VICTORIA UNIVERSITY OF WELLINGTON

Department of Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 471 5328
Fax: +64 4 495 5232
Internet: Tech.Reports@comp.vuw.ac.nz

DRUMS: A Distributed Performance Information Service

Andy Bond and John H. Hine

Technical Report CS-TR-91/1
October 1991

Abstract

DRUMS is a distributed system used to collect and distribute information about the performance of the individual systems making up the components of a modern network computing facility. DRUMS is designed to achieve a very high level of availability, to have minimum impact on the host systems, to adjust its processing capacity in response to changing demand, and to be easy to manage. Replication is used for both availability and performance. This paper discusses the design and structure of DRUMS and presents results based on our experience with its use.

Publishing Information

This report appeared in the *Proceedings of the 14th Australian Computer Science Conference* Sydney. February 1991.

Author Information

Andy Bond is currently a PhD student at Victoria University of Wellington. In 1987, he completed an Honours degree in Computer Science at Victoria and has since been researching task allocation and resource management in distributed systems. His research interests include distributed and multi-computer systems.

John Hine is Professor of Computer Science at Victoria University. His current research is centered on the measurement and management of resource use in distributed systems. He is also involved in the development of educational and research networks in the Asia and Pacific region.

1 Introduction

Modern operating systems are evolving toward a collection of workstations and servers connected by a network. The total amount of computing power available in such a system is enormous. Unfortunately our ability to effectively manage these systems has not kept pace with the technology. Important aspects of resource management include the allocation of tasks to hosts and the measurement of the performance of networked systems.

We are investigating several aspects of this problem. One of these is the high level scheduling of tasks to take advantage of lightly loaded workstations. As a foundation for much of our work, we need dynamic performance data from all the hosts on the network. We want to be able to answer questions such as

> From the set of hosts with at least 12 Mb of memory and a directly attached disk which currently has the lightest cpu utilisation?

We have built a distributed service, DRUMS[1], that maintains performance data for a network of over 30 Unix[2] workstations. DRUMS currently runs on both Hewlett-Packard and Sun workstations runing HP-UX, MORE/BSD and SunOS versions of Unix.

The service may be interrogated to determine a range of performance data. DRUMS is highly resilient to all but the most catastrophic of failures. The number of servers supporting the service is automatically adjusted to meet demand. The servers themselves move about the system in such a way as to minimize their impact on the performance of individual hosts.

This paper reports on the design and structure of DRUMS with an emphasis on the dynamic behaviour of the servers. We believe the general structure is applicable to a range of system services that do not require strict coherence among the databases of the individual servers.

The following section describes the goals of the service offered. Section three provides an overview of the structure and operation of DRUMS. Section four covers the design in detail. Following sections discuss the performance of the current implementation, and some other application areas including the need for further work. A more detailed description of the implementation of the various components of DRUMS may be found in [3].

2 Goals

In a typical system composed of a number of workstations it is possible to query an individual system to retrieve a variety of measures of its current and/or recent performance. To address the problem of task allocation among workstations, or to compare the performance of/load on a set of workstations we require the orthogonal view. Our requirement is generally a small set of measures across a number of workstations. For instance we might want to compare or display the CPU usage of a set of workstations. To obtain this we needed to build a measurement system that could hold performance information for all the hosts in our system.

We elected to build a service that could provide performance data for all workstations. We set the following goals for our service:

- *The service must be resilient to virtually all "partial interruptions" to the underlying computer systems and network.*

  The service will be an integral part of our higher level scheduling, thus it must be available at all times. Even a medium size distributed system has frequent partial

[1] DRUMS: Distributed Resource MeasUrement Service.

[2] Unix is a trademark of A.T.&T.

