A Replicated Monitoring Tool

Darrell D. E. Long[†]
Computer & Information Sciences
University of California, Santa Cruz

Abstract

Modeling the reliability of distributed systems requires a good understanding of the reliability of the components. Careful modeling allows highly fault-tolerant distributed applications to be constructed at the least cost. Realistic estimates can be found by measuring the performance of actual systems. An enormous amount of information about system performance can be acquired with no special privileges via the Internet.

A distributed monitoring tool called a tattler *is described. The system is composed of a group of tattler processes that monitor a set of selected hosts. The tattlers cooperate to provide a fault-tolerant distributed data base of information about the hosts they monitor. They use weak-consistency replication techniques to ensure their own fault-tolerance and the eventual consistency of the data base that they maintain.*

1 Introduction

Distributed systems are now pervasive. Few system architects would consider designing a system that could not interact with other systems. Soon it will be rare to find computers that are not connected by a network. With distribution comes an increased incidence of partial failure. Replication of both control and data can be employed to provide systems capable of tolerating partial failures.

To use replication techniques most effectively, it is important to understand the nature of the failures to be masked. Recent studies include analyses of Tandem systems [1, 2] and the IBM/XA system [3]. Research covering heterogeneous systems is less common. Very few such studies have appeared in the open literature, although it is certain that most companies perform reliability studies of their products internally.

An earlier study [4] that used the Internet to estimate several parameters, including mean-time-to-failure (MTTF) and availability. These estimates were then used to derive an estimate of mean-time-to-repair (MTTR). While this study provided many important results, it suffered from several weaknesses. First, the assumptions made about the distributions that described host failures may not reflect those found in actual systems. Second, the network that connected the polling host (pollster) to the polled hosts (respondents) can affect the statistics by reporting false failures. As a result, the estimates of the parameters may differ significantly from the actual values. In particular, the derived estimate of mean-time-to-repair can be much larger than expected since small errors in the availability estimate that are introduced by the intervening network can have a significant effect.

Estimates of mean-time-to-failure were based on reported up-times and not the actual time of failure. This was the best information available since a host is not in a position to give its failure time as it goes down. As a result, there was a bias towards more reliable hosts which means that the estimate of MTTF is larger than the true value.

Similarly, availability was estimated by the fraction of hosts that were reachable by the pollster. To ensure that only hosts capable of answering were queried, an initial poll was made and only hosts that answered this poll were counted in the availability census. Unfortunately, there are a significant number of network segments separating the pollster from most respondents and so a network failure may be misinterpreted as a host failure.

The most direct way of determining statistics such as MTTF and availability is through direct measurement. A fault-tolerant monitor is being developed that can be placed at strategic locations around the Internet. Instances of the monitor will be placed to minimize the amount of shared network so that a failure of a router or a link will be unlikely to disable more than one monitor. They replicate their statistics so that even the permanent failure of one monitor will not cause a significant loss of information.

These monitors are called *tattlers* since they periodically inquire about other hosts and then tattle to each other about what they learn. The distributed data base is managed using an epidemic replication protocol [5, 6, 7]. Such protocols provide *weak* consistency guarantees, which are sufficient

[†]Supported in part by a grant from Sun Microsystems, Incorporated, and the University of California MICRO program.

for statistical purposes. Given the frequency of polls and the potentially large number of tattlers, a pessimistic replication protocol is impractical.

2 Tattlers

For the sake of concurrency and modularity each tattler is composed of several parts: a *client interface*, a *polling daemon*, a *data base daemon*, and a *tattler daemon*.

The *polling daemon* produces sample observations. It takes samples at a specified rate, and can be requested to start or stop sampling using the *client interface*. The *data base daemon* provides stable storage for sample observations (from the polling daemon), and meta-data from the client interface and the tattler daemon. The *tattler daemon* is responsible for group membership (adding and deleting tattlers) and managing the consistency of the replicated data base through anti-entropy sessions. The structure of a tattler is depicted in figure 1.

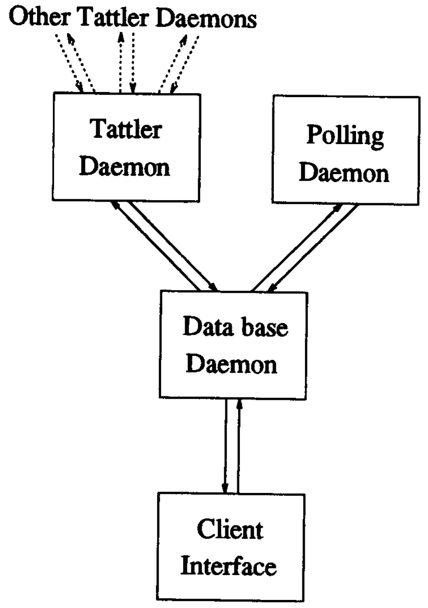

Figure 1: Structure of a Tattler.

