Evaluating the Impact of Communication Architecture on the Performability of Cluster-Based Services*

Kiran Nagaraja, Neeraj Krishnan, Ricardo Bianchini, Richard P. Martin, Thu D. Nguyen
{*knagaraj, neerajk, ricardob, rmartin, tdnguyen*}*@cs.rutgers.edu*
Technical Report DCS-TR-496
Department of Computer Science, Rutgers University
110 Frelinghuysen Rd, Piscataway, NJ 08854
July 19, 2002

Abstract. *We consider the impact of different communication architectures on the performability (performance + availability) of cluster-based servers. In particular, we use a combination of fault-injection experiments and analytic modeling to evaluate the performability of two popular communication protocols, TCP and VIA, as the intra-cluster communication substrate of a sophisticated Web server. Our analysis leads to several interesting conclusions, the most surprising of which is, under the same fault load, VIA-based servers deliver greater availability than TCP-based servers. If we assume higher fault rates for VIA-based servers because the underlying technology is more immature and programming model more complex, we find that packet errors or application faults would have to occur at approximately 4 times the rate in TCP-based servers before their performability become the same. We also use results from the study to make suggestions for the design of a high-performance and robust communication layer for highly available cluster-based servers. More specifically, we argue that it should use messaging (not a byte stream), single-copy transfers, pre-allocated channel resources, and match the network fabric's fault model.*

1. Introduction

Popular Internet services frequently rely on clusters of commodity computers as their computing platform [7]. The performance and scalability of cluster-based servers have been studied extensively [1, 7, 10]. However, understanding designs for availability, behavior during component failures, and the relationship between performance and availability (and combined *performability*) of these servers has received much less attention.

In this paper, we seek to understand the impact of different communication architectures on the performability of cluster-based servers in the presence of communication-related faults. In particular, our goal is to assess server performability when using two common communication subtrates: the kernel-based Transmission Control Protocol (TCP) and the user-level Virtual Interface Architecture (VIA) [23]. These substrates have been studied extensively in terms of their achievable performance [19, 11], but their availability has not been addressed.

*This work was supported in part by NSF grants EIA-0103722 and EIA-9986046

We seek to understand how the interplay between the application, operating system, and communication stack affects availability. This approach stands in contrast to the rich body of existing work that examines faults arising from noise and packet loss. This line of research has produced results ranging from fundamental limits on encoding in the presence of noise [39], to error detecting codes [18], forward and backward error correction [31], to sophisticated time-out and retry algorithms [6, 24]. However, all the studies fail to address the effect of channel faults on entities outside of the communication systems, e.g., the operating system and applications.

Along the same lines, little is understood about the effect of communication-related resource exhaustion and application faults on server performance and availability. An example of such faults is when the operating system denies memory for the communication layer to use as buffer space, as is possible in TCP. From the perspective of the communication stack, this failure originates from the top of the stack rather than the bottom, as in the case of the channel faults. Understanding the overall system reaction to these non-channel faults is also critical to increasing the availability of next-generation systems. Returning to our example, a communication substrate robust to memory allocation faults allows the service designer to have a memory-stressed node send load-shedding messages. On the other hand, if the communication breaks down because of memory allocation faults, the faulty node must be able to relieve the memory pressure without communicating.

Based on these observations, in this paper we examine the impact of communication hardware, resource exhaustion, and application faults on the availability, performance, and performability of PRESS, a cluster-based locality-conscious Web server [10]. The study relies on our own fault-injection and analysis methodology [34] to examine the differences between TCP and VIA, according to their implementations over the Giganet cLAN network.

Our study illustrates several performance vs. robustness tradeoffs. We show that TCP, in spite of its sophisticated time out and retry, does not provide greater availability than a user-level communication protocol such as VIA in the context of cluster-based services. TCP has two features that limit its applicability in this context: it assumes that packet drops are a sign of transient problems, and it uses a byte-stream abstraction in messaging. The former makes TCP fault detection too slow to be useful, whereas the latter makes the server vulnerable to bad parameters, because a

bad offset or pointer corrupts the entire byte stream after the fault.

We also found that pre-allocation of resources, in particular for memory, can affect the robustness of cluster-based servers. Pre-allocation of buffers and VIA descriptors is essential to maintaining communication during resource stress periods. For VIA, this suggests that dynamic pinning and un-pinning of memory regions exposes servers to resource exhaustion faults.

The benefits of using a user-level communication system such as VIA over TCP ultimately depend on the real fault load. If the designers of a cluster-based server believe that the fault load remains the same regardless of which communication substrate is used, then clearly user-level communication systems provide better performance and higher availability. On the other hand, if their instincts are that a user-level communication substrate might be subjected to a more stressful fault load, because the technology is less mature and the programming model more complex, then a TCP-based implementation may be more desirable. By studying a range of fault loads, we determine the circumstances under which one protocol is preferable over the other.

Given our experience with TCP and VIA substrates, we suggest directions for the design of future communication layers. For example, we believe that a high-performance, robust communication layer should be message-based, single-copy, and pre-allocate all resources.

2. Methodology

We will use the two-phase methodology proposed in [34] to evaluate the performability of PRESS in the presence of faults. In the first phase, the evaluator defines the set of all possible faults, then injects them (and the subsequent recovery) one at a time into a running system. During the fault and recovery periods, the evaluator must quantify throughput and availability as a function of time. For servers such as PRESS, the throughput metric is requests served per second and availability is the percentage of requests served successfully. In the second phase, the evaluator uses an analytic model to compute the expected average throughput and availability, combining the server's behavior under normal operation, the behavior during component faults, and the rates of fault and repair of each component. To parameterize this model, the evaluator defines an expected fault load in terms of the Mean Time To Failure (MTTF) and Mean Time To Repair (MTTR) for each fault. Also, for each fault, the evaluator maps the measurements acquired in phase 1 for each fault to a 7-stage piece-wise linear model of performance over time.

2.1. Phase 1: Measuring Performance Under Single-Fault Fault Loads

There a few tricky issues when injecting faults and recoveries. First, when measuring the server's performance in the presence of a particular fault, the fault must last long enough to allow all stages in the model of phase 2 to be observed and measured. The one exception to this guideline is that a server may not exhibit all model stages under certain faults. In these cases, the evaluator must use his understanding of the server to correctly determine which stages are missing (and later setting the time of the stage in the abstract model to 0). Second, a benchmark must be chosen to drive the server such that the delivered throughput is relatively stable throughout the observation period (except for transient warm up effects). This is necessary to decouple measured performance from the injection time of a fault. Finally, the system should be exercised close to the its peak performance as defined by the bottleneck component. For example, for PRESS, the bottleneck component is the CPU. Thus, we will be running at 90% CPU utilization for all of our experiments.

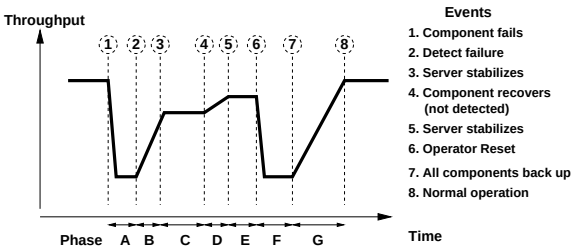

Figure 1. *The 7-stage piece-wise linear model specified by our methodology for evaluating the performability of cluster-based servers.*

2.2. Phase 2: Modeling Server's Performability Under Expected Fault Loads

Figure 1 illustrates our 7-stage model of service performance in the presence of a fault. Time is shown on the X-axis and throughput is shown on the Y-axis. Stage A models the degraded throughput delivered by the system from the occurrence of the fault to when the system detects the fault. Stage B models the transient throughput delivered as the system reconfigures to account for the fault; the system may take some amount of time to reach a stable performance regime because of warming effects. We model the throughput during this transient period as the average throughput for the period. After the system stabilizes, throughput will likely remain at a degraded level because the failed component has not yet recovered, been repaired or replaced. Stage C models this degraded performance regime. Stage D models the transient performance after the component recovers. Stage E models the stable performance regime achieved by the service after the component has recovered. Note that in the figure, we show the performance in E as being below that of normal operation; this often occurs because the system was unable to reintegrate the recovered component or reintegration does not lead to full recovery. In this case, throughput remains at the degraded level until an operator detects the problem. Stage F represents throughput delivered while the server is reset by the operator, whereas stage G represents the transient throughput immediately after reset.