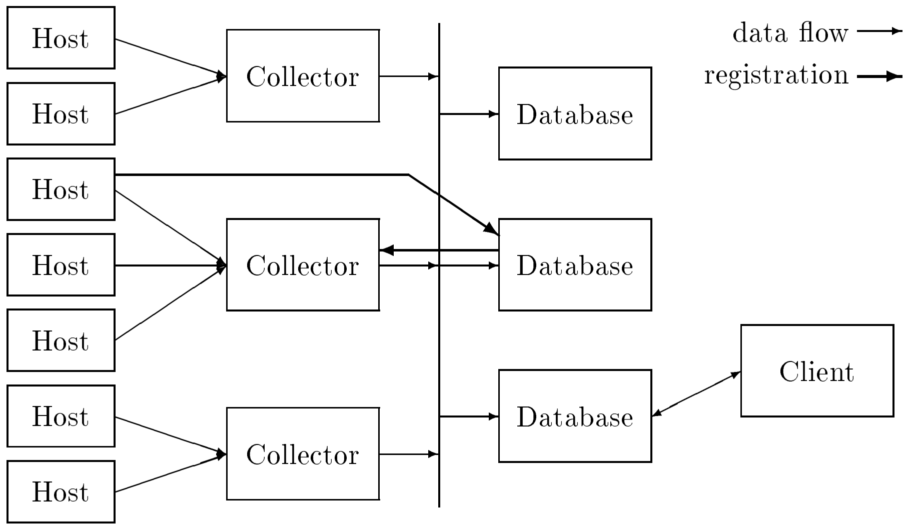

Figure 1: DRUMS consists of replicated data collectors and replicated databases.

outages as individual systems fail, system software is updated, etc. Note that we are much more interested in the availability of the system than in the correctness or coherence of the state of the individual servers. For example, it is quite acceptable for one server to not have the latest reading on the CPU utilisation of host "Otari".

- *The server must be responsive to requests for service.*

  The service is being designed to provide information in a real-time environment. It must be able to meet requests in a timely fashion. Ideally it should dynamically adjust its capacity to an increase or decrease in demand.

- *The service should have minimal impact on the other work of the host systems.*

  Part of our overall philosophy is make use of lightly loaded hosts in a distributed environment. The service should be designed to use "spare" capacity where it is available rather than running on an otherwise busy workstation.

- *DRUMS should require a minimal amount of management intervention.*

  The management of a modest size cluster of workstations is already too much work. We would like our service to simply "be there" requiring no intervention from the system administrator when new system software is installed, new hosts are added to the network, etc. DRUMS should just keep on working.

3 An Overview of DRUMS

Fundamental to any highly available service is replication. DRUMS is built from two replicated services: a data collection service and a database service. See figure 1. This design is similar in concept to the *troupes* presented by Cooper in [8]. We will adopt the concept of a troupe to represent a set of replicated servers.

In normal operation the Collectors periodically query a set of hosts to obtain performance information.[3] Each Collector maintains a list of "registered" hosts for which it is responsible. Each host is the responsibility of one Collector. Thus responsibility for the hosts is partitioned among the replicated Collectors. The partitioning is dynamic, responding to load or partial failures.

The Collectors then forward the data to all copies of the Database. Each Database maintains its own copy of the collective statistics. There is no attempt to maintain the coherence of the separate databases. The set of Database servers present a "performance information service" to clients.

A client may obtain information about the performance of any number of hosts with a single query. At the moment we have built a single client, `any`. The `any` program is a prototype scheduler for a distributed environment. It accepts as an argument a logical expression that specifies the attributes of a target system. It then interrogates DRUMS to find the current state of various hosts and returns a set of hosts that satisfy the original attributes.

If a host is temporarily out of service this is noted by the Collector responsible for it. The Collector continues to send messages about the host to each Database. These messages tell the Database that the host is "not available" rather than supplying statistics.

Each Database maintains a list of hosts and monitors the data it receives. If a Database notices that it is no longer receiving data about a particular host it assumes the Collector responsible for that host has failed. The Database attempts to register the host with another Collector. If it can not find a Collector prepared to accept the host it starts a new Collector and then registers the host with this Collector.

In a similar way the Collectors monitor the Databases. As each Collector sends out its data it counts the number of acknowledgements it receives. If this number drops below a minimum value the Collector starts a new Database. Complete details of how DRUMS recovers from partial failures of both hosts and network components are given in [3].

When a new host is added to the environment, DRUMS must be told that the new host is available. This is handled in a very natural way by the structure discussed above.

1. As part of the installation process the host is "registered" with a single Database by inserting some nominal statistics.

2. The Database notices these statistics are not being updated and registers the host with a Collector.

3. The Collector queries the host and sends the data to all current Databases.

The same technique is used for bootstrapping DRUMS. Each host in the networked environment is "registered" with a single, initial Database. The dynamics of DRUMS then creates all the other required servers.