There are several advantages to replicating the tattlers around the network. First, it provides a fault-tolerant method for monitoring hosts. All but one of the tattlers can fail and the set of hosts can still be monitored (albeit in a degraded mode). It also provides a way of mitigating the effect of transient network failures. When monitoring hosts from a single point, the failure of one router can prevent any host from being polled. When several relatively independent polling daemons work together, it will be very unlikely that a total failure can occur. Second, instead of estimating parameters such as MTTF based on a large sample with an unknown distribution, the tattlers will be able to record the actual events (with an epsilon error). Since the quantities are being directly measured, questions such as the governing distribution are less important. Third, because the tattlers are distributed they can perform many more queries than a single polling program. While a single polling program would create roughly the same message traffic, it would be vulnerable to failure. It would also take significantly longer to complete its task since there would be no parallelism and it would have to poll for a longer period to make up for the data lost due to failures.

Collectively the tattlers are responsible for maintaining a list of hosts to monitor, and collecting statistics on them. A record of the form $\langle$*host, poll method, poll interval*$\rangle$ is kept for each host. The client interface allows hosts to be added or deleted from this list. The recorded statistics are stored in logs. These logs can take any form, but are initially sequences of tuples of the form $\langle$*host, boot time, sample time*$\rangle$.

The tattler daemon communicates this information to other tattlers, using weak-consistency group communication protocols. These protocols ensure that the logs and host list are eventually consistent [8] (see §3 for details). To accomplish this, each tattler daemon periodically contacts another tattler and the two exchange their log and host files in anti-entropy sessions. Both then merge the information to obtain a log with better coverage of the monitored hosts.

The group communication protocols require the tattlers to know the identity of the other tattlers. This is done using a weak-consistency membership protocol [9]. This protocol requires that a new tattler join the group of tattlers by obtaining one or more sponsors. A tattler can leave the group by following a two-phase protocol: it first declares its intent to leave. It can destroy state information once it knows that all other tattlers have observed its declaration of intent. The client interface provides a mechanism to request tattlers to shut down.

The tattler's state – logs, host list and group information – are written to secondary storage by the data base daemon to simulate a fault-tolerant process. When a host on which the tattler runs fails and recovers, the tattler returns with a slight case of amnesia. It will remember statistics about events that were written to secondary store, but it will not know about more recent events that occurred while it was down. The group communication protocols ensure that it will eventually receive any missing information.

At the heart of the tattler are several event queues. All the events the tattler must perform in the future are recorded in these queues, including initiating anti-entropy sessions and polling hosts to obtain measurements. By breaking the tattlers up into several processes, several activities can take place concurrently. Since the time when an event occurs can be important, serializing the events in a single program could compromise the statistical data.

The polling daemon periodically polls hosts using the polling method given in the list of hosts. This method is not coded into the polling daemon; instead, it is provided as separate programs to be executed by the polling daemon. This also allows the tattler to go on with its business instead of waiting for the poll to complete. Polling can be done at any selected interval, though in the initial configuration it polls at exponentially-distributed random intervals (with a given mean) both for statistical purposes and to prevent synchronous behavior where multiple tattlers poll the same host at once.

The polling method program is determined from the poll method field in the host list. The system will provide methods that use such protocols as Sun RPC and ICMP echo. By separating the polling method from the tattler daemon, new methods can be added with relative ease.

Each polling method will require a corresponding merge method, to be used when other logs are merged with the local log. This is necessary because each polling method may record different information with its own unique semantics. The tattler daemon is unconcerned with the semantics of the data that it gathers.

A client interface is provided to manipulate the tattlers. It allows hosts to be added and deleted from the monitoring list, and allows a user to suspend monitoring of certain hosts. It can inform a tattler that it should shut down and leave the process group. Similarly, new tattlers can be added easily. It is sufficient to contact a single tattler to perform all of these operations. In fact, the client interface need only contact the closest tattler, and the group communication protocols ensure that the operation will eventually be known by all tattlers.

3 Weak Consistency Replication

The tattler uses new weak-consistency replication and group membership protocols developed by members of our research group. The tattler, like many distributed applications can be written as a group of processes that communicate through a *group communication protocol*. This protocol ensures that the group member processes have a *consistent* view of the service they are to provide, by stipulating the way that messages are sent between processes. The group communication protocol generally provides a *multicast* service that sends a message from one process to all other processes in the group.

