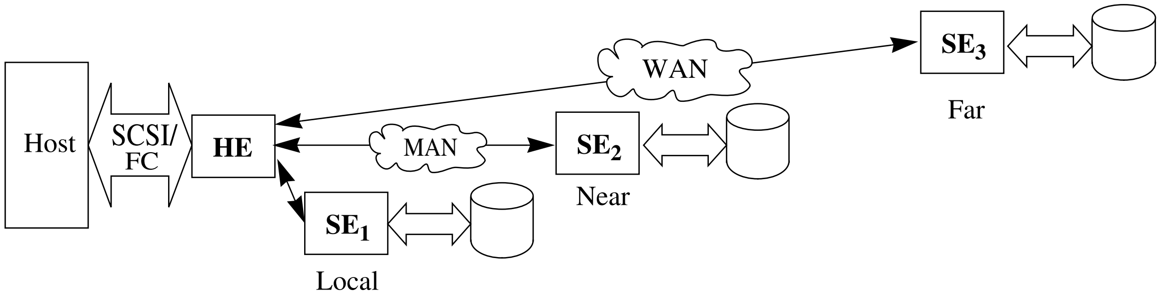

Figure 1: StarFish architecture and recommended setup.

in the host computer, or software changes in the operating system or applications. Replicating storage at the block level rather than at the file system level is general and transparent: it works with any host software and hardware that is able to access a hard disk. In particular, the host may use any local file system or a database that requires access to a hard disk, and not just a remote file system, such as NFS. The StarFish system design includes a host-site appliance that we call the host element (HE), which connects to a standard I/O bus on the host computer, as shown in Figure 1. The host computer detects the HE to be a pool of directly-attached disk drives; the HE transparently encapsulates all the replication and recovery mechanisms. In our prototype implementation, the StarFish HE connects to an Ultra-2 SCSI port (or alternately, to a Fibre Channel port) on the host computer.

If an application can use host-attached storage, it can equally well use StarFish. Thus, the StarFish architecture is broadly applicable — to centralized applications, to data servers or application servers on a SAN, and to servers that are accessed by numerous client machines in a multi-tier client/server architecture.

StarFish implements single-owner access semantics. In other words, only one host element can write to a particular logical volume. This host element may be connected to a single host or to a cluster of hosts by several SCSI buses. If we require the ability for several hosts to write to a single logical volume, this could be implemented by clustering software that prevents concurrent modifications from corrupting the data.

Large classes of data are owned (at least on a quasi-static basis) by a single server, for example in shared-nothing database architectures, and in centralized computing architectures, and for large web server clusters that partition the data over a pool of servers. The benefits that motivate the single-owner restriction are the clean serial I/O semantics combined with quick recovery/failover performance (since there is no need for distributed algorithms such as leader election, group membership, recovery of locks held by failed sites, etc.). By contrast, multiple-writer distributed replication incurs unavoidable tradeoffs among performance, strong consistency, and high reliability as explained by Yu and Vahdat [25].

To protect against a site failure, a standby host and a standby HE should be placed in a different site, and they could commence processing within seconds using an up-to-date image of the data, provided that StarFish was configured with an appropriate number of replicas and a corresponding write quorum size. See Section 3 for details.

The remainder of this paper is organized as follows. Section 2 compares and contrasts StarFish with related distributed and replicated storage systems. Section 3 describes the StarFish architecture and its recommended configuration. Section 4 analyzes availability and reliability. Section 5 describes the StarFish implementation, and Section 6 contains performance measurements. We encountered several unexpected hurdles and dead ends during the development of StarFish, which are listed in Section 7. The paper concludes with a discussion of future work in Section 8 and concluding remarks in Section 9.

2 Related work

Many previous projects have established a broad base of knowledge on general techniques for distributed systems (e.g., ISIS [2]), and specific techniques applicable to distributed storage systems and distributed file systems. Our work is indebted to a great many of these; space limitations permit us to mention only a few.

The EMC SRDF software [4] is similar to StarFish in several respects. SRDF uses distribution to increase reliability, and it performs on-the-fly replication and updating of logical volumes from one EMC system to another, using synchronous remote writes to favor safety, or using asynchronous writes to favor performance. The first EMC system owns the data, and is the *primary* for the classic primary copy replication algorithm. By comparison, the StarFish *host* owns the data, and the HE implements primary copy replication to *multiple* SEs. StarFish typically uses synchronous updates to a subset of the SEs for safety, with asynchronous updates to additional SEs to increase availability. Note that this comparison is not intended as a claim that StarFish has features, performance, or price equivalent to an EMC Symmetrix.

Petal [11] is a distributed storage system from Compaq SRC that addresses several problems, including scaling up and reliability. Petal's network-based servers pool their physical storage to form a set of virtual disks. Each block on a virtual disk is replicated on two Petal servers. The Petal servers maintain mappings and other state via distributed consensus protocols. By contrast, StarFish uses N-way replication rather than 2-way replication, and uses an all-or-none assignment of the blocks of a logical volume to SEs, rather than a declustering scheme. StarFish uses a single HE (on a quasi-static basis) to manage any particular logical volume, and thereby avoids distributed consensus.

Network Appliance SnapMirror [17] generates periodic snapshots of the data on the primary filer, and copies them asynchronously to a backup filer. This process maintains a slightly out-of-date snapshot on the backup filer. By contrast, StarFish copies all updates on-the-fly to all replicas.

The iSCSI draft protocol [19] is an IETF work that encapsulates SCSI I/O commands and data for transmission over TCP/IP. It enables a host to access a remote storage device as if it were local, but does not address replication, availability, and system scaling.

StarFish can provide replicated storage for a file system that is layered on top of it. This configuration is similar but not equivalent to a distributed file system (see surveys in [24] and [12]). By contrast with distributed file systems, StarFish is focused on replication for availability of data managed by a single host, rather than on unreplicated data that are shared across an arbitrarily scalable pool of servers. StarFish considers network disconnection to be a failure (handled by failover in the case of the HE, and recovery in case of an SE), rather than a normal operating condition.

Finally, Gibson and van Meter [6] give an interesting comparison of network-attached storage appliances, NASD, Petal, and iSCSI.

3 StarFish architecture

As explained in Section 1, the StarFish architecture consists of multiple storage elements (SEs), and a host element (HE). The HE enables the host to access the data stored on the SEs, in addition to providing storage virtualization and read caching. In general, one HE can serve multiple logical volumes and can have multiple connections to several hosts. The HE is a commodity server with an appropriate SCSI or FC controller that can function in target mode, which means that the controller can receive commands from the I/O bus. The HE has to run an appropriate target mode driver, as explained in Section 5.