A unique aspect of DRUMS is the dynamic nature of the servers, both Collectors and Databases. The servers are not bound to any particular host. Servers may migrate from one workstation to another or a new server may be created on a lightly loaded workstation to increase the performance of the overall service. While this leads to a number of advantages it also means care must be taken to ensure that a client can efficiently locate a server when service is required.

[3] We have replaced the `rstat` daemon with a similar facility that provides more information about system performance and provides a common interface across several varieties of Unix systems.

4 Design of DRUMS

This section deals with particular aspects of the design of DRUMS. Aspects that deal with the availability and mobility of DRUMS as a distributed service are emphasised.

4.1 Replication and Modularity

The design of highly available systems centers on the replication of components and the requirements for agreement about state or data among the components. Considerable work has been done in the area of file and database coherence [1, 11, 12, 13] and the design of fault tolerant systems [2]. The data held in the DRUMS database is built from real-time data collected by the Collectors. There is no requirement for a high degree of coherence between the data held by the individual servers. For this reason we have opted for a design that would be less expensive than one demanding strict coherence.

The performance of a service can also be increased by replication. In the case of DRUMS there are two distinct loads to be met: the collection of data from hosts and the responses to queries by clients. Our design has seperated these two functions into seperate modules, the Collectors and the Databases. This design will allow the number of either server to change without effecting the level of service offered by the other.

The Collectors are responsible for a number of hosts. This is more effective than running a single process on each host, but does require that the number of servers increase as new hosts are added to the system. With the modular design the Collectors can expand to handle the number of hosts and the Databases can expand independently in response to the level of queries.

Although both DRUMS' servers are replicated neither are written with any knowledge of their own replication. This design yields several advantages:

1. The servers do not depend on knowing the number or location of other servers in their troupe. There is no need for an algorithm to maintain this information in a consistent manner.

2. Communication of control information between members of a troupe is eliminated. The communication of data between troupes is used to allow one troupe to monitor the status of the other. Additional communication is only required when a problem is detected.

3. The coding of each server is greatly simplified. It is written as if the server were a standalone program.

4. Since a DRUMS' server is not tied to any particular workstation the service is not dependent on the operation of any particular server.

4.2 Communication

While one Collector is responsible for a number of workstations each Database requires data about every workstation. To accomplish this each Collector must send the data it has collected to all the Databases using the broadcast feature of Sun's RPC facility. Each Database acknowledges receipt of the message.

A client that wishes to use DRUMS follows a similar approach. Since the client does not know how many copies of the server are running, which hosts they are located on, or what the load on particular servers is, it must use the same RPC facility specifying a well known port. The RPC facility will deliver the message to all of DRUMS' Database servers and all will eventually reply, though the client is only interested in the first response.

Each Collector caches the address of one Database server to use when it wishes to migrate as described below. Clients of DRUMS could also cache the address of the Database that replies to their first query to avoid the subsequent use of the broadcast protocol.

The reader will note that the communication requirements of DRUMS are ideally suited to some form of group or multicast communication. For instance the semantics of the V-Kernel[6] group communication are ideally suited for both the transmission of data from the collectors to the database and for querying the database by clients. The VMTP protocol[4, 9] has been designed as a protocol to support distributed systems. Support for multicast communication is one of its principle features.

There are also forms of remote procedure call designed for reliable, replicated systems. Cooper used Circus Replicated RPC [7] to implement troupes in [8]. The RPC implementations generally have stricter semantics, for instance "exactly once" versus the "at most once" semantics of group multicast communication. The latter is adequate for our requirements.

4.3 Server Management

There are two aspects of server management in DRUMS. The first is the migration of a server from one system to another. This is handled by the servers themselves. The second is the creation or deletion of a server. Since the servers are not aware of their replication this must be handled by an outside agent. This is solved within DRUMS by allowing the two subsystems to manage each other, the Collectors manage the Databases and vice versa.

This approach has a number of advantages over the traditional design for distributed services in which the individual servers cooperate to maintain a service. Several advantages of our approach are listed above. They should become evident in the following discussion.

4.3.1 Server Migration

A broader goal of our research is to develop algorithms for making more effective use of resources in a system composed of networked workstations. DRUMS has been designed to be consistent with this policy. The DRUMS' servers seek lightly loaded hosts to run on. Both the Collectors and the Databases continuously monitor the load on their host system. When a "better" system becomes available, they will migrate.

