

Co-designing the Failure Analysis and Monitoring of Large-Scale Systems

Abhishek Chandra, Rohini Prinja, Sourabh Jain, Zhi-Li Zhang *
{chandra,rohini,prinja,sourabh,zhizhang}@cs.umn.edu
Department of Computer Science, University of Minnesota - Twin Cities

ABSTRACT

Large-scale distributed systems provide the backbone for numerous distributed applications and online services. These systems span over a multitude of computing nodes located at different geographical locations connected together via wide-area networks and overlays. A major concern with such systems is their susceptibility to failures leading to downtime of services and hence high monetary/business costs. In this paper, we argue that to understand failures in such a system, we need to co-design monitoring system with the failure analysis system. Unlike existing monitoring systems which are not designed specifically for failure analysis, we advocate a new way to design a monitoring system with the goal of uncovering causes of failures. Similarly the failure analysis techniques themselves need to go beyond simple statistical analysis of failure events in isolation to serve as an effective tool. Towards this end, we provide a discussion of some guiding principles for the co-design of monitoring and failure analysis systems for planetary scale systems.

1. INTRODUCTION

Large-scale distributed systems such as content distribution networks [1], peer-to-peer systems [3, 4], computation Grids [7, 6, 11, 12], and network testbeds [8] provide an essential platform for numerous distributed applications ranging from content sharing and Web services to VoIP and scientific simulations. Many of these systems consist of large number of nodes communicating with each other over widely distributed networks. Due to their inherent scale, diversity and complexity, these systems are prone to frequent failures, which could be caused by a variety of factors: network instabilities, power outages, node crashes, application software failures or security attacks. Any downtime due to failures, whatever the cause, can lead to large disruptions and huge

*The work is supported in part by the National Science Foundation grants CNS-0435444, CNS-0626812 and CRI 0709048, an NSF CAREER Award CNS-0643505, as well as an IBM Faculty Partnership Award and a University of Minnesota DTC DTI grant.

losses; a recent example of such a failure is the one affecting Skype users in August 2007 in which 200M users could not use the Skype service for nearly 48 hours [5].

To provide high level of reliability and availability in such large-scale systems, it is critical to detect, diagnose and fix these failures as they happen. Identifying the location and cause of a failure is especially important for such troubleshooting. Accurate diagnosis can not only lead to quick repair and recovery of failures, but also provides insights for making longer-term decisions on resource provisioning, software debugging, and hardware upgrades. For instance, the ability to determine that a failure occurred at an end-host rather than in the network can help system administrators focus their troubleshooting effort on that end-host. Moreover, they can come up with the correct remedy if they can determine whether the failure was due to a hardware crash or a software bug. However, identifying the actual cause of failures in large scale systems is a challenging task due to several reasons. First of all, simply identifying the failure events themselves does not provide enough information about their cause, or the possible remedy. For instance, discovering that a node has stopped responding does not indicate whether it has suffered a node failure, a link failure, or a software crash. In order to get a better understanding of these failures, we may want to employ additional information, such as periodic data collected by monitoring systems and system logs that are routinely collected in such systems. However, the scale of these systems typically makes it difficult to analyze these vast amount of data without any prior domain knowledge, and thus limited in their utility to make higher-level inferences.

In this paper we advocate *co-designing* the system monitoring and failure analysis for large-scale distributed systems, where the needs of failure inference and diagnosis can be synergistically incorporated into the system monitoring. This should be done in such a manner that it provides an administrator control over the collection of data (e.g., in terms of both the granularity and types of data) to meet their needs while addressing scalability and efficiency concerns at the same time. Such a position is based on our experience in attempting to perform failure classification, inference and diagnosis using an existing large-scale distributed system monitoring infrastructure (the CoMon system for Planet-Lab [8]), where due to the limitations of the types and granularity of data collected, we had limited success in accomplishing our failure analysis goals. We believe this method-

ology of co-design is likely to provide more accurate and timely diagnosis and troubleshooting capabilities for large-scale distributed systems, while at the same time it is likely more scalable and less complex by obviating the need for handling extraneous data or requiring explicit human intervention throughout the process.

The remainder of the paper is organized as follows. We begin by presenting a brief overview of existing large-scale failure analysis and monitoring systems in Section 2. In Section 3, we present a failure analysis case study for PlanetLab using the CoMon monitoring system, in which we identify some successes and limitations of the existing monitoring system. In Section 4, we present a set of recommendations and guidelines to co-design a cooperative failure analysis-monitoring system that would meet the failure analysis goals. We finally conclude in Section 5.