We replicate the data in N SEs to achieve high availability and reliability; redundancy is a standard technique to build highly-available systems from unreliable components [21]. For good write performance, we use a quorum technique. In particular, the HE returns a success indication to the host after Q SEs have acknowledged the write, where Q is the *write quorum size*. In other words, StarFish performs synchronous updates to a quorum of Q SEs, with asynchronous updates to additional SEs for performance and availability.

Since StarFish implements single-owner access to the data, it can enforce consistency among the replicas by serialization: the HE assign global sequence numbers to I/O requests, and the SEs perform the I/Os in this order to ensure data consistency. Moreover, to ensure that the highest update sequence number is up to date, the HE does not delay or coalesce write requests. To simplify failure recovery, each SE keeps the highest update sequence number (per logical volume) in NVRAM. This simple scheme has clear semantics and nice recovery properties, and our performance measurements in Section 6 indicate that it is fast.

Figure 1 shows a recommended StarFish setup with 3 SEs. The "local" StarFish replica is co-located with the host and the HE to provide low-latency storage. The second replica is "near" (e.g., connected by a dedicated high-speed, low-latency link in a metro area) to enable data to be available with high performance even during a failure and recovery of the local replica. A third replica is "far" from the host, to provide robustness in the face of a regional catastrophe. The availability of this arrangement is studied in Section 4, and the performance is examined in Section 6.

The HE and the SEs communicate via TCP/IP sockets. We chose TCP/IP over other reliable transmission protocols, such as SCTP [23], since TCP/IP stacks are widely available, optimized, robust, and amenable to hardware acceleration. Since many service providers sell Virtual Private Network (VPN) services, the HE may communicate with the far SE StarFish via a VPN, which provides communication security and ensures predictable latency and throughput. StarFish does not deal with communication security explicitly.

StarFish could be deployed in several configurations depending on its intended use. Figure 2 shows a deployment by an enterprise that has multiple sites. Note that in this configuration the local SE is co-located with the host and the HE. A storage service provider (SSP) may deploy StarFish in a configuration similar to Figure 1, in which all of the SEs are located in remote sites belonging to the SSP. In this configuration the local SE may not be co-located with the host, but it should be nearby for good performance.

StarFish is designed to protect against SE failures,

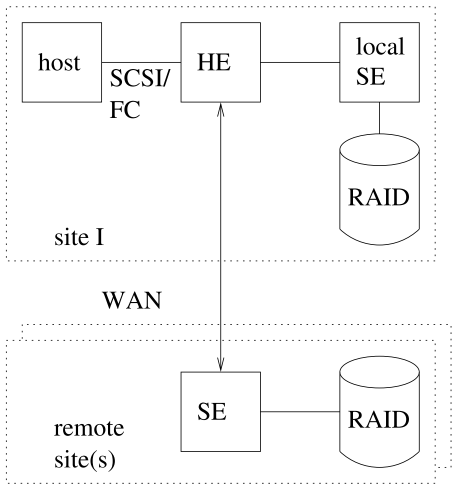

Figure 2: StarFish deployment in an enterprise.

network failure to some SEs, and HE failure. When an SE (or its RAID or network connection) fails, the HE continues to serve I/Os to the affected logical volumes, provided that Q copies are still in service. When the failed SE comes back up, it reconnects to the HE and reports the highest update sequence number of its logical volumes. This gives the HE complete information about what updates the SE missed. For each logical volume, the HE maintains a circular buffer of recent writes (the "write queue"). If an SE fails and recovers quickly (in seconds), it gets the missed writes from the HE. This recovery is called "quick recovery". Also, each SE maintains a circular buffer of recent writes on a log disk. If an SE fails and recovers within a moderate amount of time (in hours — benchmark measurements from Section 6 suggest that the log disk may be written at the rate of several GB per hour) the HE tells the SE to retrieve the missed writes from the log disk of a peer SE. This recovery is called "replay recovery". Finally, after a long failure, or upon connection of a new SE, the HE commands it to perform a whole-volume copy from a peer SE. This recovery is called "full recovery". During recovery, the SE also receives current writes from the HE, and uses sequence number information to avoid overwriting these with old data. To retain consistency, the HE does not ask a recovering SE to service reads.

The fixed-size write queue in the HE serves a second purpose. When an old write is about to be evicted from this queue, the HE first checks that it has been acknowledged by all SEs. If not, the HE waits a short time (throttling), and if no acknowledgment comes, declares the SE that did not acknowledge as failed. The size of the write queue is an upper bound on the amount of data loss, because the HE will not accept new writes from the host until there is room in the write queue.

In normal operation (absent failures), congestion cannot cause data loss. If any SE falls behind or any internal queue in the HE or the SE becomes full, the HE will stop accepting new SCSI I/O requests from the host.

The HE is a critical resource. We initially implemented a redundant host element using a SCSI switch, which would have provided automatic failure detection and transparent failover from the failed HE to a standby HE. However, our implementation encountered many subtle problems as explained in Section 7, so we decided to eliminate the redundant host element from the released code.

The current version of StarFish has a manually-triggered failover mechanism to switch to a standby HE when necessary. This mechanism sends an SNMP command to the SEs to connect to the standby HE. The standby HE can resume I/O activity within a few seconds, as explained in Section 6.4. The standby HE will assume that same Fibre Channel or SCSI ID of the failed HE. The manually-triggered failover mechanism can be replaced by an automatic failure detection and reconfiguration mechanism external to StarFish. One challenge in implementing such automatic mechanism is in the correct handling of network partitions and other communication errors. We can select one of the existing distributed algorithms for leader election for this purpose. If the HE connects to the host with Fibre Channel, the failover is transparent except for a timeout of the commands in transit. However, starting the standby HE on the same SCSI bus as the failed HE without a SCSI switch will cause a visible bus reset.

Since the HE acknowledges write completions only after it receives Q acknowledgments from the SEs, the standby HE can find the last acknowledged write request by receiving the highest update sequence number of each volume from Q SEs. If any SE missed some updates, the standby HE instructs it to recover from a peer SE.

4 Availability and reliability analysis

In this section we evaluate the availability and reliability of StarFish with respect to various system parameters. This enables us to make intelligent trade-offs between performance and availability in system design. Our main objectives are to quantify the availability of our design and to develop general guidelines for a highly available and reliable system.