For each stage, we need two parameters: (i) the length of time that the system will remain in that stage, and (ii) the average throughput delivered during that stage. The latter is measured in phase 1. The first is either measured, or is a parameter that must be supplied. For example, the time that a service will remain in stage B assuming that the fault last sufficiently long is typically measured; the time a service will remain in stage E is typically a

supplied parameter.

Sometimes stages may not be present or may be cut short. For example, if there are no warming effects, then stages B, D, and G would not exist. In practice, we set the length of time the system is in the stage to zero. If the assumed MTTR of a component is less than the measured time for stages A and B, then we assume that B is cut short when the component recovers. The evaluator must analyze the measurements gathered in phase 1, the assumed parameters of the fault load, and the environment carefully to correctly parameterize the model.

To simplify the analysis, we assume that faults of different components are not correlated and all fault arrivals are exponentially distributed, so that we can add together the various fractions of time spent in degraded modes. Then, if T_n is the server throughput under normal operation, c the faulty component, T_c^s the throughput of each stage s in Figure 1 during c's failure, and D_c^s be the duration of each stage, our model leads to the following equations for average throughput (AT) and average availability (AA):

$$AT = (1 - \sum_c W_c)T_n + \sum_c \sum_{s=A}^{G} (\frac{D_c^s}{MTTF_c} T_c^s)$$

$$AA = \frac{AT}{T_n}$$

where $W_c = (\sum_{s=A}^{G} D_c^s)/MTTF_c$.

Finally, it is interesting to consider why the denominator of W_c is just $MTTF_c$ instead of $MTTF_c + MTTR_c$. The equation for W_c is correct as it is because we assume fault arrivals for each component are exponentially distributed, and thus memoryless. Our assumption means that when a component fails, another fault could arrive and "queue" at the component. This assumption makes our model tractable but it does differ from a fault model where a faulty component cannot become faulty again before it has been repaired. The impact on our model is that we compute the fraction of downtime as $\frac{MTTR}{MTTF}$, not as the more typical $\frac{MTTR}{MTTF+MTTR}$. In practice, because $MTTF >> MTTR$ the numerical impact of this difference in assumptions is minimal. However, a full investigation of discrepancies between these assumptions is beyond the scope of this work.

3 The PRESS Server

PRESS is a highly optimized yet portable cluster-based locality-conscious WWW server that has been shown to provide good performance in a wide range of scenarios [10, 11]. Like other locality-conscious servers [35, 1, 9, 4], PRESS is based on the observation that serving a request from any memory cache, even a remote cache, is substantially more efficient than serving it from disk, even a local disk.

In PRESS, any node of the cluster can receive a client request and becomes the *initial node* for that request. When the request arrives at the initial node, it decides, based in the content, whether to service the request itself or forward the request to another node, the *service node*.

A forwarded request is handled in a straightforward way by the service node. If the requested file is cached, the service node simply transfers it to the initial node. If the file is not cached, the service node reads the file from its local disk, caches it locally, and finally transfers it to the initial node. Upon receiving the file from the service node, the initial node sends it to the client. The initial node does not cache the file received from the service node to avoid excessive file replication across the cluster.

To intelligently distribute the HTTP requests it receives, each node needs locality and load information about all the other nodes. Locality information takes the form of the names of the files that are currently cached, whereas load information is represented by the number of open connections handled by each node. To disseminate caching information, each node broadcasts its action to all other nodes whenever it replaces or starts caching a file. To disseminate load information, each node piggy-backs its current load onto any intra-cluster message.

Communication architecture. PRESS is comprised of one `main` coordinating thread and a number of helper threads used to ensure that the `main` thread never blocks. The helper threads include a set of `disk` threads used to access files on disk and a pair of `send/receive` threads for intra-cluster communication.

PRESS can use either TCP or VIA for intra-cluster communication. The TCP version basically has the same structure of its VIA counterpart; the main differences are the replacement of the VI end-points by TCP sockets and the elimination of flow control messages, which are implemented transparently to the server by TCP itself.

Reconfiguration. PRESS is often used (as in our experiments) without a front-end device, exposing the IP address of the cluster nodes to clients. To prevent clients from continuously experiencing failed requests (due to a node failure, for instance), some versions of PRESS implement node failure detection, temporary recovery through IP address take-over, and final recovery with the re-integration of recovered nodes. The detection mechanism when TCP is used for intra-cluster communication can be connection or heartbeat-based. In the heartbeat-based approach, PRESS employs periodic heartbeat messages. To avoid sending too many messages, we organize the cluster nodes in a directed ring structure. A node only sends heartbeats to the node it points to. If a node does not receive three consecutive heartbeats from its predecessor in the ring, it assumes that the predecessor has failed. In the connection-based approach, a node assumes that another node has failed if the TCP connection between them is broken. In this implementation, nodes are also organized in a directed ring, but only for recovery purposes.

The fault detection when VIA is used for intra-cluster communication is also connection-based. PRESS relies on our VIA implementation to maintain node and communication up/down information. Any detected faults cause the corresponding connections to be terminated/broken. Again, nodes are organized in a directed ring for recovery purposes.

Temporary recovery is implemented by simply excluding the failed node from the server or by having the successor node take-over the IP address of the failed node. (In our experiments, we eliminated the IP take-over part of the recovery process, so we do not discuss IP take-over further.) Multiple node failures can occur simultaneously. Every time a failure occurs, the ring structure is

Version	Main Features	Expected Behavior	Throughput
TCP-PRESS	TCP used for intra-cluster communication; connection breaks are used as trigger for reconfiguration	Performance may suffer in the presence of faults because TCP connection timeouts are typically lengthy	4965 reqs/sec
TCP-PRESS-HB	TCP used for intra-cluster communication; loss of heartbeat messages are used as trigger for reconfiguration	Faster response to faults but may give false positives if communication of heartbeats is delayed	4965 reqs/sec
VIA-PRESS-0	VIA used for intra-cluster communication; connection breaks are used as trigger for reconfiguration	Outperforms the TCP versions but may be more vulnerable to user-level errors	6031 reqs/sec
VIA-PRESS-3	VIA used for intra-cluster communication; remote memory writes used in all messages; connection breaks are used as trigger for reconfiguration	Outperforms VIA-PRESS-0, but remote memory writes can diffuse pointer and message size faults.	6221 reqs/sec
VIA-PRESS-5	VIA used for intra-cluster communication; remote memory writes used in all messages; zero-copy used for data transfers; connection breaks are used as trigger for reconfiguration	Gives best performance but remote writes can diffuse faults and zero-copy requires dynamic page pinning, making it more vulnerable to OS faults.	7058 reqs/sec

Table 1. *Versions of PRESS available for study, their differences, expected impact on performance and availability, and peak throughput.*

modified to reflect the new cluster configuration.