2. RELATED WORK

2.1 Failure Analysis and Prediction

Prior studies have focused on failure characterization [15], availability prediction [14], and churn characterization [9] in large-scale systems. Many of these studies have focused on predicting node-level failures. While these studies provide useful insights into failure characteristics, they suffer from several limitations. First of all, most of them rely only on failure characteristics such as failure durations, or failure occurrence times to characterize these failures. However, as we will show in the next section, this information by itself does not provide complete insights into the failure causes. Secondly, many of these studies ignore correlations between multiple failures, and consider most errors to be independent. Statistical cross-node correlations are considered in [15], however, it provides limited insights into the reasons behind these correlations. Finally, many of these studies focus only on the node or network-level failures, or else do not distinguish between them. Overall, while such studies may provide useful prediction/characterization models, they do not provide a better understanding of the failure causes, or the correct remedy to overcome their impact. In [19] authors analyzed the failures caused by different applications and in [18] the use of past failures to provide an automated diagnosis of future failures was suggested. However these works were limited to identification and diagnosis of node-level failures only and did not look at the interaction of failures occurring in a large-scale system.

2.2 Monitoring

Monitoring data can be useful for failure analysis, and several monitoring systems have been deployed in large-scale systems. These designs include i) CoMon [17]: centralized resource monitoring system for PlanetLab, ii) CoTop [16]: a centralized resource monitoring system for distributed services in PlanetLab, iii) PlanetSeer [20]: a monitoring system for failures in the communication network, and iv) Ganglia [13]: a monitoring system for grid/ cluster systems used for high-performance computing. Table 1 shows a comparison of these different monitoring systems.

Most of these systems have been designed for specific purposes, e.g., CoMon is designed mainly to monitor the resource usage and connectivity of PlanetLab nodes and slices,

while Ganglia is mainly designed for accounting of node-level resource usage. While the monitoring information provided by these systems can be useful, they are not inherently designed to provide metrics specifically suited for failure analysis. For instance, CoMon has a centralized architecture, which is suitable for its intended use, but it is prone to failures of the monitoring node itself. Similarly, while Ganglia uses a scalable and distributed design, which is more robust, it is not suitable for large-scale systems because of its reliance on a static tree topology, prior knowledge of node identities, and limited data aggregation capabilities.

3. FAILURE ANALYSIS ON PLANETLAB: A CASE STUDY

In this section we present our experience in performing failure analysis of the PlanetLab system using the monitoring data collected by CoMon [17]. Here, we give a brief overview of our study, and details can be found in a technical report [10]. We chose PlanetLab for our study, because it is a widely used networking testbed consisting of autonomous nodes running a large variety of applications and experimental services that stress the system in a variety of ways and lead to frequent failures. The goal of our failure analysis is to understand and classify failures based on their characteristics so as to help us diagnose possible causes of failures. Using the monitoring data collected by CoMon, we set out to answer the following basic questions: i) can we infer and distinguish between *hard* (i.e., machine crashes) vs. *soft* (i.e., application crashes or network outages) failures; ii) can we infer and distinguish between (random) node failures and correlated failures due to, for instance, site-wide power outage, network instabilities, or application-level or slice-wide correlated failures? In the following we first provide a brief overview of the data collected by CoMon that is used in our study, and then present our experience and lessons learned in our study. We use these insights to provide guidelines for co-designing monitoring and failure analysis systems in the next section.

Dataset Description: PlanetLab deploys a centralized monitoring infrastructure called CoMon [17] that collects and reports statistics on active PlanetLab nodes. The main purpose of this system is to enable PlanetLab administrators to spot problematic machines and/or slices (applications) running on them. The system is designed in such a way that a central monitoring node located in Princeton queries daemons running on the remote nodes every five minutes and collects data. The data consists of various metrics such as uptime, CPU and memory utilization, CPU load, etc. Some of these metrics are actively measured by CoMon, e.g., CPU available to a spin-loop program (i.e. Free CPU), while other metrics are passively measured or synthesized, e.g., number of live slices. If a PlanetLab node is reachable, it responds to the queries with its node-level/application-level information, which is written to a central repository. If a node is unreachable, the central monitor simply records a query failure with a corresponding error message. The archived data resides in the repository which is publicly accessible through the CoMon website [2]. For our analysis, we used a 3-month long (Dec'06-Feb'07) data trace collected by CoMon.

