Challenges in Long-Term System Logging

Ian F. Adams
University of California, Santa Cruz

Ethan L. Miller
University of California, Santa Cruz

1 Introduction

A wide variety of techniques exist for tracing both general and storage specific system behavior. These range from periodic crawls of file system metadata [2, 5] and system call interception [3] to network port mirroring [6] and complicated end-to-end tracing methodologies in distributed systems [4, 9]. While these works all have good techniques for *producing* data on system activities, by themselves they do not address issues in *maintenance* of the logged data, particularly in long-term logging and tracing scenarios. They have no provisions for dealing with the potentially massive volume of data that will accumulate over long periods, nor do they have any reliability mechanisms for safely storing log data over many years. Put another way, with large volumes of logged data we need the ability to intelligently *forget* less useful data to control data volume and usability and reliably store what we *want* to keep for years on end.

Yet, why should one even care about logging and tracing of system behaviors over the long-term? The short answer is that there are system behaviors that may take months or even years to become clear. For example, in our own archival workload studies [1], years of data were often necessary for behaviors to be clear. In one instance, Google did a slow scan of available materials over the course of a month, eventually accounting for over 70% of retrievals for the *entire 3 year trace*. Had we had shorter a dataset we would have missed events like the Google crawl that radically changed our conclusions. Another area where this can be of benefit is within computer security. For example, a port scan is usually fairly evident if it is done quickly, but if it is done over the course of weeks or months it can be easy to miss without storage and subsequent analysis of logged data.

In the rest of this paper we discuss the primary challenges we have identified for long-term logging, and our proposed approaches to addressing them.

2 Challenges and Approaches

Physical Scalability In general we are moving away from monolithic systems, to distributed systems comprised of many discrete physical and logical entities, *e.g.* the cloud. Correspondingly, a logging framework should be able to smoothly scale from a few nodes up to many thousands without saturating compute and network resources. It should be decentralized, as single points of failure make scalability more difficult, as well as hinder robustness.

Our approach to this challenge is to build a peer-to-peer system using a distributed hash table (DHT) implementation such as Chord [7]. DHTs provide a robust communication substrate that can smoothly scale from a few nodes to many thousands with low per-node and network overhead. Additionally we plan on subdividing the logging system into a hierarchy of groups such that any individual group of nodes has detailed knowledge of nodes within its group and coarser knowledge of neighboring groups. This allows for global location of resources and hierarchical system summarization without the need for any individual node to have detailed global knowledge of the entire system.

Temporal Scalability A challenge unique to long-term tracing is the storage and maintenance of activity logs over the course of many years. The volume of data that is likely to be logged over many years may be very large, and we must also ensure the safe storage of the logs themselves. Put another way, we must provide a way for the system to intelligently and selectively forget logged data that may be of limited use after as it ages, while reliably storing the desired data for years on end.

Large volumes of log data can be a problem for 2 primary reasons. First, depending on the granularity of data being logged, logs can take up enormous amounts of space. Imagine keeping every disk IO a system saw for 10 years. Second, what is considered a useful granularity of data may change over time. Using the prior example of a long-term IO trace, one may not care what every IO was 10 years prior, but may still want to know broad information about the system, such as read and write ratios.

Our approach to dealing with large volumes of data over time is to periodically *transform* logged data into coarser granularity formats via user-defined policies. Using our above example of a block-level IO trace, after a logged activity passes a certain age, it may be acceptable to transform or truncate portions of the logged activities data into a higher level form, such as storing hourly summary or per-file access information. Using transformation along with data compression techniques will allow for large, long-term traces to be efficiently gathered.

Maintaining the safety of the logs is just as important as keeping them at a usable size and granularity. In a short term observation of a few days or week, the loss of a logged data can be painful but not necessarily catastrophic. Often times it is possible to simply re-run an experiment or restart one's observations. When there are years of logs involved, logs may be effectively irreplaceable.

To address this, we will be incorporating automated replica management of logs into our proposed framework. This ensures not only the safe storage of logged data over time, it provides the opportunity to explicitly note node absence in a distributed system, which we describe next.

Noting Absence A common problem we have run into is trying to understand the cause of sudden reductions in activity in a log. For example, in activity logs we obtained from National Center for Atmospheric Research (NCAR) tertiary storage system in 2011 we see some days that have an order of magnitude *less* activity than others. It is unclear if this is the result of a crash or maintenance cycle, or actually is just a sudden drop in activity. Because of this, we see a need for a long-term tracing system to be able to explicitly note the absence of processes or entire physical devices.

Our approach for tackling this problem is to leverage replica management of logs. Each node that is logging activity will have one or more replicas of its logs. The nodes maintaining those replicas can explicitly note *within* the replicas they are managing when another node fails or is otherwise non-responsive.

Implementation As our design matures, we plan on implementing and testing our solutions in a framework we are calling *Janus*. Janus will provide an interface for applications and users to send events they wish to be logged, and allow them to provide specifications for how to transform and trim logged data over time. Janus will also provide for replica management of logged data, ensuring the safe storage of logged data as it is produced with minimal human interaction.

References

[1] ADAMS, I. F. *et al.* Analysis of workload behavior in scientific and historical long-term data repositories. Tech. Rep. UCSC-SSRC-11-01, University of California, Santa Cruz, Mar. 2011.

[2] AGRAWAL, N. *et al.* A five-year study of file-system metadata. In *Proc. of FAST'08*

[3] ARANYA, A. *et al.* Tracefs: A file system to trace them all. In *Proc. of FAST'04*

[4] BARHAM *et al.* Using Magpie for request extraction and workload modelling. In *Proc. of OSDI '04*

[5] GIBSON, T. J., AND MILLER, E. L. Long-term file activity patterns in a UNIX workstation environment. In *Proc. of MSST'98*

[6] LEUNG, A. W *et al.* Measurement and analysis of large-scale network file system workloads. In *Proc. USENIX ATC'08*

[7] STOICA, I *et al.* Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. of SIGCOMM '01*

[8] STORER, M. W. *et al.* Logan: Automatic management for evolvable, large-scale, archival storage. In *Proc. PDSW '08*

[9] THERESKA, E. *et al.* Stardust: Tracking activity in a distributed storage system. In *Proc. SIGMETRICS '06*