The second and final step in recovery is to re-integrate a recovered node into the cluster. When IP take-over is not in effect and the intra-cluster communication protocol is TCP, the rejoining node broadcasts its IP address to all other nodes. The currently active node with lowest id responds by informing the rejoining node about the current cluster configuration and its node id. With that information, the rejoining node can reestablish the intra-cluster connections with the other nodes. After each connection is reestablished, the rejoining node is sent the caching information of the respective node. When the intra-cluster communication is done with VIA, the rejoining node simply tries to reestablish its connection with all other nodes. As connections are reestablished, the rejoining node is sent the caching information of the respective nodes.

Versions. Several versions of PRESS have been developed in order to study the performance impact of different communication mechanisms [11]. Table 1 lists the versions of PRESS that we consider in this paper. For each version, we summarize its main characteristics, their expected impact on performance and availability, and their near-peak throughputs on our 4 cluster nodes. The throughputs of the various versions of PRESS will be compared against throughput when various faults are injected into the cluster.

All PRESS versions cooperate in caching files, but differ in terms of their approach to detecting failed nodes, and the performance of their intra-cluster messaging. TCP-PRESS and the VIA-PRESS versions use connection breaks to detect node failures, whereas TCP-PRESS-HB uses heartbeat messages.

PRESS uses intra-cluster messages for request forwarding, dissemination of caching information, and transfer of file data. When using VIA for intra-cluster communication, flow-control messages are also used. In TCP-PRESS and TCP-PRESS-HB, all message types involve data copies on both sides and interrupt-driven message reception. VIA-PRESS-0 also utilizes regular messages, but they are sent directly from user-space. To avoid the overhead of receiver interrupts in VIA-PRESS-0, VIA-PRESS-3 utilizes remote memory writes and polling in all messages. Remote writes are implemented by allocating buffers at each node for each other node. Polling is done by looking at message sequence numbers stored at the last position of each (fixed-size) buffer entry. Processors poll for messages at the end of the main server loop. VIA-PRESS-5 improves performance further by avoiding the large copies involved in file data transfers. The copy at the receiver is eliminated by sending the data to the client right out of the communication buffer. The copy at the sender is eliminated by transferring the data directly from the sender's file cache. For that, all the pages corresponding to cached files must be pinned in physical memory.

4. Fault Injection and Fault Model

We use Mendosus [29], a cluster-based fault injection and network emulation infrastructure, to study the behavior of PRESS in the presence of faults. Mendosus is implemented completely in software, via a set of kernel modules and user-level libraries. The service being studied runs live on top of Mendosus; faults are injected in real-time in order to measure the service's live response to faults.

Mendosus can inject a wide range of faults. The fault model that we consider for this study includes physical failures of the networking hardware, end-node crashes, resource exhaustion failures within the operating system, and application faults, focusing particularly on the large class of faults arising from inappropriate parameters passed to the communication interfaces. We chose to consider operating system resource exhaustion, in particular, memory exhaustion, in addition to the hardware faults because previous studies have shown that memory exhaustion may be a dominant factor in the "aging" of operating systems [16, 43]. Likewise, we chose the specific application-level faults because a previous study suggests that they are the dominant categories of application errors [13]. We could have assumed a completely generic application fault model, such as the crashing of a process. However, this would not have brought out the different ways that these errors propagate through the different communication layers. Table 2 lists the specific faults that we consider.

Fault Category	Faults	Example Error Source
Network hardware failures	Link failure	Faulty cable, accidental unplugging, mis-configuration
	Switch failure	Power failure, software bug, mis-configuration
Node faults	Node crash	Operator error, OS bug, hardware failure, power failure
	Node hang	OS bug, OS recovering after killing faulty process
Resource exhaustion	Kernel memory allocation failure	System low on (kernel) memory / out of virtual address space
	Memory locking failure	Out of pinnable physical memory
Application faults	Application hang	Application bugs, paging effects
	Application crash	Application bugs, operator mis-termination
	Bad parameters: NULL pointers, off-by-N data pointer, off-by-N size	Uninitialized pointers, logical error, pointer corruption, stale memory handle (RDMA)

Table 2. *Faults to be injected and possible sources of these faults.*

4.1 Network Hardware Failures and Node Faults

We consider fail-stop failures of links and switches, components that typically comprise the intra-cluster network of today's server cluster. (In this study, NIC failures are considered a source of node failures and are accounted for in node crashes and hangs.) We only consider fail-stop failures as we are more concerned with application-level effects, rather than channel errors that are typically dealt with in the communication substrate or protocol. We refer the reader to [34] for a description of how we effect these faults.

Mendosus can inject three types of node faults: hard reboot, soft reboot, and node freeze. All can be either transient or permanent, depending on the specified fault load. The node crash fault type we study here corresponds to the hard reboot fault. Again, we refer to the reader to [34] for our implementation of these faults.

4.2 Resource Exhaustion Faults

Transient situations on a server node running many processes can at times create a shortage of resources (e.g., memory, disk space, file descriptors, etc.). This can happen when a buggy process asks for too many resources or a bug in the kernel leads to leakage. In this study, as already mentioned, we only consider exhaustion of memory.

We implemented two types of memory failures. First, failure to allocate `skbufs` within the kernel for a specified duration of time. This simulates situations when the kernel can no longer allocate more kernel memory. We implemented this by trapping the calls to `skbuf` allocations for intra-cluster communication and returning an error to the caller.

The second failure involves running out of memory on a pin-down request. This again simulates situations such as a buggy (or just greedy) process pinning an unusual amount of memory, causing shortage of pinnable physical pages for other processes. Kernels usually limit the number of pinnable pages to only a fraction of the available physical pages (e.g., the Linux 2.2x kernels limit this amount to half the number of physical pages) to maintain a safe buffer for possible future allocations. To implement this fault, we had to modify the cLAN driver since the driver directly manipulates the page table when a client registers some memory with it. In particular, we modified the cLAN driver to dynamically adjust the threshold above which requests to lock memory were returned with error status.

4.3 Application Faults

We implement two classes of application faults. First, generic errors that we do not model in detail lead to either a process crash or process hang. These faults are effected by a user-level daemon running on each node. For our study of PRESS, the server process on each node is started by the daemon. An application hang fault is injected by having the daemon send a SIGSTOP to the server process. The process can be restarted if the fault is transient by sending a SIGCONT to it. A crash is injected by killing the application process.

The next class of faults models actual application level bugs. Again, we thought that studying actual application bugs would be more enlightening than just using the above coarse-grain faults because it would give us insight into the propagation of such errors through the different communication subsystems.

The class of "real" errors involve bad parameters passed to calls into the communication subsystem. We implement the injection of these faults by interposing a software layer between the application and the normal communication library. Our layer traps specific calls, modifies one or more parameters, and then passes the call to the communication library.

Specifically, we injected faults into the `send()` and `recv()` calls for socket-based communication, and the `VipPostSend()` and `VipPostRecv()` calls to the cLAN VIPL library. For the VIA calls, the parameters were actually inside VIA descriptors, and so we modified the corresponding fields within a descriptor before passing it up to VIPL library. We considered three types of corrupted parameters: passing of NULL pointers, off-by-N for data pointers, and off-by-N for buffer sizes. N in all cases were in the range of 0 to 100 bytes, which has been observed by Sullivan and Chillarege to be the dominant range for offset errors [41].

5 PRESS Behavior Under Single-Fault Loads

We now apply the first phase of our methodology to evaluate the performability of PRESS. In particular, we measure and explain the behavior of all 5 versions of PRESS under single faults injected in isolation.

5.1 Experimental Setup

In all experiments, we run a four-node version of PRESS on four 800 MHz PIII PCs, each of which is equipped with 206 MByes of memory and 2 10,000 rpm SCSI disks. Nodes are interconnected by a 1 Gb/s cLAN network. We can communicate with TCP or VIA over this network. PRESS was allocated 128 MBytes on each node for its file cache; the remainder of the memory was sufficient for the operating system and the server code, such that no swapping took place during the experiments.