Table 1: Comparison of different monitoring systems

| Monitoring System | Design | Active/Proactive/Reactive | Monitoring Task |
|-------------------|------------------------|-----------------------------|---|
| CoMon | Centralized | Active Monitoring | Node level resource monitoring |
| CoTop | Centralized | Active Monitoring | Distributed Application (slice) level resource monitoring |
| PlanetSeer | Decentralized | Reactive/Passive Monitoring | Failures in Wide-area networks |
| Ganglia | Distributed Monitoring | Active Monitoring | Cluster/grids |

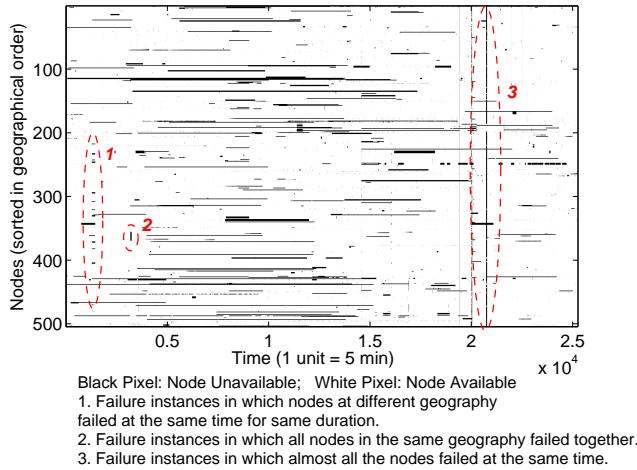


Figure 1: Availability time-series of nodes in PlanetLab for 3-month period.

3.1 Using Multi-dimensional Views to Understand Failures

Using the monitoring data we first extract the node up-down¹ binary series. We use this up-down series to classify the failures along two dimensions: duration and size² of group failure. We then use *Uptime* information to distinguish between hard and soft failures and show interesting characteristics of hard and soft failures along duration and size dimensions. Next, we distinguish between (random) node-failures and correlated failures by using additional information available from CoMon. We also infer possible cause of failures using local resource usages (CPU, Mem) and socket error messages.

Figure 1 shows a graphical view of the per-node failures observed in PlanetLab during the 3-month period of our study. The key observation from this figure is that failures have many diverse characteristics. For instance, the failures differ widely in terms of their durations; some failures appear to occur together while others appear to be independent³; the number of such co-occurring failures also varies widely with time. The question is whether these different characteristics can be explained using any additional information that is gathered by the monitoring system. In particular, is

¹UP: CoMon received periodic monitoring data from this node; DOWN: Failure in collecting the periodic monitoring data for the node from CoMon node.

²Failure group size is defined as number of nodes failing together.

³There is an interesting period (circled in the figure) during which *all* nodes appear to have failed.

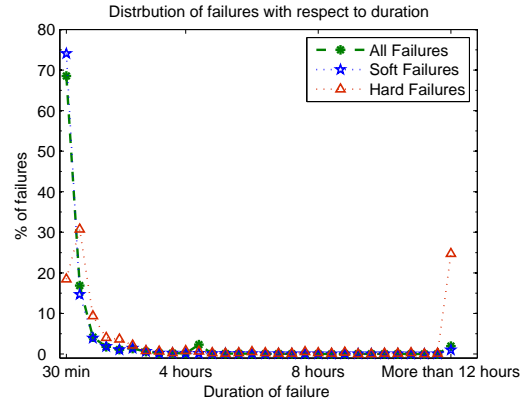


Figure 2: Distribution of failures with respect to duration.

there an underlying difference among the different kinds of failures that can explain their different properties?

To understand the basic characteristics of failures, we first examine the distribution of failures along *failure duration*. Intuitively a failure that lasts for few minutes would have a different cause behind it than a failure which lasts for more than a day. For instance, a transient overload or a network failure can cause a node not to respond for a small duration but this may not last for a long time. A relatively longer duration failure can be caused by power outage, maintenance routines, etc., and still longer duration failures might happen due to hardware malfunction which requires human intervention and can take days to fix.