The availability of StarFish depends on the SE failure ($\lambda(t)$) and recovery ($\mu(t)$) processes, the number of SEs (N), the quorum size (Q), and the permitted probability of data loss. The latter two parameters also bear on StarFish reliability. We assume that the failure and recovery processes of the network links and storage elements are independent identically distributed Poisson processes [3] with combined (i.e., network + SE) *mean* failure and recovery rates of λ and μ failures and re-

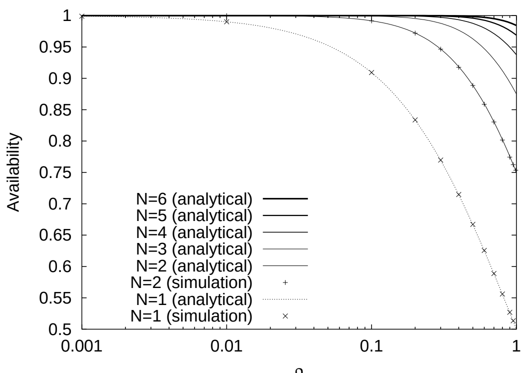

Figure 3: Availability with $Q = 1$, $N =$1–6.

coveries per second, respectively. Similarly, the HE has Poisson-distributed λ_{he} and μ_{he}. Section 9 suggests ways to achieve independent failures in practice.

At time t, a component SE or HE is available if it is capable of serving data. We define the availability of StarFish as the steady-state probability that at least Q SEs are available. For example, a system that experiences an 1-day outage every 4 months is 99% available. We assume that if the primary HE fails, we can always reach the available SEs via a backup HE.

We define the reliability of a system as the probability of no data loss. For example, a system that is 99.9% reliable has a 0.1% probability of data loss.

4.1 Availability results

The steady-state availability of StarFish, $A(Q, N)$, is derived from the standard machine repairman model [3] with the addition of a quorum. It is the steady-state probability that at most Q SEs are down and at least $(N-Q)$ SEs are up. Thus, the stead-state availability can be expressed as

$$A(Q,N) \quad = \quad \frac{\sum_{i=0}^{N-Q} \binom{N}{i} \rho^i}{(1+\rho)^N} \qquad (1)$$

where $\rho = \lambda/\mu$ is called the load, and $1 \leq Q \leq N \; \forall Q, N$. Eq. 1 is valid for $0 < \rho < 1$ (i.e., when the failure rate is less than the recovery rate). Typical values of ρ range from 0.1 to 0.001, which correspond to hardware availability of 90% to 99.9%. See [5] for derivation of Eq. 1.

Figure 3 shows the availability of StarFish using Eq. 1 with a quorum size of 1 and increasing number of SEs. We validate the analytical model up to $N = 3$ with an event-driven simulation written using the `smpl` simulation library [13]. Because the analytical results are in close agreement with the simulation results, we will not present our simulation results in rest of this paper.

We observe from Figure 3 that availability increases with N. This can also be seen from Eq. 1, which is strictly monotonic and converges to 1 as $N \to \infty$. We also note that availability $\to 1$ as $\rho \to 0$. This is because the system becomes highly available when each SE seldom fails or recovers quickly.

We now examine the system availability for *typical* configurations. Table 1 shows StarFish's availability in comparison with a single SE. We use a concise availability metric widely used in the industry, which counts the number of 9s in an availability measure. For example, a system that is 99.9% available is said to have three 9s, which we denote $3 \star 9$. We use a standard design technique to build a highly available system out of redundant unreliable components [21]. The SE is built from commodity components and its availability ranges from 90% [14] to 99.9% [7]. StarFish combines SEs to achieve a much higher system availability when $N = 2Q + 1$. For example, Table 1 indicates that if the SEs are at least 99% available, StarFish with a quorum size of 1 and 3 SEs is 99.9999% available ($6 \star 9$).

We also notice from Table 1 that, for fixed N, StarFish's availability *decreases* with larger quorum size. In fact, for $Q = N = 3$, StarFish is less available than a single SE, because the probability of keeping all SEs concurrently available is lower than the probability that a single one is available. Increasing quorum size trades off availability for reliability. We quantify this trade-off next.

4.2 Reliability results

StarFish can potentially lose data under certain failure scenarios. Specifically, when $Q \leq \lfloor N/2 \rfloor$, StarFish can lose data if the HE and Q SEs containing up-to-date data fail. The *amount* of data loss is bounded by the HE write queue size as defined in Section 3. This queue-limited exposure to data loss is similar to the notion of time-limited exposure to data loss widely used in file systems [20]. We make this trade-off to achieve higher performance and availability at the expense of a slight chance of data loss. The *probability* of data loss is bounded by $1/4^Q$, which occurs when $\rho = 1$ and $\rho_{he} = 0$. This implies that the lowest reliability occurs when $Q = 1$, and the reliability increases with larger Q.

We note that there is no possibility of data loss if $Q > \lfloor N/2 \rfloor$ and at least Q SEs are available. This is because when we have a quorum size which requires the majority of SEs to have up-to-date data when available, failures in the remaining SEs do not affect system reliability as we still have up-to-date data in the Q remaining SEs. However, this approach can reduce availability (see Table 1) and performance (see Section 6.3).

Another approach is to trade-off the system functionality while still maintaining the performance *and* reliability requirements. For example, we may allow StarFish to be available in a *read-only* mode during failure, which we call StarFish with *read-only consistency*.

Table 1: Availability of StarFish for typical configurations.

	$Q=1$			$Q=2$			$Q=3$			
SE availability	$N=2$	$N=3$	$N=4$	$N=3$	$N=4$	$N=5$	$N=3$	$N=5$	$N=6$	$N=7$
90.00% ($1 \star 9$)	$2 \star 9$	$3 \star 9$	$4 \star 9$	$1 \star 9$	$2 \star 9$	$3 \star 9$	$0 \star 9$	$2 \star 9$	$3 \star 9$	$3 \star 9$
99.00% ($2 \star 9$)	$4 \star 9$	$6 \star 9$	$8 \star 9$	$3 \star 9$	$5 \star 9$	$7 \star 9$	$1 \star 9$	$5 \star 9$	$6 \star 9$	$8 \star 9$
99.90% ($3 \star 9$)	$6 \star 9$	$9 \star 9$	$12 \star 9$	$5 \star 9$	$8 \star 9$	$11 \star 9$	$2 \star 9$	$8 \star 9$	$10 \star 9$	$13 \star 9$
99.99% ($4 \star 9$)	$8 \star 9$	$12 \star 9$	$16 \star 9$	$7 \star 9$	$11 \star 9$	$15 \star 9$	$3 \star 9$	$11 \star 9$	$14 \star 9$	$18 \star 9$

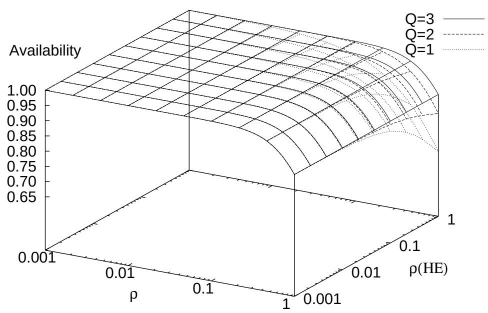

Figure 4: Availability with $N = 3$, $Q = $1–3, read-only consistency.