The workload for all experiments is generated by a set of clients running on separate machines. To stress the communication aspect of PRESS, our experiments only involve static content and the entire set of documents is replicated at each node. The client machines are connected to PRESS by the same cLAN network that connects the nodes of the cluster. Using a single network for communication is not at all a problem. The total network traffic does not saturate any of the cLAN network interfaces, links, or the switch, and so the interference between the two classes of traffic is minimal in our setup. Furthermore, our fault-injection infrastructure allows us to differentiate between intra-cluster communication and client-server communication when injecting network-related faults. Thus, the clients are never disturbed by faults injected into the intra-cluster communication.

Each client generates load by following a trace gathered at Rutgers; we chose this trace from several that Carrera and Bianchini previously used to evaluate the performance of PRESS, because it has the largest working set [10]. We modified the file set so that all files have the same size (the average size of the original file set). This modification was necessary to ensure that PRESS delivers a stable throughput throughout the trace.

To achieve a particular load on the server, each client generates a stream of requests according to a Poisson process with a given average arrival rate. Each request is set to time out after 2 seconds if the connection cannot be completed, and to time out after 6 seconds if, after successful connection, the request cannot be completed.

5.2 Network Hardware Failures

In this section, we study the behavior of PRESS for faults in the intra-cluster communication hardware. First we discuss the effects of the faults on each version, and then draw some conclusions on the differing behavior. Because of space constraints, we will only show results for a subset of the faults injected; the reader can find all our data at http://www.panic-lab.rutgers.edu/Research/mendosus/.

Figure 2 shows the effects of a single transient link failure. We do not show the graphs for VIA-V0 and VIA-V3 because they are essentially the same as that of VIA-V5. TCP-PRESS exhibits its expected behavior under link failure, by stalling for the period of the fault. The TCP protocol on the cooperating nodes keeps trying to re-send packets across the faulty component, causing the filling up of communication queues on all nodes, thus dropping the throughput to zero until slightly after the component recovers and the messages start flowing again. Note that the fault does not last long enough for TCP timeouts to occur. These timeouts tend to be very long, on order of 10-15 minutes.

In contrast, TCP-PRESS-HB detects the fault in a very short time and reconfigures. The reason is that the heartbeat messages lost over the faulty link cause the other nodes to assume that the unreachable node is down. This splinters the cluster into 3 cooperating nodes and 1 independent node. The detection and recovery durations correspond to a failure detection threshold of 15 seconds (3 heartbeats) used by the heartbeat code.

Similar to TCP-PRESS-HB, the VIA versions detect the failure almost instantaneously because the connections to the unreachable node all break. VIA versions splinter into 3 cooperating nodes and 1 independent node just as TCP-PRESS-HB. Interestingly, TCP-PRESS-HB and the VIA versions do not reconfigure back into a single cluster once the link returns to normal operation. This surprising behavior arises from a mismatch between the fault model assumed by PRESS and the actual fault. PRESS assumes that nodes fail but links and switches do not. Thus, reconfiguration only occurs at startup and on loss of 3 heartbeats; nodes do not merge again after partitions. Return to normal operation thus requires the intervention of an administrator to restart all but one of the sub-clusters. This, in effect, makes these versions *less* available than the basic TCP-PRESS in the face of relatively short transient faults.

5.3 Node Faults

Figure 3 shows the effects of a hard reboot fault, i.e., a node crash. Again, we do not show the graphs for VIA-PRESS-0 and VIA-PRESS-3 because they are similar to that of VIA-PRESS-5.

Because it is not capable of immediately detecting node failures, TCP-PRESS grinds to a halt while the faulty node is down; all communication queues get filled up with messages for the crashed node. When the crashed node comes back up, an interesting timing problem develops. The recovered node tries to rejoin the cluster, as Mendosus starts another PRESS process automatically, but is not able to. The reason is that the other nodes do not detect the reboot until a little while later. During this period, all the rejoin messages sent by the recovered node are disregarded by the rest of the cluster. After the recovered node gives up trying to rejoin, the other 3 nodes detect the termination of the connections and form a group of cooperating servers.

TCP-PRESS-HB and the VIA versions behave exactly as expected and almost identically. In TCP-PRESS-HB the node crash is detected by the heatbeat protocol, whereas in the VIA versions the connections to the faulty node break. After detection, the 3 remaining nodes continue cooperation. After the node reboots and Mendosus restarts the PRESS process, the node is able to rejoin the cluster and throughput returns to the normal level.

Results for node hangs are similar except that TCP-PRESS correctly deduce that no fault has occurred (although throughput does fall to zero while everyone waits for the hung node) while TCP-PRESS-HB incorrectly decides that a fault has occurred and splinters into two sub-clusters.

5.4 Resource Exhaustion Faults

In this section we consider the effects of communication-related resource exhaustion within the operating system (in particular, within the memory subsystem). Figure 4 shows the effect

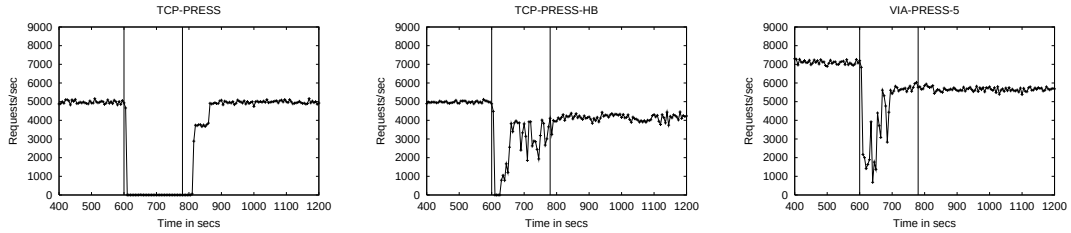

Figure 2. *Throughput of PRESS when a link failure is injected.*

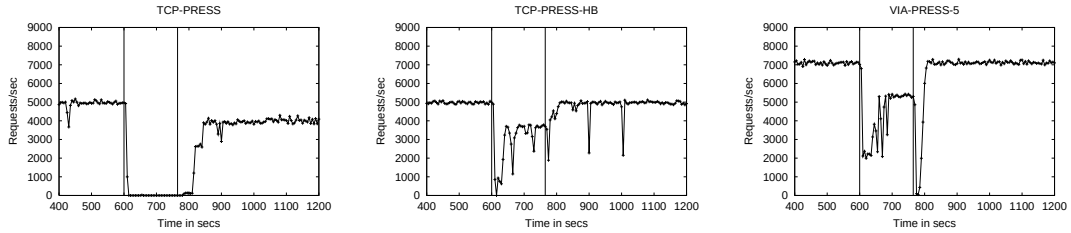

Figure 3. *Throughput of PRESS when a node crash is injected.*

of kernel memory exhaustion, which causes failures in the allocation of communication buffers. The throughput of TCP-PRESS drops to zero during the fault, since the stalling of communication to the faulty node freezes the entire cluster. The TCP packets arriving at the faulty node are dropped, whereas packets leaving the node get queued up within the operating system, waiting for buffer allocations. In contrast, TCP-PRESS-HB splinters into 3 cooperating nodes and a singleton after 3 heartbeats from the faulty node are not received. The VIA versions perform pre-allocation of most network resources during service initiation. This makes them less vulnerable to dynamic fluctuations in memory resources. Since they did not show any degradation in performance during the fault period, we do not present their graphs here.

VIA-PRESS-5 is succeptible to pinnable memory exhaustion faults, however. The zero-copy messaging in this version requires memory to be registered with the VIA library, which in turn pins the registered region. Cache replacements cause memory to be unpinned (file that is being replaced) and later pinned (file that is entering the cache). When a node in VIA-PRESS-5 is unable to pin memory, it drops files from its cache to free up memory. The cache misses arising from these dropped files degrade the throughput during the fault period, as also shown in Figure 4.