Both Collectors and Databases use the same mechanism for determining whether or not they should migrate to another host. Periodically the server invokes the procedure `find_host_by_application`. This procedure is the heart of the `any` program mentioned earlier. The procedure is invoked with parameters that cause it to return a list of hosts that are the "best" candidates for hosting the server. This list is currently the top 20% of available hosts.

The server will initiate the migration procedure if

1. its current host is not in the list of candidates, and
2. there exists a member of the candidate list that is not currently hosting a similar server.

The actual migration procedures used by the two types of servers differ sufficiently that we will discuss each separately.

Collectors The state held by a Collector consists of the set of registered hosts that it is responsible for. Rather than migrate with this state the Collector follows a procedure that may allow it to terminate, reducing the number of Collectors.

The Collector attempts to re-register each host it is responsible for with a different Collector. If it succeeds it is no longer needed and simply terminates. If it does not succeed, i.e. the remaining Collectors will not accept a registration, a new Collector is started on the "best" host from the list obtained at the start of the procedure. The remaining hosts are then re-registered with the new Collector. Experience over the past several months indicates that a new Collector is required about 50% of the time.

Databases The state information held by a Database may either be retrieved from another Database or (eventually) reconstructed from updates received from the Collectors. Thus a Database may simply "forget" all it's information before terminating. Once a Database determines that it should migrate to a new host it starts a copy of the Database on that host and terminates. The new Database queries the DRUMS service, i.e. the other Database servers, to get an initial copy of the database. A Database server always migrates by starting a copy of itself rather than simply dying as the Collectors due. Management of the number of Database servers is discussed in the following section.

4.3.2 Controlling the Number of Servers

In the current implementation of DRUMS the number of Collectors or Databases is automatically increased whenever the number present are not able to cope with the offered load. The initiation of a new server is done by the client of the service rather than by agreement among the servers offering the service.

Collectors The load placed on a Collector is determined by three parameters:

U is the target cycle time between updates of the statistics held in the Databases. The Collector will try to refresh each set of statistics every U seconds.

T is the timeout interval for the broadcast of host data to the Database servers. The Collector may need to wait for a period of up to T seconds to count the number of Databases acknowledging receipt.

W is the time in seconds between successive broadcasts of host data to individual Databases.

Each Collector uses subprocesses to perform the broadcast of data to the Databases. This allows the Collector to overlap retrieval of data from another host with counting the acknowledgements from the Databases. We set $W > T$ to limit the number of concurrent subprocesses involved in the transmission of data.

These parameters are set by the system administrator. A collector can then determine the maximum number of hosts it is able to handle from

$$N = \lfloor \frac{U}{W} \rfloor$$

A Collector will refuse a request to register a host whenever its list is full. A new Collector is started whenever all existing Collectors refuse to register a host. The original message will have been sent by a Database or a Collector that is terminating. Note that in the latter case this "terminal" Collector is no longer a part of the data collection service, but a client of the service.

Databases A new Database is started by a Collector if that Collector cannot find a minimum number of Databases to accept some statistical information it is trying to distribute. This may be caused by the Database servers being either busy or down and is detected by the lack of sufficient acknowledgements in period T. The underlying assumption is that the Databases are too busy servicing clients to keep their data up to date.

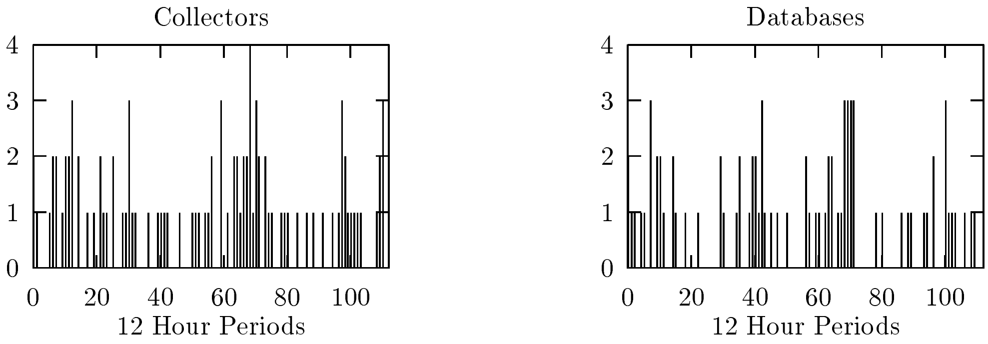

Figure 2: The number of abnormal deaths of servers in each 12 hour interval over a period of 8 weeks.