Read-only mode obviates the need for Q SEs to be available to handle updates. This increases the availability of the system, as it is available for reads when the HE and at least 1 SE is available, or if any SE with up-to-date data is available and we fail over to a standby HE as described in Section 3. With read-only consistency, StarFish has steady-state availability $A_{\text{ReadOnly}}(Q, N)$ of

$$\frac{\sum_{i=0}^{N-1} \binom{N}{i} \rho^i}{(1+\rho_{he})(1+\rho)^N} + \frac{\rho_{he} \sum_{i=0}^{Q-1} \binom{Q}{i} \rho^i}{(1+\rho_{he})(1+\rho)^Q}. \quad (2)$$

Eq. 2 is derived using Bayes Theorem [3]. We assume that the HE and SEs fail independently. The availability of StarFish with read-only consistency is union of two disjoint events: the HE and *any Q* SEs are alive, the HE fails and *Q distinct* SEs with current updates are alive.

Figure 4 shows the availability of StarFish with read-only consistency with respect to the SE load, ρ, and HE load (i.e., $\rho(HE)$ in Figure 4). We observe that when the HE is always available (i.e., $\rho(HE) = 0$), StarFish's availability is independent of the quorum size since it can always recover from the HE. When the HE becomes less available, we observe that StarFish's availability *increases* with larger quorum size. Moreover, the largest increase occurs from $Q = 1$ to $Q = 2$ and is bounded by 3/16 when $\rho = 1$. This implies a diminishing gain in availability beyond $Q = 2$, and we recommend using a quorum size of 2 for StarFish with read-only consistency. Table 2 shows StarFish's availability with read-only consistency for typical failure and recovery rates.

We suggest that practical systems select $Q = 2$, since a larger Q offers diminishing improvements in availability. To prevent data loss, N must be $< 2Q$. Thus we suggest that $N = 3$ and $Q = 2$ is a reasonable choice. In this configuration, StarFish with SE availability of 99% and read-only consistency is 99.9999% available ($6 \star 9$).

Increasing N in general reduces the performance, since it increases the load on the HE. The only way that a large N can improve performance is by having more than one local SE. In this configuration the local SEs can divide the load of read requests without incurring transmission delays. Alternately, all local SE may receive the same set of read requests, and the response from the first SE is sent to the host.

Although StarFish may lose data with $Q = 1$ after a single failure, the $Q = 1$ configuration may be useful for applications requiring read-only consistency, and applications that can tolerate data inconsistency like newsgroup and content distribution. Note that $Q = 1$ is the current operating state for commercial systems that are not using synchronous remote copy mechanisms, so this configuration is useful as a reference.

Table 3: StarFish code size

component	language	source lines
host element (HE)	C++	9,400
storage element (SE)	C++	8,700
common code	C/C++	18,000
SCSI target mode driver	C	5,700
NVRAM driver	C	700
total		42,500

5 Implementation

Table 3 shows the number of source lines and language for various components of StarFish. The "common code" includes library functions and objects that are used by both the HE and SE. The HE and SE are user-level processes, which provide better portability and simplify debugging relative to kernel-level servers. The main disadvantage of user-level processes is extra data copies to/from user buffers, which is avoided by kernel-level servers. Since the main bottleneck of the HE is the TCP/IP processing overhead (see Section 9), this is a reasonable tradeoff.

The HE and SE are multi-threaded. They use the LinuxThreads package, which is largely compatible with

Table 2: Availability of StarFish with read-only consistency for typical failure and recovery rates.

	$Q=1$			$Q=2$			$Q=3$		
SE availability	$N=2$	$N=3$	$N=4$	$N=3$	$N=4$	$N=5$	$N=3$	$N=4$	$N=5$
90.00% ($1 \star 9$)	$2 \star 9$	$3 \star 9$	$4 \star 9$	$3 \star 9$	$4 \star 9$	$5 \star 9$	$3 \star 9$	$4 \star 9$	$5 \star 9$
99.00% ($2 \star 9$)	$4 \star 9$	$5 \star 9$	$6 \star 9$	$6 \star 9$	$7 \star 9$	$8 \star 9$	$6 \star 9$	$8 \star 9$	$9 \star 9$
99.90% ($3 \star 9$)	$5 \star 9$	$6 \star 9$	$7 \star 9$	$8 \star 9$	$9 \star 9$	$10 \star 9$	$9 \star 9$	$11 \star 9$	$12 \star 9$
99.99% ($4 \star 9$)	$7 \star 9$	$8 \star 9$	$8 \star 9$	$11 \star 9$	$12 \star 9$	$12 \star 9$	$12 \star 9$	$15 \star 9$	$16 \star 9$

Pthreads [16]. LinuxThreads creates a separate process for each thread with the `rfork()` system call, enabling the child process to share memory with its parent.

StarFish maintains a collection of log files. There is a separate log file per StarFish component (HE or SE). StarFish closes all active log files once a day and opens new ones, so that long-running components have a sequence of log files, one per day. We found that StarFish initially suffered from many bugs and transient problems that caused failure and recovery of one component without causing a global failure. Scanning the log files once a day for failure and recovery messages was invaluable for detecting and correcting these problems.

StarFish code expects and handles failures of every operation. Every routine and method returns a success or failure indication. The calling code has the opportunity to recover or propagate the failure. The following code excerpt illustrates StarFish's error handling.

```
if ((code = operation(args)) != OK) {
   // log the failure
   user_report(severity,
      "operation failed due to %s",
      get_error_text(code));
   // propagate the error
   return code;
}
```

There are several reasons for choosing this coding style over exceptions. First and foremost, this coding style is applicable for C and C++. Second, it encourages precise handling of failures at the place they occurred, instead of propagating them to an exception handler in a routine higher in the calling chain. Third, failures are the rule rather than the exception in a distributed system. Handling failures close to the place they occur enables StarFish to recover gracefully from many transient failures.

StarFish has two new device drivers: a SCSI target mode driver, and an NVRAM driver. These drivers are the only part of the project that is operating-system dependent. The drivers are written in C for FreeBSD versions 4.x and 5.x.

The SCSI target mode driver enables a user-level server to receive incoming SCSI requests from the SCSI bus and to send responses. In this way, the server emulates the operation of a SCSI disk. The target driver uses the FreeBSD CAM [8] subsystem to access Adaptec and Qlogic SCSI controllers. It can also be used to access any other target mode controllers that have a CAM interface, such as Qlogic Fibre Channel controllers.