5.5 Application Faults

Figure 5 shows the behavior of PRESS when a *null* value is passed as the data pointer to the send API. The TCP versions detect this fault synchronously and return an EFAULT error code to the invoker. The VIA versions diagnose the error code as a fatal error, and terminate themselves. The recovery, achieved by restarting the application, reintegrates the faulty server into the cluster. In VIA-PRESS-0 the asynchronous error reporting through error status in completed descriptors achieves the same effect as the TCP versions. However, in the remote memory write-intensive VIA-PRESS-3 and VIA-PRESS-5, the error is reported on both nodes involved in the remote operation. This causes the termination of 2 nodes as per PRESS's fail-fast approach. However, the restart of the PRESS processes returns the performance from an extremely degraded state to a normal throughput.

We also considered off-by-N values for data pointer and the data size parameters. In these experiments, we observe that either, only the sender or only the receiver node experience an error as a result of these incorrect parameters. In contrast, the fault is reported at both ends of the communication in the remote memory write versions. In fact, remote memory writes poses a greater risk as *'valid'* bad parameters result in corruption of data (and possibly meta-data) on the remote node as well. In our current study though, we have not quantified this additional risk. We believe that this demands a deeper statistical study that is beyond the scope of this paper.

6 Performability of the PRESS Versions

We now proceed to the second phase of our methodology to evaluate the performability of the different PRESS versions. We first examine performability assuming the same fault load for all versions of PRESS. Then, we also consider what happens if we assume that the VIA versions of PRESS experience more stressful fault loads; there are two reasons why it is interesting to consider this case: (1) programs written on user-level communication subsystems such as VIA may be more buggy because VIA forces the programmer to deal with communication issues such as buffer management and flow control that, in TCP, are implemented by the communication protocol, and (2) user-level communication subsystems such as VIA are still relatively immature, possibly leading to a higher fault rate because of hardware and/or firmware bugs.

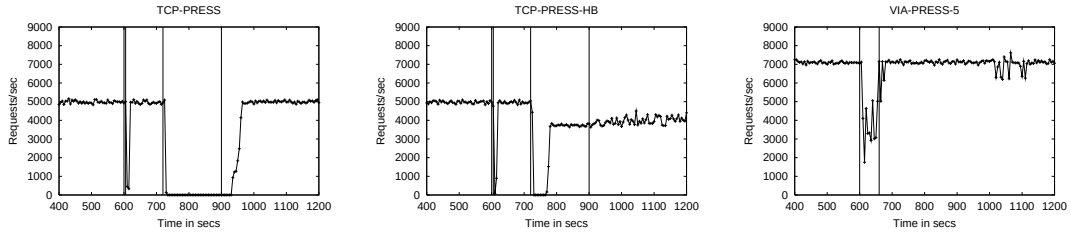

Figure 4. *Throughput of TCP-PRESS and TCP-PRESS-HB when a kernel memory exhaustion failure is injected and throughput of VIA-PRESS-5 when a pinnable memory exhaustion failure is injected.*

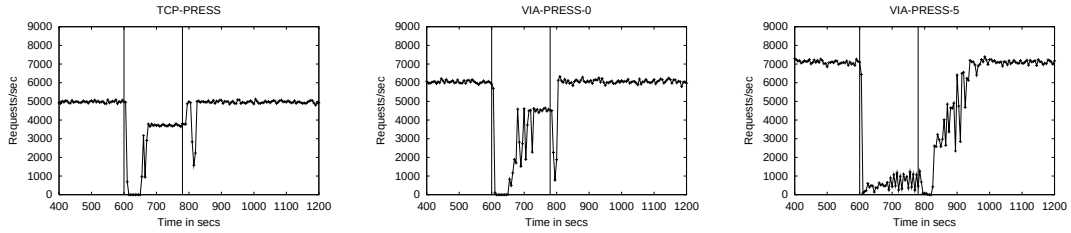

Figure 5. *Throughput of PRESS when a NULL pointer fault is injected.*

Fault	MTTF	MTTR
Link down	6 months	3 minutes
Switch down	1 year	1 hour
Node crash	2 weeks	3 minutes
Node freeze	2 weeks	3 minutes
Memory pinning failure	61 days	3 minutes
Memory allocation failure	61 days	3 minutes
Process crash	–	3 minutes
Process hang	–	3 minutes
Bad parameters - null pointer	–	3 minutes
Bad parameters - off-by-N data pointer	–	3 minutes
Bad parameters - off-by-N size	–	3 minutes

Table 3. *Failures and their MTTFs and MTTRs.*

6.1 Fault Load

Table 3 gives the initial fault load used to compare the performability of the different versions of PRESS. Recall that to make the modeling tractable, we assume that faults in different components are not correlated and all fault arrivals are exponentially distributed. We have done our best to derive meaningful parameters from the available data [41, 13, 16, 27, 42, 43, 20]. However, data is sparse, particularly for application-level errors. Thus, we examine performability for a range, once per day to once per month, of MTTFs for application level faults. In addition, because we have multiple classes of errors, we divided the fault rate between these errors according to the distribution observed by Chillarege et al. [13]. This led to approximately the following ratio: process crash 40%, process hang 40%, null pointer 8%, off by N data pointer 9%, off by N size 2%.

6.2 Evaluation Metrics

Our model computes two metrics to evaluate each server. The first is the unavailability, which is the average fraction of requests dropped. We use unavailability instead of availability because it is easier to reason about changes in unavailability compared to availability. For example, it is quite natural to call a system with an unavailability of 1% as "twice as good" as on which has an unavailability of 2%. However, the relationship between systems with a 98% and 99% availability is not so intuitive.

Combining the throughput and availability metrics into a *performability* measure for each version of PRESS allows us to capture the performance differential as well. Because there are no established performability metrics in the server context [32], we define performability as the average throughput multiplied by a measure of availability. Our availability measure is a log-scaled ratio of how well or poorly each server compares to a hypothetical, ideal benchmark system. In our case, the ideal benchmark delivers "five nines", i.e. an availability of 99.999% or five minutes of unavailability per year. Because we operate on a log ratio, each decimal point of availability results in factor of 10 improvement in the availability ratio.

6.3 Performability Under the Same Fault Load

Figure 6 shows (a) the modeled unavailability and (b) performability of the different versions of PRESS when scaling the application fault rate from once per day to once per month. Figure 6(a) also shows the contribution of each fault type to overall unavailability. Overall, perhaps surprisingly, the availability of all three VIA versions is *better* than that of the two TCP version (albeit only slightly better). While non-intuitive, this result arises

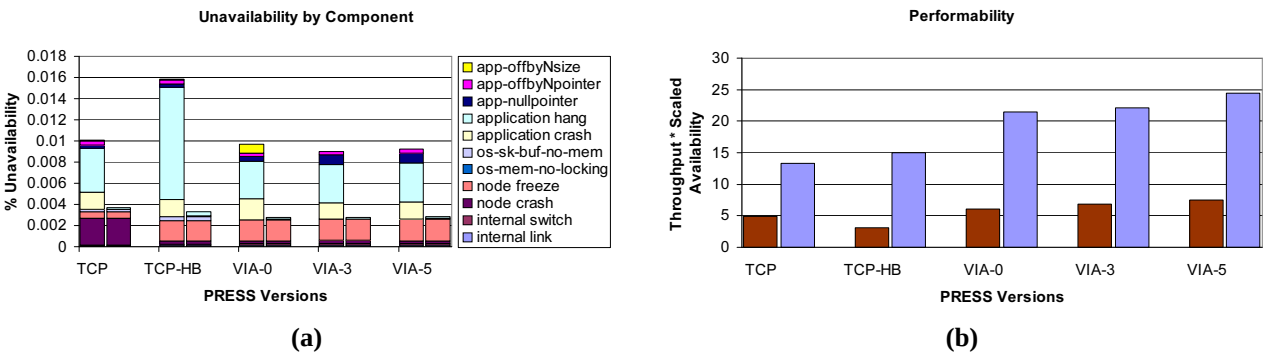

Figure 6. *The modeled (a) unavailability and (b) performability of the 5 versions of PRESS. For each group of 2 bars, the left is for an application fault rate of 1 per day and the right is for a rate of 1 per month.*

because the VIA fault reporting is more accurate for the cluster environment. The two TCP versions only use end-to-end information to try and detect failures and can be led astray. On the one hand, TCP-PRESS takes too long to detect a number of the faults because it assumes that lost packets are due to transient congestion, not component faults. TCP-PRESS-HB, on the other hand, sometimes concludes too quickly that lost messages (heartbeats) translate to faults, rather than delay because of other reasons.