Removing a Database server is a much more difficult problem. The current design is such that each server is not aware of its own replication. This greatly simplifies the design of the server in ways mentioned above. Similarly, the alternate servers responsible for increasing the number of servers make that decision without being aware of the degree of replication of the server. They simply perceive that service is not good enough and create a new server.

In deleting a server it is necessary to ask questions such as:

1. If a server is deleted will a sufficient number be left to offer adequate performance?
2. If a server is deleted will a sufficient number be left to survive a "partial failure" of the system?

These questions cannot be answered without a single entity having an accurate global view of the state of the system and the various servers. For instance, each of the above questions needs to be considered within the context of a partitioned network. Thus we have a conflict in principles. We cannot remove a server without an accurate global picture; however, we have otherwise avoided the need for a completely correct picture of the system state.

The current implementation of DRUMS does not remove Database servers. Our experience suggests this is not a significant problem. The number of Databases does not grow unreasonably and, as the following section will show, the occasional loss of a server due to a reboot of a host is sufficient to remove excessive servers.

5 Results

DRUMS has been in every day use for the past year. This section reports on our experiences with DRUMS.

5.1 Availability

As we write this, the most recent release of DRUMS has been continuously available for eight weeks. During this time individual workstations and file servers have been rebooted at random times, on one night new system software was installed on three quarters of the system and another night a new automount facility was installed.

Figure 2 indicates the number of unexpected terminations of DRUMS' servers in each 12 hour interval over the eight week period. From the data we can determine that the MTBF[4] of the Collectors is 14 hours and the Databases is 17.2 hours. This in itself gives an indication of the environment distributed applications such as DRUMS run in.

[4]Mean time between failures.

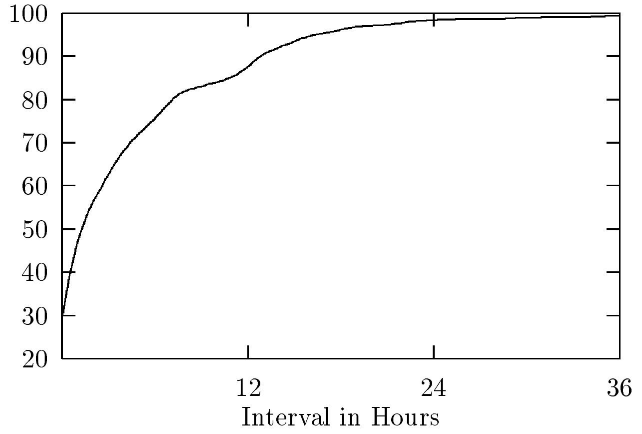

Figure 3: Frequency distribution of server residency times for both Collectors and Databases.

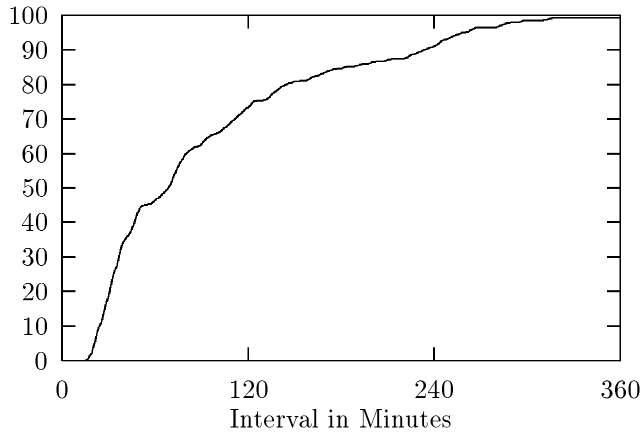

Figure 4: Frequency distribution of server residency times for both Collectors and Databases limited to 9am through 5pm.

	All Servers	Collectors	Databases
24 hours	19.8	12.28	7.52
9 to 5 Only	17.8	11.85	5.95

Figure 5: Server movement rates in movements/day.