As a side note, we could not use Justin Gibbs' target mode driver since it was not sufficient for our needs when we started the project. By the time Nate Lawson's target mode driver [10] was available, our driver implementation was completed. In addition, our driver supports tagged queuing, which does not appear to be supported by Lawson's driver.

The NVRAM driver maps the memory of Micro Memory MM5415 and MM5420 non-volatile memory (NVRAM) PCI cards to the user's address space. The HE and the SE store persistent information that is updated frequently in the NVRAM, such as update sequence numbers.

6 Performance measurements

This section describes the performance of StarFish for representative network configurations and compares it to a direct-attached RAID unit. We present the performance of the system under normal operation (when all SEs are active), and during recoveries. We also investigate the performance implications of changing the parameters of the recommended StarFish configuration.

6.1 Network configurations, workloads, and testbed

We measure the performance of StarFish in a configuration of 3 SEs (local, near, and far), under two emulated network configurations: a dark-fiber network, and a combination of Internet and dark fiber. In our testbed, the dark-fiber links are modeled by gigabit Ethernet (GbE) with the FreeBSD `dummynet` [18] package to add fixed propagation delays. The Internet links are also modeled by GbE, with `dummynet` applying both delays and bandwidth limitations. We have not studied the effects of packet losses, since we assume that a typical StarFish configuration would use a VPN to connect to the far SE for security and guaranteed performance. This is also the reason we have not studied the effects of Internet transient behavior.

Since the speed of light in optical fiber is approximately 200km per millisecond, a one-way delay of 1ms

Table 4: PostMark parameters

parameter	value
# files	40904
# transactions	204520
median working set size (MB)	256
host VM cache size (MB)	64
HE cache size (MB)	128

represents the distance between New York City and a back office in New Jersey, and a delay of 8ms represents the distance between New York City and Saint Louis, Missouri. The delay values for the far SE when it is connected by Internet rather than dark fiber (values from the AT&T backbone [1]) are 23ms (New York to Saint Louis), 36ms (continental US average), and 65ms (New York to Los Angeles). We use Internet link bandwidths that are 20%, 33%, 40%, 60%, and 80% of an OC-3 line, i.e., 31Mb/s, 51Mb/s, 62Mb/s, 93Mb/s, and 124Mb/s.

We measure the performance of StarFish with a set of micro-benchmarks (see Section 6.3) and PostMark version 1.5. PostMark [9] is a single-threaded synthetic benchmark that models the I/O workload seen by a large email server. The email files are stored in a UNIX file system with soft updates disabled. (Disabling soft updates is a conservative choice, since soft updates increase concurrency, thereby masking the latency of write operations.) We used the PostMark parameters depicted in Table 4. Initial file size is uniformly distributed between 500 and 10,000 bytes, and files never grow larger than 10,000 bytes. In all of the following experiments, we measured only performance of the transactions phase of the PostMark benchmark.

In FreeBSD, the VM cache holds clean file system data pages, whereas the buffer cache holds dirty data [15, section 4.3.1]. The host VM cache size and the HE cache size are chosen to provide 25% and 50% read hit probability, respectively. Because the workload is larger than the host VM cache, it generates a mixture of physical reads and writes. We control the VM cache size by changing the kernel's physical memory size prior to system reboot. We verified the size of the host VM cache for every set of parameters to account for the memory that is taken by the OS kernel and other system processes that are running during the measurements.

Figure 5 depicts our testbed configuration of a single host computer connected to either (a) a direct-attached RAID unit, or (b) a StarFish system with 3 SEs. All RAID units in all configurations are RaidWeb Arena II with 128MB write-back cache and eight IBM Deskstar 75GXP 75GB 7,200 RPM EIDE disks and an external Ultra-2 SCSI connection. The HE and SEs are connected via an Alteon 180e gigabit Ethernet switch. The host computer running benchmarks is a Dell PowerEdge 2450 server with dual 733 MHz Pentium III processors running FreeBSD 4.5. The HE is a SuperMicro 6040 server with dual 1GHz Pentium III processors running FreeBSD 4.4. The SEs are Dell PowerEdge 2450 servers with dual 866 MHz Pentium III processors running FreeBSD 4.3.

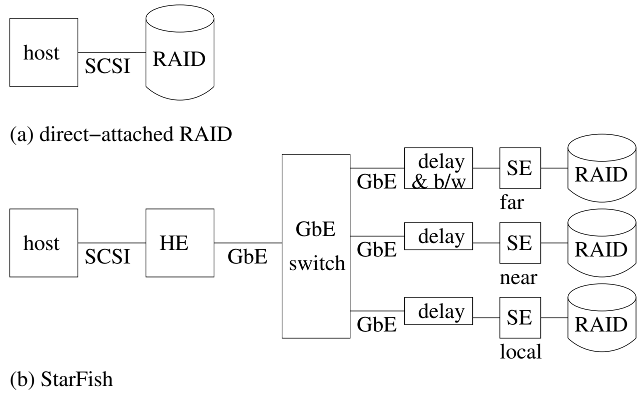

Figure 5: Testbed configuration.

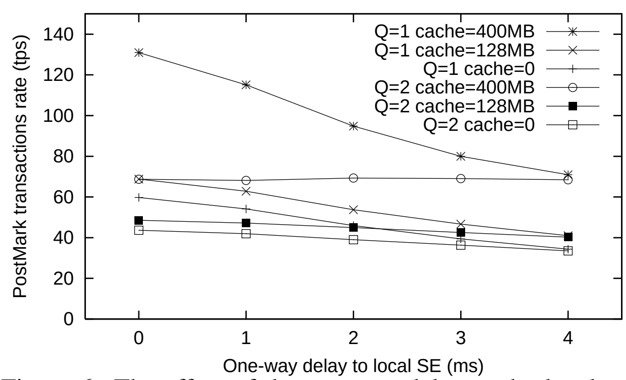

Figure 6: The effect of the one-way delay to the local SE and the HE cache size on PostMark transaction rate. The near SE and the far SE have one-way delays of 4 and 8ms, respectively.

6.2 Effects of network delays and HE cache size on performance

In this section we investigate variations from the recommended setup to see the effect of delay to the local SE and the effect of the HE cache. Figure 6 shows the effects of placing the local SE farther away from the HE, and the effects of changing the HE cache size. In this graph the near SE and the far SE have one-way delays of 4 and 8ms, respectively. If the HE cache size is 400MB, all read requests hit the HE cache for this workload. The results of Figure 6 are not surprising. A larger cache improves PostMark performance, since the HE can respond to more read requests without communicating with any SE. A large cache is especially beneficial when the local SE has significant delays. The benefits of using a cache to hide network latency have been established before (see, for instance, [15]).