Of course, the above discussion does *not* mean that heartbeats are useless. The reason that heartbeats were less useful than one might expect is that PRESS does not reconfigure when faults do not lead to one or more process crashes. To make heartbeats more effective, one needs to implement a rigorous membership algorithm that can repair the group membership correctly when loss of heartbeats leads to the incorrect splintering of the cluster.

At low application fault rates, a second reason why the VIA versions exhibit better availability than the TCP versions is that they are not vulnerable to operating system resource exhaustion errors. Since they preallocate resources at startup time, the VIA versions continue to operate well even when the operating system runs out of memory.

We also observe that there is little difference in the availability of the VIA versions. This is perhaps not unexpected because, for the vast majority of application errors, the effect of the fault is the same: the hanging or crashing of an process. This is also true of the node faults. It is interesting to note, however, that there are noticeable differences in the contribution to unavailability of the different specific application fault types at high application fault rates. The reason is that these faults lead to a few differences in the behavior of the different versions. At lower application fault rates, careful analysis shows that VIA-PRESS-5 gives the worst availability because the dynamic pinning of memory makes it slightly vulnerable to resource exhaustion faults. However, since the VIA versions manage their own communication resources, VIA-PRESS-5 is able to adapt to the resource exhaustion faults better than the TCP versions; in particular, it releases some of the memory that it had previously pinned to free up the needed resources. These observations lead us to surmise that different dependency on user-level communication mechanisms would likely lead to different availability if only we could more accurately emulate real application and operating system faults, allowing them to propagate through the communication subsystem.

Finally, while the above differences are interesting, perhaps the most significant conclusion is that availability is uniformly terrible. With an application fault rate of once per day, availability is only in the 99% range, and of course, the effect of application crashes dominate. Even at a fault rate of once per month, availability does not make to 99.9%. Since the differences in availability are relatively small, the performability graphs show the expected results: the highest performing version of PRESS leads to the best performability.

6.4 Pessimistic Fault Loads for VIA Versions

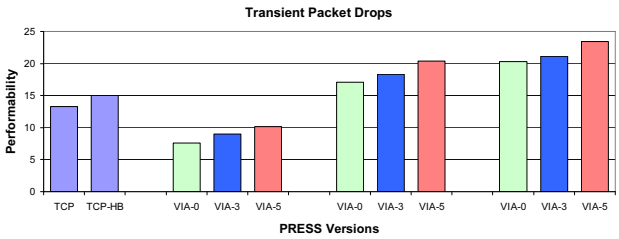

Figure 7. *Comparison of performability in the presence of transient packet drops. For TCP, we model no effect. For VIA we show the impact of packet drop rates, which re-set the channel, of 1 per day (left), 1 per week (center), 1 per month (right)*

We now look at what happens if we assume a more pessimistic fault load for the VIA versions than the TCP versions. First, we look at what happens when there is transient packet loss. While the VIA specification states that transient packet loss should be extremely rare, in practice, it may occur more often because of hardware and/or software bugs [44]. Thus, we model transient packet loss as application process crashes, assuming that the packet loss would be reported as an error, leading to the process terminating itself. On the other hand, since much of TCP's robustness comes

from tolerating transient packet drops, we assume that such faults have no effect on the TCP versions of PRESS.

Figure 7 compares the performability of the different versions as the transient packet loss rate is varied from 1 per day to 1 per month. These faults are in addition to the fault rates in Table 3. Observe that the performability of the TCP versions are roughly comparable to those of the VIA versions when the packet loss rate is about 1 per week; TCP wins if the packet loss rate is greater than 1 per week and loses if the rate if less than 1 per week. These results imply that if the designer believes that the LAN/SAN is relatively immature, then it is perhaps better to use TCP and sacrifice some performance. On the other hand, if the LAN/SAN is relatively mature technology with little reason to suspect a high rate of packet loss, then it is worthwhile to bypass the software overhead of TCP.

Next, we examine what happens if, by making direct use of VIA, the server software is more buggy, due to the more complex and unfamiliar programming model of VIA. Figure 8 compares the performability of the different PRESS versions as the VIA versions are subjected to increasing rates of application faults. These results show that the performability is comparable when the additional application fault load on the VIA versions is around 1 per week. Again, the implication is that if a designer is confident that his programming team can deal well with the additional complexity of programming directly on VIA, leading to little additional software bugs, then choosing VIA would give the best performance and performability. On the other hand, if the implementation team is inexperienced or just does not have enough time, perhaps going with TCP is the best option, giving up some performance but also reducing the complexity of the programming task.

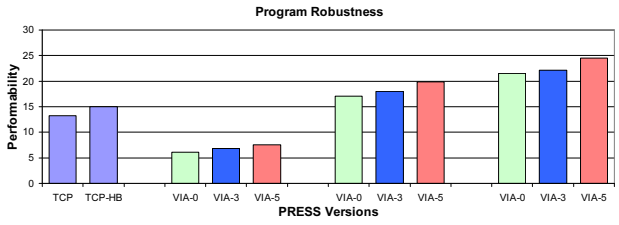

Figure 8. *Comparison of performability in the presence of software bugs. For TCP we model 1 fault per month. The VIA fault rates scale from 1 per day (left), 1 per week (center), 1 per month (right)*

Finally, we consider what happens when there are occasional system crashes in the VIA networking subsystem due to hardware or firmware bugs[1]. This is a plausible scenario, because leading edge technologies such as VIA are often less mature (and so more buggy) than standard technologies that have been around longer and have much larger user bases. We model such a system bug as a switch crash. Figure 9 shows the impact of adding on failures

[1]Note that while we are running the TCP versions over the same hardware as the VIA versions, what we are assuming here is that there is an alternative networking technology, Gigabit Ethernet, that would be more reliable than the VIA hardware and yet would perform approximately as well.

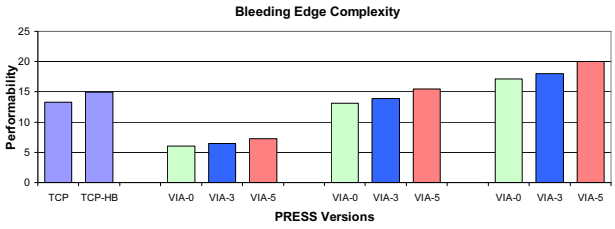

Figure 9. *Comparison of performability in the presence of system faults due to immaturity of communication substrate. For TCP, we model no errors. The VIA rates are 1 per week (left), 1 per month (center), 1 per 3 months (right)*

due to system bugs. The tradeoffs here are similar to the previous variations.