In Figure 2 we plot the distribution of failures with respect to their duration. As seen in this figure a majority of the failures last for very small durations; however there is a good fraction of failures which last for longer durations. Based on our intuitive understanding, it appears that long-duration failures might be hard failures during which a machine was shutdown and required manual intervention. On the other hand, short-duration failures might happen because of transient conditions and hence are soft in nature⁴. This motivates us to look at how hard and soft failures are distributed along failure duration dimension. Figure 2 shows the distribution and we provide summarized statistics for hard and soft failures in Table 2. We see that most of the soft failures last for small duration and hard failures tend to last for long duration. However surprisingly, we also see that there are some instances of soft failures which tend to last for very long duration, and some instances of hard failures that last

⁴We use resets of uptime counter to separate hard-failures from soft-failures.

Table 2: Comparison of distributions for hard and soft failures

| | Hard Failures | Soft Failures |
|-------------------|---------------|---------------|
| Frequency | 623 | 20670 |
| Total Downtime | 1500 days | 766 days |
| Mean Duration | 34 hrs | 53 min |
| Failures per node | Mean = 2.1 | Mean = 41 |

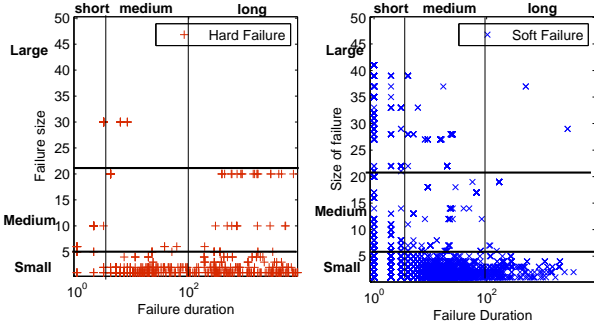


Figure 3: Hard and Soft failure in comparison to duration and size of failures.

for very small durations. We are unable to explain these failures with the available data.

To get further insights into the nature of hard and soft failures, we use socket error message information provided by CoMon. These error messages are received by the monitoring node when it is unable to reach a remote node. For instance, ‘Connection Refused’ error message suggests a failure of monitoring daemon on the remote node while the node is up, similarly ‘Network unreachable’ suggests a possible network failure. For a large number of both hard and soft failures, we saw ‘No Response’ error message which does not provide any clue about the possible problems. However, to our surprise, during some of the hard failures we found a long sequence of ‘Connection Refused’ error message. One possible reason for this could be that machine was highly unstable during this time and kept on rebooting again and again. However in CoMon, data is collected only at a five-minute interval, which is too coarse-grained to conclude that the machine was indeed restarting during five minutes. (For detailed information, please refer to tech-report [10].)

Now we turn our attention to the other dimension called failure group-size. Thus far we have attached with each failure information about its duration and whether it was hard/soft; we now add group-size information to these failures. Results are shown in Figure 3. We see that most of the large-size failures are soft and last for short durations. This can be explained by the fact that it is highly unlikely for a large number of machines to go down all at once, and the actual cause may be a network or monitoring-side problem. We also observe that a large number of both hard and soft failures are small in size. We investigate small-size failures further in order to see if they are correlated; this correlation can also be seen in Figure 1 (failures marked as 1 and 2). In failures marked as 1, we see that nodes at different geographical locations fail at the same time and for

same duration. Our initial hypothesis that this might be an application-level error proved wrong as we did not observe much correlation between application-level resource usage information and failures on these nodes. We attribute this to the fact that applications operate in a constrained environment in PlanetLab and are limited in terms of resources that they can use. Failures marked as 2 are the failures in which nodes in the same geographical region failed at the same time. Small-size of these failures led us to explore if these failures could be site-wise correlated. By site-wise correlation we mean that nodes at the same PlanetLab site failed together due to routine maintenance. We found few instances of failures when all the nodes belonging to a site were down and confirmed these by messages posted by site-administrators on PlanetLab user-lists. Lastly, in Figure 1 we also see some large-size failures (marked as 3) which show that nearly all the nodes in the PlanetLab system were down all at once. In reality this scenario is least likely to occur and this led us to hypothesize that it might be a failure of monitoring nodes at Princeton. Through our communication with PlanetLab administrators, we found that this was indeed the case for some of these instances.