However, a larger cache does not change the response time of write requests, since write requests must receive responses from Q SEs and not from the cache. This is the reason PostMark performance drops with increasing latency to the local SE for $Q = 1$. When $Q = 2$, the limiting delay for writes is caused by the near SE, rather than the local SE. The performance of $Q = 2$ also depends on the read latency, which is a function of the cache hit rate and the delay to the local SE for cache misses.

All configurations in Figure 6 but one show decreasing transaction rate with increasing delay to the local SE. The performance of the configuration $Q = 2$ and 400MB cache size is not influenced by the delay to the local SE, because read requests are served by the cache, and write requests are completed only after both the local and the near SE acknowledge them.

In summary, the performance measurements in Figure 6 indicate that with $Q = 1$ StarFish needs the local SE to be co-located with the HE; with $Q = 2$ StarFish needs a low delay to the near SE; and the HE cache significantly improves performance.

6.3 Normal operation and placement of the far SE

To examine the performance of StarFish during normal operation, we use micro-benchmarks to reveal details of StarFish's performance, and PostMark to show performance under a realistic workload.

The 3 micro-benchmarks are as follows. **Read hit**: after the HE cache has been warmed, the host sends 50,000 random 8KB reads to a 100MB range of disk addresses. All reads hit the HE cache (in the StarFish configuration), and hit the RAID cache (in the host-attached RAID measurements). **Read miss**: the host sends 10,000 random 8KB reads to a 2.5GB range of disk addresses. The HE's cache is disabled to ensure no HE cache hits. **Write**: the host sends 10,000 random 8KB writes to a 2.5GB range of disk addresses.

We run a variety of network configurations. There are 10 different dark-fiber network configurations: every combination of one-way delay to the near SE that is one of 1, 2, or 4ms, and one-way delay to the far SE that is one of 4, 8, or 12ms. The 10th configuration has one-way delay of 8ms to both the near and far SEs. In all configurations, the one-way delay to the near SE is 0. We also present the measurements of the Internet configurations which are every combination of one-way delay to the far SE that is one of 23, 36, or 65ms, and bandwidth limit to the far SE that is one of 31, 51, 62, 93, or 124Mbps. In all of the Internet configurations the one-way delay to the local and near SEs are 0 and 1ms, respectively.

Table 5: StarFish average write latency on dark-fiber and Internet network configurations with N=3. The local SE has a one-way delay of 0 in all configurations. The Internet configurations have one-way delay to the far SE that ranges from 23 to 65 ms and bandwidth limit that ranges from 31 to 124 Mb/s. Maximum difference from the average is 5%.

	configuration			write latency (ms)
	near SE delay	far SE delay/bandwidth	# conf.	
RAID	-	-	1	2.6
dark fiber configurations:				
$Q = 1$	1–8	4–12	10	2.6
$Q = 2$	1	4–12	3	3.4
$Q = 2$	2	4–12	3	5.3
$Q = 2$	4	4–12	3	9.2
$Q = 2$	8	8	1	17.2
Internet configurations:				
$Q = 1$	1	23–65/31–124	4	2.6
$Q = 2$	1	23–65/31–124	4	3.4

Table 6: StarFish micro-benchmarks on dark-fiber networks with N=3. Maximum difference from the average is 7%. The results of the Internet network configurations are equivalent to the corresponding dark-fiber network configurations.

configuration	# conf.	single-threaded latency (ms)		multi-threaded throughput (MB/s)		
		read miss	read hit	write	read miss	read hit
RAID	1	7.2	0.4	3.0	4.9	25.6
$Q = 1$	10	9.4	0.4	3.0	4.6	26.3
$Q = 2$	10	9.4	0.4	3.0	4.6	26.3

Table 5 shows the write latency of the single-threaded micro-benchmark on dark-fiber network configurations. The write latency with $Q = 1$ is independent of the network configuration, since the acknowledgment from the local SE is sufficient to acknowledge the entire operation to the host. However, the write latency with $Q = 2$ is dependent on the delay to the near SE, which is needed to acknowledge the operation. For every millisecond of additional one-way delay to the near SE, the write latency with $Q = 2$ increases by about 2ms, which is the added round-trip time. The write latency on the Internet configurations is the same as the corresponding dark-fiber configurations, since the local SE (for $Q = 1$) and near SE (for $Q = 2$) have the same delay as in the dark-fiber configuration.

Table 6 shows the micro-benchmark results on the same 10 dark-fiber network configurations as in Table 5. Table 6 shows that the read miss latency is independent of the delays and bandwidth to the non-local SEs be-

Table 7: Average PostMark performance on dark-fiber and Internet networks with 2 and 3 SEs. The local SE has a one-way delay of 0 in all configurations. The Internet configurations have one-way delay to the far SE that is either 23 or 65ms and bandwidth limit that is either of 51 or 124Mbps. Maximum difference from the average is 2%. For this workload, PostMark performs 85% writes.

Q	near SE delay	far SE delay/band.	# conf.	PostMark (tps)
RAID			1	73.12
dark fiber configurations N=2:				
1	1	-	1	71.01
2	1	-	1	65.64
dark fiber configurations N=3:				
1	1–8	4–12	10	68.80
2	1	4–12	3	63.85
2	2	4–12	3	57.97
2	4	4–12	3	48.57
2	8	8	1	35.53
Internet configurations N=3:				
1	1	{23,65}/{51,124}	4	67.98
2	1	{23,65}/{51,124}	4	62.46

cause StarFish reads from the closest SE. The read hit latency is fixed in all dark-fiber configurations, since read hits are always handled by the HE. The latency of the Internet configurations is equivalent to the dark-fiber configurations since no read requests are sent to the far SE. Tables 5 and 6 indicate that as long as there is an SE close to the host, StarFish's latency with $Q = 1$ nearly equals a direct-attached RAID.

To examine the throughput of concurrent workloads, we used 8 threads in our micro-benchmarks, as seen in Table 6. StarFish throughput is constant across all ten dark-fiber network configurations and both write quorum sizes. The reason is that read requests are handled by either the HE (read hits) or the local SE (read misses) regardless of the write quorum size. Write throughput is the same for both write quorum sizes since it is determined by the throughput of the slowest SE and not by the latency of individual write requests. For all tested workloads, StarFish throughput is within 7% of the performance of a direct-attached RAID.

It is important to note that Starfish resides between the host and the RAID. Although there are no significant performance penalties in the tests described above, the HE imposes an upper bound on the throughput of the system because it copies data multiple times. This upper bound is close to 25.7MB, which is the throughput of reading data from the SE cache by 8 concurrent threads with the HE cache turned off. The throughput of a direct-attached RAID for the same workload is 52.7MB/s.