Figure 10 compares the performability of the different versions of PRESS assuming a combination of added faults for the VIA versions: a packet drop rate of 1 per month, added application fault rate of 1 per every 2 weeks, and system failure of 1 per month. Observe that under this fault load, the performability of two of the three VIA versions are well below that of the TCP-HB version. Thus, we conclude that the performance advantage of a user-level network such as VIA really depends on how mature the product is and whether the programmer can use the exported API without introducing additional software bugs.

7 Discussion

In the course of our study, we uncovered a number of important performance vs. reliability tradeoffs in communication stacks of high availability systems. In this section, we discuss some of these lessons and speculate on the properties of such a communication layer.

A transport stack must match the underlying fault model with the class of faults that occur in the network fabric. Our results show that a mismatch between the stack and fabric fault models has a tremendous impact on overall system availability. When running over fabrics that drop packets in the face of queue overruns, and in environments with otherwise uncontrolled bandwidth usage, TCP-style timeout and retry is appropriate. On the other hand, when running over SAN-style networks with hop-by-hop flow control, and where packet losses are at a minimum, a VIA fail-stop model is appropriate. Future protocol stacks may be able to determine which fault-model to use dynamically, but in the near future designers should pick a transport layer that best maps to the fabric's actual fault characteristics.

We also found it useful for each component in the communication path to report errors immediately and all the way to endpoints. In the cluster context, LANs that discard bad packets are undesirable because they inhibit early fault detection. Some SANs do forward bad packets to the final endpoint [5, 22], but this information is often not used by the transport protocols to initiate error recovery. Turning to congestion, using packet drops to signal con-

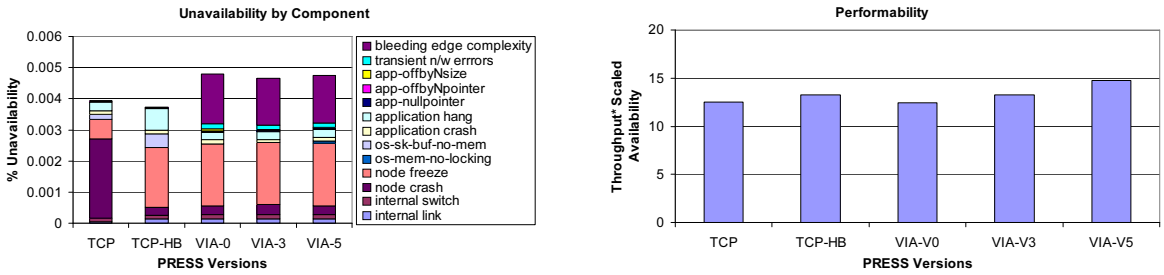

Figure 10. *Comparison of performability assuming a more pessimistic fault load for VIA.*

gestion [15], while increasing network throughput, is a very slow method of fault detection. In a LAN that drops packets, schemes that actively signal congestion [26] may result in higher availability.

We found that copying can be a critical fault-containment point for communication layers. While our rates of corrupted packets due to overruns was low, we also note that we could not detect overruns for VIA-PRESS-3 and VIA-PRESS-5.

Our last lesson concerns pre-allocation of memory resources. In general, if there are enough resources these should be pre-allocated during channel set-up. While this increases overall memory usage, such a pre-allocation strategy gives the designer more flexibility in recovering from periods of memory stress.

Finally, we observe that in order to better compare systems, much more empirical measurement of actual MTTR and MTTF of real systems is needed. While we did base many of our modeled MTTFs and MTTRs on measured systems, data were quite sparse and it was unclear how general they were.

8 Related Work

There has been much work on examining the robustness of communication protocols, mostly in the context of packet loss. A seminal work on the subject [37] makes the argument that only the endpoints have enough semantic information to handle faults. However, the work does not discuss the appropriate error recovery mechanisms needed upon a fault, nor does it address how intermediate nodes of the system should report faulty conditions to the endpoints. Indeed, our experience hints that a conclusion often drawn from this work — that intermediate nodes should do very little error detection and recovery — is mistaken. Intermediate nodes should rather provide fast, accurate error reporting to endpoints.

There is an enormous volume of work on TCP's tolerating of transient packet loss and probing the network for bandwidth; a few of the works include [6, 14, 21, 24]. All these works, however, assume that packet loss is due to congestion, where retransmission has a likely probability of success. Two areas where the fault model implicit in TCP breaks down are in the wireless networks and System Area Networks (SANs). In wireless networking, packet loss often implies channel degradation, so the recovery actions for TCP should be retransmit quickly [3]. In the SAN context, researchers argued that faults signal catastrophic failure [5, 38, 45], requiring human intervention. However, they also argue the fault rates of these networks are very low.

The LAN and WAN networking communities have recognized that byte stream abstraction is not always appropriate and so have proposed many alternative messaging based protocols. A few of the originals include [12, 36, 40]. The key difference from the MPP and SAN networks models is that these protocols, like TCP, viewed packet loss to signal congestion.

There has been extensive work in analysing faults and how they impact systems [17, 41, 28]. However, the focus of these studies was not on the communication system. Studies benchmarking system behavior under fault loads include [25, 30]. However, these works do not provide a good understanding of how one would estimate overall system availability under a given fault load. System availability studies such as [2, 33] are works in this direction. An interesting paper is that of Brown et al. [8], which outlines a methodology for benchmarking systems' availability. Our work here focuses more closely on cluster-based servers, and in particular the impact of faults affecting the communication system on service performability.

9 Conclusions

We have studied the impact of TCP and VIA on the availability and performance of cluster-based servers using a combination of fault-injection and analytic modeling. Surprisingly, the results show that, under our estimated fault load, a VIA-based server delivered better availability than a TCP-based server. We found the single most important factor to be how well the internal fault model of the communication substrate matches the actual faults. In a SAN context, the transient fault model used by TCP reduced overall performability, because of the lengthy time for TCP to report failures. VIA, with its fail-stop model, was much more appropriate for many faults because it allowed higher-level recovery to be initiated quickly. We also found that the use of a heartbeat protocol can mitigate the impact of TCP's inaccurate fault-model; however, the heartbeat protocol must be accompanied by a robust membership protocol since the heartbeat protocol can be led astray by lengthy communication delays. Finally, we found the pre-allocation of memory made VIA immune to resource allocation failures, whereas the implementation of TCP required dynamic memory resources.

We have also considered the case in which VIA-based systems may experience more stressful fault loads, because the underlying technology is less mature and the programming model more

complex. By varying the fault rates, we were able to quantify the cross-over point at which it may be advisable to use a more tested protocol such as TCP. We found that faults in a VIA-based server, such as switch, link, and application errors, would have to occur at approximately 4 times the rate of a TCP-based system for the performability of VIA and TCP systems to be equal.

We speculated how a communication layer which addressed availability and performance might be structured. We argue that it should use messaging (not a byte stream), single-copy transfers, pre-allocated channel resources, and match the network fabric's fault model.

Finally, in spite of the significant fault detection and recovery actions taken by both TCP and VIA, we expect overall system availability to remain between 99% and 99.8%, implying cluster-based systems will be unavailable for several days a year. We thus have a long way to go to make the availability of these systems utterly transparent to everyday users.

References

[1] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable Content-Aware Request Distribution in Cluster-Based Network Servers. In *Proceedings of USENIX'2000 Technical Conference*, San Diego, CA, June 2000.

[2] S. Asami. Reducing the cost of system administration of a disk storage system built from commodity components. Technical Report CSD-00-1100, University of California, Berkeley, June 2000.

[3] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving TCP/IP Performance Over Wireless Networks. *ACM MobiCom*, Nov. 1995.