To investigate some of the non-correlated node failures, we looked at the usage of different resources on the nodes just before it failed. We expected that resource usage just before a failure might appear abnormal (high) when compared to normal functioning of the machine. We did multiple experiments to analyze any correlation that might exist between usage of resources such as CPU, memory, network bandwidth and failures on a node; however we found only weak correlations with CPU usage (these results can be found in [10]). We observed that this might be due to coarse granularity of data collected by CoMon and more fine-grained information might be able to explain some of these failures.

While we are able to get some insights into PlanetLab failures using multi-dimensional views of the CoMon data and incorporating additional information, we are unable to explain or understand several failure instances. However doing this study helped us understand shortcomings of CoMon for our purposes and we present some of the lessons we learned.

3.2 Lessons Learned

- *Inappropriate data granularity:* CoMon collects data at 5 minute granularity, which is too coarse for some events such as transient overloads or network fluctuations, but too fine-grained for several events, such as node shutdowns for maintenance. Therefore, we had insufficient data in some cases, and too much redundant data in other cases, leading to lack of evidence or data over-fitting.
- *Lack of appropriate metrics for failure analysis:* CoMon is primarily designed for system-level monitoring, and therefore does not collect other information that might make it easier to infer failure causes. An example of such information could be application-level performance metrics, e.g., response time, throughput, etc., that could provide indications of application health.
- *Lack of diverse view-points of the system:* CoMon uses a centralized monitor, which is sufficient for the scale of PlanetLab, and also serves the primary purpose for which CoMon

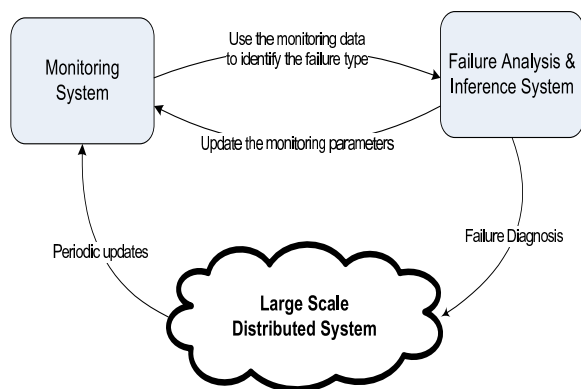


Figure 4: Co-design of the Monitoring and Failure Analysis System

is set up. However, using this data to infer failures has the pitfall that monitor-side failures can be falsely attributed to system-wide failure events. This phenomenon is observed as the circled time period (marked as 3) in Figure 1, where a monitoring-side network failure prevented the central monitor from connecting to other nodes in PlanetLab.

- *Lack of feedback between analysis and monitoring:* In our study, we used a statically monitored dataset for failure analysis. However, the quality of the analysis could be improved if system can be made adaptive by introducing a feedback loop from the analysis to the monitoring system.

4. DESIGN PRINCIPLES

Motivated by our experience with failure analysis using an existing monitoring system, we advocate co-designing monitoring and failure analysis of large-scale distributed systems (Figure 4) to accommodate both the needs of failure analysis (including failure classification, inference, diagnosis or root-cause analysis) as well as the scalability and efficiency concerns of system monitoring. In the following we lay out a few key design principles for such co-design.

Allow flexibility in data collection and failure analysis: The first design decision that one should make is what data to collect and how frequently it should be collected. For the purpose of failure analysis, ideally we would like to monitor and collect as much data as possible and as fine-granularity as needed. Unfortunately due to scalability requirements, overhead on the monitored system, storage requirement, etc., this is often not feasible. To accommodate trade-offs, it is important to i) allow flexibility in monitoring system in terms of being able to collect additional data at different granularities, based on failure analysis needs; and ii) allow flexibility in the failure analysis methodology to provide different degrees of insights based on the quality of data available, e.g., limited insights if only coarser-grained, fewer dimensional data is available, and more accurate and detailed insights if richer, more relevant and finer-grained data is available. With built-in flexibility in data collection and failure analysis, we can then configure and tailor the system monitoring and failure analysis in accordance to the basic operation requirements, and adapt and adjust them based on the varying system conditions and failure analysis

needs, as discussed below.