In general, two processes are consistent at time t if they have received the same set of messages. Various degrees of consistency place different constraints on the orders in which the messages can be delivered. In general, the stronger the consistency requirements, the more expensive the protocol. Protocols that provide synchronous communication can require long latency for the multicast operation. Those that provide strong guarantees on message ordering append ordering information to each message or can impose latency requirements.

There are a number of weak-consistency protocols that provide *eventual consistency*: they ensure that every message will eventually be delivered to every process, but they provide only weak bounds on the time required. Various message orders are possible, ranging from unordered delivery to various total orderings. These protocols allow changes to the data base to be processed at any host, then later forwarded to other hosts. This provides a highly available service, and communication can occur at off-peak times; it also handles failures well. On the other hand, the application must be able to tolerate temporarily inconsistent information.

The *time-stamped anti-entropy* method is the heart of these protocols. As in the Grapevine anti-entropy protocol [5], messages are exchanged by two processes. The exchange of information continues until they are mutually consistent. As long as processes in a group continue to perform these exchanges, changes will eventually propagate to all replicas. Unlike the Grapevine protocol, this method maintains lists of time-stamps at each process. These lists are also exchanged, and allow the processes to identify both what information the other is missing and what messages other processes have received. The ability to indentify missing information increases the efficiency and allows applications to build stronger forms of consistency.

Our group has also developed a new light-weight group membership mechanism that allows temporary inconsistencies in membership views [10]. Each group member maintains a *view* of the group, listing those processes it believes to be members. The members use weak-consistency communication protocols to ensure that all group members eventually converge to a consistent view of the membership, as discussed in the last section. The mechanism is resilient to $k \leq n - 2$ members failing by crashing, where n is the number of members currently in the group.

4 Status

At the time of writing, the tattler design has been completed and the coding of the first prototype is nearing completion. The tattler is expected to be fully operational by September 1, 1992. The first version uses the Sun RPC protocol to gather information. This allows information on a wide variety of host types to be gathered, while remaining a manageable programming task.

Once the tattler becomes operational, it will be used to study weak-consistency replication protocols, in particular circularity, in addition to its primary task of host monitoring.

As experience is gained with the tattler, other protocols will be added. For example, as the *record route* option could be used to map routes from tattlers to the hosts being monitored. Other commonly available network services are being investigated as sources of information.

Acknowledgments

The weak-consistency protocols used by the tattler were developed by R. Golding as part of his dissertation research. K. B. Sriram and J. Wright are contributing to the development of the tattler. J. Carroll, K. Taylor and M. Long contributed through their thoughtful comments.

References

[1] J. Gray, "Why do computers stop and what can be done about it?," Tech. Rep. 85.7, Tandem Computers, June 1985.

[2] J. Gray, "A census of Tandem system availability between 1985 and 1990," Tech. Rep. 90.1, Tandem Computers, Jan. 1990.

[3] S. Mourad and D. Andrews, "The reliability of the IBM/XA operating system," in *Proceedings 15th Annual International Symposium on Fault-tolerant Computing*, IEEE, June 1985.

[4] D. D. E. Long, J. L. Carroll, and C. J. Park, "A study of the reliability of internet sites," in *Proceedings of the 10th IEEE Symposium on Reliable Distributed Systems*, (Pisa, Italy), IEEE, Sept. 1991.

[5] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," *Operating Systems Review*, vol. 22, pp. 8–32, Jan. 1988.

[6] A. D. Birrell, R. Levin, R. M. Needham, and M. D. Schroeder, "Grapevine: an exercise in distributed computing," *Communications of the ACM*, vol. 25, pp. 260–74, Apr. 1982.

[7] M. D. Schroeder, A. D. Birrell, and R. M. Needham, "Experience with grapevine: the growth of a distributed system," *ACM Transactions on Computer Systems*, vol. 2, pp. 3–23, Feb. 1984.

[8] R. A. Golding, "Distributed epidemic algorithms for replicated tuple spaces," Tech. Rep. Technical report HPL–CSP–91–15, Concurrent Systems Project, Hewlett-Packard Laboratories, June 1991.

[9] R. A. Golding, *Weak-consistency group communication and membership*. Ph.D. dissertation, University of California, Santa Cruz, Sept. 1992 (expected).

[10] R. A. Golding and K. Taylor, "Group membership in the epidemic style," Tech. Rep. UCSC–CRL–92–13, Computer and Information Sciences, University of California, Santa Cruz, Apr. 1992.