Table 7 shows that PostMark performance is influenced mostly by two parameters: the write quorum size, and the delay to the SE that completes the write quorum. In all cases the local SE has a delay of 0, and it responds to all read requests. The lowest bandwidth limit to the far SE is 51Mbps and not 31Mbps as in the previous tests, since PostMark I/O is severely bounded at 31Mbps. When $Q = 1$, the local SE responds first and completes the write quorum. This is why the performance of all network configurations with $N = 3$ and $Q = 1$ is essentially the same. When $Q = 2$, the response of the near SE completes the write quorum. This is why the performance of all network configurations with $N = 3$ and $Q = 2$ and with the same delay to the near SE is essentially the same.

An important observation of Table 7 is that there is no performance reason to prefer $N = 2$ over $N = 3$, and $N = 3$ provides higher availability than $N = 2$. As expected, the performance of $N = 2$ is better than the corresponding $N = 3$ configuration. However, the performance of $N = 3$ and $Q = 2$ with a high delay and limited bandwidth to the far SE is at least 85% of the performance of a direct-attached RAID. Another important observation is that StarFish can provide adequate performance when one of the SEs is placed in a remote location, without the need for a dedicated dark-fiber connection to that location.

6.4 Recoveries

As explained in Section 3, StarFish implements a manual failover. We measured the failover time by running a script that kills the HE process, and then sends SNMP messages to 3 SEs to tell them to reconnect to a backup HE process (on a different machine). The SEs report their current sequence numbers for all the logical volumes to the backup HE. This HE initiates replay recoveries to bring all logical volumes up to date. With $N = 3$, $Q = 2$, and one logical volume, the elapsed time to complete the failover ranges from 2.1 to 2.2 seconds (4 trials), and the elapsed time with four logical volumes ranges from 2.1 to 3.2 seconds (9 trials), which varies with the amount of data that needs to be recovered.

Table 8 shows the performance of the PostMark benchmark in a system where one SE is continuously recovering. There is no idle time between successive recoveries. The replay recovery recovers the last 100,000 write requests, and the full recovery copies a 3GB logical volume. Table 8 shows that the PostMark transaction rate during recovery is within 74–98% of the performance of an equivalent system that is not in recovery, or 67–90% of the performance of a direct-attached RAID.

The duration of a recovery is dependent on the the recovery type (full or replay). Table 8 shows that on a dark-fiber network, replay recovery transfers 7.53–

Table 8: Average PostMark performance during recovery. The one-way delay to the local and near SEs in all configurations are 0 and 1 ms, respectively. The Internet configurations have one-way delay to the far SE that is either 23 or 65ms, and bandwidth limit that is either 51 or 124Mbps. Maximum difference from the average transaction rate is 7.3%. PostMark transactions rate of a direct-attached RAID is 73.12 tps.

Q	far SE delay/bandwidth	recovering SE	# conf.	replay recovery PostMark (tps)	replay recovery recovery rate (MB/s)	full recovery PostMark (tps)	full recovery recovery rate (MB/s)	no recovery PostMark (tps)
1	8	local,near,far	3	60.65	8.28–9.89	53.24	7.99–10.12	68.80
1	{23,65}/{51,124}	far	4	63.92	2.52–5.62	60.09	2.97–6.32	67.98
2	8	local,near,far	3	58.41	7.53–9.02	49.64	8.64–9.22	63.85
2	{23,65}/{51,124}	far	4	59.92	2.53–5.79	56.85	2.68–6.64	62.46

9.89MB/s. If the far SE recovers over a 51Mbps Internet link, the average replay recovery rate is about 2.7MB/s. When the Internet bandwidth is 124Mbps and one-way delay is 65ms, our TCP window size of 512KB becomes the limiting factor for recovery speed, because it limits the available bandwidth to *window_size/round-trip time* ($512/0.130 = 3938$KB/s). A separate set of experiments for dark fiber show that replay recovery lasts about 17% of the outage time in the transaction phase of the PostMark benchmark. I.e., a 60 second outage recovers in about 10 seconds.

The duration of full recovery is relative to the size of the logical volume. Table 8 shows that on a dark-fiber network, the full recovery transfer rate is 7.99-10.12MB/s. (Experiments show this rate to be independent of logical volume size.) If the SE recovers over a 51Mbps Internet link, the average full recovery rate is 2.9MB/s, which is similar to the replay recovery rate for the same configuration. When the bandwidth to the recovering SE increases, as seen before, the TCP window becomes the limiting factor.

Table 8 indicates that PostMark performance degrades more during full recovery than during replay recovery. The reason is that the data source for full recovery is an SE RAID that is also handling PostMark I/O, whereas replay recovery reads the contents of the replay log, which is stored on a separate disk.

7 Surprises and dead ends

Here are some of the noteworthy unexpected hurdles that we encountered during the development of StarFish:

- The required set of SCSI commands varies by device and operating system. For example, Windows NT requires the `WRITE AND VERIFY` command, whereas Solaris requires the `START/STOP UNIT` command, although both commands are optional.
- We spent considerable effort to reduce TCP/IP socket latency. We fixed problems ranging from intermittent 200ms delays caused by TCP's Nagle algorithm [22, section 19.4], to delays of several seconds caused by buffer overflows in our memory-starved Ethernet switch.
- Spurious hardware failures occurred. Most SCSI controllers would return error codes or have intermittent errors. However, some controllers indicated success yet failed to send data on the SCSI bus.
- For a long time the performance of StarFish with multiple SEs was degraded due to a shared lock mistakenly held in the HE. The shared lock caused the HE to write data on one socket at time, instead of writing data on all sockets concurrently. This problem could have been solved earlier if we had a tool that could detect excessive lock contention.

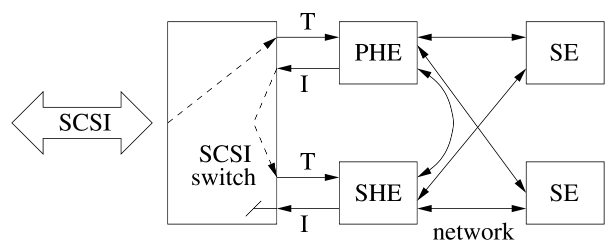

Figure 7: Redundant host element (RHE) architecture.

There was a major portion of the project that we did not complete, and thus removed it from the released code. We have designed and implemented a redundant host element (RHE) configuration to eliminate the HE as a single point of failure, as depicted in Figure 7. It consists of a primary host element (PHE), which communicates with the host and the SEs, and a secondary host element (SHE), which is a hot standby. The SHE backs up the important state of the PHE, and takes over if the PHE should fail. We use a BlackBox SW487A SCSI switch to connect the PHE or SHE to the host in order to perform a transparent recovery of the host element without any host-visible error condition (except for a momentary delay). After spending several months debugging the code, we still encountered unexplained errors, probably due to the fact that the combination of the SCSI switch and the target mode driver was never tested before. In retrospect, we should have implemented the redundant host element with a Fibre Channel connection instead of a SCSI connection, since Fibre Channel switches are more common and more robust than a one-of-a-kind SCSI switch.