[4] R. Bianchini and E. V. Carrera. Analytical and Experimental Evaluation of Cluster-Based WWW Servers. *World Wide Web Journal*, 3(4):215–229, December 2000.

[5] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet—A Gigabit-per-Second Local-Area Network. *IEEE Micro*, 15(1):29–38, Feb. 1995.

[6] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP vegas: New techniques for congestion detection and avoidance. In *SIGCOMM*, pages 24–35, 1994.

[7] E. Brewer. Lessons from Giant-Scale Services. *IEEE Internet Computing*, July/August 2001.

[8] A. Brown and D. A. Patterson. Towards Availability Benchmarks: A Case Study of Software RAID Systems. In *Proceedings of the 2000 USENIX Annual Technical Conference, San Diego, CA*, June 2000.

[9] E. V. Carrera and R. Bianchini. Evaluating Cluster-Based Network Servers. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing*, pages 63–70, Pittsburgh, PA, August 2000.

[10] E. V. Carrera and R. Bianchini. Efficiency vs. Portability in Cluster-Based Network Servers. In *Proceedings of the 8th Symposium on Principles and Practice of Parallel Programming*, Snowbird, UT, June 2001.

[11] E. V. Carrera, S.Rao, L.Iftode, and R. Bianchini. User-Level Communication in Cluster-Based Servers. In *Proceedings of the Proceedings of the 8th IEEE International Symposium on High-Performance Computer Architecture (HPCA 8)*, February 2002.

[12] D. R. Cheriton and C. L. Williamson. VMTP as the Transport Layer for HighPerformance Distributed Systems. *IEEE Communications Magazine*, 27(6):37–44, June 1989.

[13] R. Chillarege, S. Biyani, and J. Rosenthal. Measurement of Failure Rate in Widely Distributed Software. *25th Int. Symp. on Fault-Tolerant Computing (FTCS-25)*, pages 424–433, June 1995.

[14] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *Computer Communication Review*, 26(3):5–21, July 1996.

[15] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[16] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. S. Trivedi. A Methodology for Detection and Estimation of Software Aging. In *International Symposium on Software Reliability Engineering (ISSRE 1998)*, Nov. 1998.

[17] J. Gray. A Census of Tandem System Availability Between 1985 and 1990. *IEEE Transactions on Reliability*, 39(4):409–418, Oct. 1990.

[18] R. W. Hamming. *Coding and Information Theory, 2nd Ed.* Prentice-Hall, 1986.

[19] T. Heath, S. Kaur, R. P. Martin, and T. D. Nguyen. Quantifying the Impact ofArchitectural Scaling on Communication. In *Proceedings of the 7th Symposium on High Performance Computer Architecture (HPCA-7)*, Monterrey, MX, Jan. 2001.

[20] T. Heath, R. Martin, and T. D. Nguyen. Improving Cluster Availability Using Workstation Validation. In *to appear in Proceedings of the ACM SIGMETRICS 2002*, Marina Del Rey, CA, June 2002.

[21] J. C. Hoe. Improving the start-up behavior of a congestion control sheme for TCP. In *Proceedings of the ACM SIGCOMM '96 Conference on Communications Architectures and Protocols*, volume 26,4, pages 270–280, Stanford, CA, Aug. 1996.

[22] R. Horst. TNet: A Reliable System Area Nework. *IEEE Micro*, 15(1):37–45, Feb. 1995.

[23] Intel Architecture and Microsoft Corporation. Virtual Interface Architecture Specification, Dec. 1997. Available at http://www.viarch.org.

[24] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of the ACM SIGCOMM '88 Conference on Communications Architectures and Protocols*, pages 314–329, Stanford, CA, Aug. 1988.

[25] P. J. K. Jr., J. Sung, C. P. Dingman, D. P. Siewiorek, and T. Marz. Comparing operating systems using robustness benchmarks. In *Symposium on Reliable Distributed Systems*, pages 72–79, 1997.

[26] R. J. K. K. Ramakrishnan and D.-M. Chu. Congestion Avoidance in Computer Networks with A Connecitonless Network Layer: Part IV: A Selective Binary Feedback Scheme for General Topologies. Technical Report DEC-TR-510, Digital Equipment Corp., Aug. 1987.

[27] M. Kalyanakrishnam, Z. Kalbarczyk, and R. Iyer. Failure Data Analysis of a LAN of Windows NT Based Computers. In *Proceedings of the 18th Symposium on Reliable and Distributed Systems (SRDS '99)*, 1999.

[28] I. Lee and R. Iyer. Faults, symptoms, and software fault tolerance in the tandem guardian90 operating system. In *Int. Symp. on Fault-Tolerant Computing (FTCS-23)*, pages 20–29, 1993.

[29] X. Li, R. P. Martin, K. Nagaraja, T. D. Nguyen, and B. Zhang. Mendosus: A san-based fault-injection test-bed for the construction of highly available network services. In *In Proceedings of the 1st Workshop on Novel Uses of System Area Networks (SAN-1)*, Cambridge, MA, Jan. 2002.

[30] T. Liu, Z. Kalbarczyk, and R. Iyer. A software multilevel fault injection mechanism: Case study evaluating the virtual interface architecture. In *Symposium on Reliable Distributed Systems*, 1999.

[31] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland Publishing Company, Amsterdam, 1977.

[32] J. F. Meyer. Performability evaluation: Where it is and what lies ahead. In *Proceedings of the IEEE International Computer Performance and Dependability Symposium*, pages 334–343, Erlangen, Germany, Apr. 1995.

[33] B. Murphy and T. Gent. Measuring System and Software Reliability using an Automated Data Collection Process. *Quality and Reliability Engineering International*, pages 341–353, 1995.

[34] K. Nagaraja, X. Li, B. Zhang, R. Bianchini, R. P. Martin, and T. D. Nguyen. Using Fault Injection to Evaluate the Performability of Cluster-Based Services. Technical Report DCS-TR-491, Department of Computer Science, Rutgers University, In preparation.

[35] 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 *Proceedings of the 8th ACM Conference on Architectural Support for Programming Languages and Operating Systems*, pages 205–216, San Jose, CA, October 1998.

[36] C. Partridge. Implementing the Reliable Data Protocol (RDP). In *Proceedings of the 1987 USENIX Summer Conference*, pages 367–380, Phoenix, AZ, June 1987.

[37] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, Nov. 1984.

[38] C. Seitz. Myrinet—A Gigabit-per-Second Local-Area Network. Talk presented at Hot Interconnects II, Aug. 1994.

[39] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423,63–656, July and October 1948.

[40] W. Strayer, B. Dempsey, and A. Weaver. *XTP: The Xpress Transfer Protocol*. Addison-Wesley, 1992.

[41] M. Sullivan and R. Chillarege. Software defects and their impact on system availability - a study of field failures in operating systems. *21st Int. Symp. on Fault-Tolerant Computing (FTCS-21)*, pages 2–9, 1991.

[42] N. Talagala and D. Patterson. An Analysis of Error Behaviour in a Large Storage System. In *The 1999 Workshop on Fault Tolerance in Parallel and Distributed Systems*, 1999.

[43] K. S. Trivedi, K. Vaidyanathan, and K. Goseva-Popstojanova. Modeling and Analysis of Software Aging and Rejuvenation. In *Proceedings of the IEEE Annual Simulation Symposium, April 2000.*, Apr. 2000.

[44] T. M. Warschko, J. M. Blum, and W. F. Tichy. A reliable transmission protocol for myrinet. In *2nd Workshop on Cluster-Computing*, pages 135–144, Karlsruhe, Germany, March 1999.

[45] J. Wilkes. Hamlyn - An interface for sender-based communication. Technical Report HPL-OSR-92-13, Hewlett-Packard Research Laboratory, Nov. 1992.