5.2 Migration

The migration of servers to lightly loaded hosts is a unique feature of DRUMS. Figures 3 and 4 show the cumulative distribution functions for the time spent on a single host. The data was collected over an eight week period. Figure 3 includes the entire sample period. Occasionally a server will reside on one host for several days but, most often, it is only for a period of hours.

Figure 4 looks at just the eight hour periods from 9am to 5pm when our system is most heavily loaded. Since academic staff are frequently away from their offices for a period of one or more hours it is not surprising to find a reasonable number of residency times of this length. The median value during this period is over an hour. There is still a bias toward shorter intervals.

From these figures it is hard to get a feeling for how often servers move. The table in figure 5 gives this data in total movements per day for all servers and movements per day for individual servers. During a working day there is an average of 17.8 migrations. This is not sufficient to place any load on the system.

6 Extension to Other Areas

The design that we have employed in DRUMS should be portable to other applications. One obvious area is other real-time systems that collect data. For instance a system that collected data from a set of sensors and needed to make that data available to displays, feedback control systems, alarms, etc. would be an obvious candidate.

Another potential area is the class of servers whose databases are built up through a caching mechanism. Consider for instance a global name server such as that proposed by Cheriton and Mann in [5]. There are requirements for replicated servers for both availability and performance, and for additional servers as the load increases. There is no requirement for strict coherence among the caches. One problem that is just beginning to be addressed is the location of services in a large internetwork[10]. A service implemented by migrating servers would clearly require an effective mechanism for accessing the service.

7 Conclusion

This paper has presented the design of DRUMS, a highly available performance information service. DRUMS is composed of two sets of replicated servers. The DRUMS servers migrate from host to host attempting to locate a lightly loaded host to reside on. Further the servers increase in number as the offered load increases. The environment in which DRUMS currently runs has resulted in a MTBF for individual servers of 14 to 17 hours. Despite this DRUMS has been continuously available. The design of DRUMS should be applicable to other data collection applications and services with minimal coherency requirements.

References

[1] J. Akoka. Design of optimal distributed database systems. In *Proc. 1st Int. Symp. on Distributed Data Base*, pages 229–245, Paris, 1980.

[2] Ken Birman and Thomas Joseph. Low cost management of replicated data in fault-tolerant distributed systems. *ACM Transactions on Computer Systems*, 4(1):54–70, February 1986.

[3] Andy Bond and John H. Hine. Drums: A distributed statistical server for stars. In *Winter USENIX 1991*, Dallas, January 1991. UNIFORUM.

[4] David R. Cheriton. VMTP: A transport protocol for the next generation of communication systems. In *Communication Architecture and Protocols, SIGCOMM 86 Symposium*, Stowe, Vt., August 1986. ACM.

[5] David R. Cheriton and Timothy P. Mann. Decentralizing a global naming service for improved performance and fault tolerance. *ACM Transactions on Computer Systems*, 7(2):147–183, May 1989.

[6] D.R. Cheriton and W. Zwaenepoel. Distributed process groups in the v kernal. *ACM Transactions on Computer Systems*, 3(2):77–107, 1985.

[7] Eric C. Cooper. Circus: A replicated procedure call facility. In *Proc. of 4th Symp. on Reliability in Distributed Software and Database Systems*, pages 11–24, October 1984.

[8] Eric C. Cooper. Replicated distributed programs. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, pages 63–78. ACM, December 1985.

[9] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[10] Richard Golding and Darrel D.E. Long. Accessing replicated data in an internetwork. Technical Report ucsc-crl-90-34, Univ. of California, Santa Cruz, Computer and Information Sciences, August 1990.

[11] S. Jajodia. Managing replicated files in partitioned distributed database systems. In *Proc. 3rd Int. Conf. on Data Engineering*, pages 412–418, Los Angeles, Ca., February 1987.

[12] Sape Mullender and Andrew S. Tanenbaum. A distributed file service based on optimistic concurrency control. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, pages 51–62. ACM, December 1985.

[13] R. van Renesse, A. S. Tanenbaum, and A. Wilschut. The design of a high-performance file server. In *Proceedings of the 9th Int. Conf. on Distributed Computing Systems*, pages 22–27, Newport Beach. CA USA, June 1989. IEEE.