8 Future work

Measurements show that the CPU in the HE is the performance bottleneck of our prototype. The CPU overhead of the TCP/IP stack and the Ethernet driver reaches 60% with 3 SEs. (StarFish does not use a multicast protocol, it sends updates to every SE separately.) A promising future addition could be using a TCP accelerator in the HE, which is expected to alleviate this bottleneck and increase the system peak performance.

Other future additions to StarFish include a block-level snapshot facility with branching, and a mechanism to synchronize updates from the same host to different logical volumes. The latter capability is essential to ensure correct operations of databases that write on multiple logical volumes (e.g. write transactions on a redo log and then modify the data).

The only hurdle we anticipate in porting StarFish to Linux is the target mode driver, which may take some effort.

9 Concluding remarks

It is known that the reliability and availability of local data storage can be improved by maintaining a remote replica. The StarFish system reveals significant benefits from a third copy of the data at an intermediate distance. The third copy improves safety by providing replication even when another copy crashes, and protects performance when the local copy is out of service.

A StarFish system with 3 replicas, a write quorum size of 2, and read-only consistency yields better than 99.9999% availability assuming individual Storage Element availability of 99%. The PostMark benchmark performance of this configuration is at least 85% of a comparable direct-attached RAID unit when all components of the system are in service, even if one of the replicas is connected by communication link with long delays and a limited bandwidth. During recovery from a replica failure, the PostMark performance is still 67–90% of a direct-attached RAID. For many applications, the improved data reliability and availability may justify the modest extra cost of the commodity storage and servers that run the StarFish software.

Although we report measurements of StarFish with a SCSI host interface, StarFish also works with a Qlogic ISP 1280 Fibre-Channel host interface.

The availability analysis in Section 4 assumes independent failures. However, a StarFish system may suffer from *correlated* failures due to common OS or application bugs. One way to alleviate it is to deploy StarFish on a heterogeneous collection of servers from different vendors running different OSes (e.g. FreeBSD and Linux).

Source code availability. StarFish source is available from `http://www.bell-labs.com/topic/swdist/`.

Acknowledgments. We would like to thank our shepherd, Robert Watson, and the anonymous referees for their valuable comments. We would also like to thank Justin Gibbs for his help in the development of the Adaptec target-mode driver for FreeBSD and for his help in problem isolation, and Matthew Jacob for his help in enhancing the Qlogic target-mode driver for FreeBSD.

References

[1] AT&T. AT&T data & IP services: Backbone delay and loss, Mar. 2002. Available at `http://ipnetwork.bgtmo.ip.att.net/delay_and_loss.shtml`.

[2] K. P. Birman. Replication and fault-tolerance in the ISIS system. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, pages 79–86, Dec. 1985.

[3] G. Bolch et al. *Queueing Networks and Markov Chains*. John Wiley & Sons, New York, 1998.

[4] EMC Corporation. Symmetrix remote data facility product description guide, 2000. Available at `www.emc.com/products/networking/srdf.jsp`.

[5] E. Gabber et al. Starfish: highly-available block storage. Technical Report Internal Technical Document number ITD-02-42977P, Lucent Technologies, Bell Labs, April 2002.

[6] G. Gibson and R. V. Meter. Network attached storage architecture. *Communications of the ACM*, 32(11):37–45, Nov. 2000.

[7] IBM Corporation. IBM 99.9% availability guarantee program. Available at `www.pc.ibm.com/ww/eserver/xseries/999guarantee.html`.

[8] A. N. S. Institute. The SCSI-2 common access method transport and SCSI interface module, ANSI x3.232-1996 specification, 1996.

[9] J. Katcher. PostMark: A new file system benchmark. Technical Report TR3022, Network Appliance, 1997. Available at `www.netapp.com/tech_library/3022.html`.

[10] N. Lawson. SCSI target mode driver. A part of FreeBSD 5.0-RELEASE, 2003. See `targ(4)` man page.

[11] E. K. Lee and C. A. Thekkath. Petal: Distributed shared disks. In *Proceedings of the 7th Intl. Conference on Arch. Support for Prog. Lang. and Operating Systems*, pages 84–92, Oct. 1996.

[12] E. Levy and A. Silberschatz. Distributed file systems: concepts and examples. *ACM Computing Surveys*, 22(4):321–374, Dec. 1990.

[13] M. MacDougall. *Simulating Computer Systems*. The MIT Press, Cambridge, Massachusetts, 1987.

[14] A. McEvoy. PC reliability & service: Things fall apart. *PC World*, July 2000. Available at `www.pcworld.com/resource/article.asp?aid=16808`.

[15] W. T. Ng et al. Obtaining high performance for storage outsourcing. In *Proceedings of the USENIX Conference on File and Storage Systems*, pages 145–158, Jan. 2002.

[16] B. Nicols, D. Buttlar, and J. P. Farrel. *Pthreads Programming*. O'Reilly & Associates, 1996.

[17] H. Patterson et al. SnapMirror: File system based asynchronous mirroring for disaster recovery. In *Proceedings of the UESNIX Conference on File and Storage Systems*, pages 117–129, Jan. 2002.

[18] L. Rizzo. *dummynet*. Dipartimento di Ingegneria dell'Informazione – Univ. di Pisa. `www.iet.unipi.it/~luigi/ip_dummynet/`.

[19] J. Satran et al. *iSCSI*. Internet Draft, Sept. 2002. Available at `www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-16.txt`.

[20] S. Savage and J. Wilkes. AFRAID – a frequently redundant array of independent disks. In *Proceedings of the Winter 1996 USENIX Conference*, pages 27–39, Jan. 1996.

[21] D. P. Siewiorek. *Reliable Computer Systems: Design and Evaluation*. A K Peters, 1998.

[22] W. R. Stevens. *TCP/IP Illustrated, Volume 1, The Protocols*. Addison-Wesley, 1994.

[23] R. Stewart et al. *Stream Control Transmission Protocol*. The Internet Engineering Task Force (IETF), RFC2960, Oct. 2000. Available at `www.ietf.org/rfc/rfc2960.txt`.

[24] L. Svobodova. File services for network-based distributed systems. *ACM Computing Surveys*, 16(4):353–368, Dec. 1984.

[25] H. Yu and A. Vahdat. Design and evaluation of a continuous consistency model for replicated services. In *Proceedings of the 4th Symposium on Operating System Design and Implementation*, pages 305–318, Oct. 2000.