Separating short-term vs. long-term monitoring data, and what is locally stored and what is reported: Failure analysis often requires monitoring and collecting fine-grain statistics (e.g., CPU usage right before a failure caused by system overload) and other relevant data (e.g., syslog) that are closely associated with a failure. On the other hand, since failures do not happen all the time, such short-term fine-grain statistics and additional data types/sources are not useful when there are no failures. In other words, under normal circumstances coarse-grain statistics (averaged over certain time scale, e.g., 5 minutes) would be adequate in providing a general picture of the overall health of the system. In addition, to reduce the communication and storage overheads, only certain types of data collected locally at a machine—especially those that are useful for failure diagnosis—need to be reported to a centralized data collection facility. For example, one possible approach to accommodate these trade-offs is to adopt a sliding window for monitoring and collecting short-term, fine-grain data, and report only average statistics over a longer period of time under normal operating conditions. When a failure occurs, the immediate short-term statistics and other relevant data collected before the failure would thus be available and could be retrieved later for failure analysis when the failure has been recovered.

Adapt to varying system conditions: The monitoring system should be designed in such a way that it can adapt based on varying system characteristics and conditions. For example, fine-grain CPU and other statistics or data types may be recorded only when certain thresholds have been exceeded. In addition, the types and granularities of data being monitored and collected can further be adjusted based on the prior history, analysis needs and feedback provided by the failure analysis system. For instance, if certain kinds of failures are more frequent in a system or certain nodes are more prone to failures, the monitoring system should detect that, figure out the relevant data that needs to be collected, or increase/decrease the frequency with which data is collected.

Multiple viewpoints and redundancy in monitoring data: To enable effective system monitoring and failure analysis, it is also important to incorporate multiple viewpoints and redundancy in monitoring data, for example, using a hybrid of distributed monitoring, local data collection and centralized reporting for monitoring data collection and failure analysis. Such an approach would overcome the problem of limited viewpoints or lack of data due to failures closer to a centralized monitoring system, and can help differentiate between failures happening due to software or hardware problems that affect only a single machine, and those caused by power or network failures that affect one site, or a larger geographical area. Redundancy in monitoring data can also provide overlapping or correlated views of the system characteristics to overcome loss of monitoring data (e.g., due to failures affecting one or multiple monitoring devices) and facilitate detecting other types of correlated failures such as application-level/slice-level failures, security attacks, and so forth.

5. CONCLUSION

Nodes in a large-scale distributed system show many diverse patterns of failures and finding what caused these failures is a very hard problem. In our study, we looked at multiple dimensions of failure characteristics and employed a variety of other data analysis techniques to understand these failures. Our techniques provided us with new insights into node-failures and we were able to explain possible causes of some of these failure instances. However, we also found that this is not sufficient to explain and understand all the failures. Therefore we argue co-designing monitoring system with the failure analysis system and based on our experience, we provide design principles to serve as guidelines when building such a system.

6. REFERENCES

- [1] Akamai: Content distribution network. <http://www.akamai.com>.
- [2] Comon-a monitoring infrastructure for planetlab. Available at <http://comon.cs.princeton.edu>.
- [3] Gnutella. <http://www.gnutelliums.com/>.
- [4] napster. <http://www.napster.com/>.
- [5] Restarts cited in skype failure. Available at <http://www.nytimes.com/2007/08/21/business/worldbusiness/21skype.html>.
- [6] D. Anderson. BOINC: A System for Public-Resource Computing and Storage. *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 365–372, 2004.
- [7] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.
- [8] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.
- [9] P. Godfrey, S. Shenker, and I. Stoica. Minimizing churn in distributed systems. In *Proceedings of the ACM SIGCOMM Conference*, sep 2006.
- [10] S. Jain, R. Prinja, A. Chandra, and Z.-L. Zhang. Failure Classification and Inference in Large-Scale Systems: A Systematic Study of Failures in PlanetLab. Technical Report 08-014, Department of Computer Science, University of Minnesota - Twin Cities, 2008. Available at http://www.cs.umn.edu/research/technical_reports.php?page=report&report_id=08-014.
- [11] S. Larson, C. Snow, M. Shirts, and V. Pande. Folding@ home and genome@ home: Using distributed computing to tackle previously intractable problems in computational biology. *Proceedings of the Computational Genomics*, 2002.
- [12] M. Litzkow, M. Livny, and M. Mutka. Condor-a hunter of idle workstations. *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111, 1988.
- [13] M. Massie, B. Chun, and D. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Proceedings of the Parallel Computing*, 30(7):817–840, 2004.
- [14] J. Mickens and B. Noble. Exploiting availability prediction in distributed systems. *Proceedings of NSDI'06 Symposium on Networked Systems Design & Implementation*, pages 6–6, 2006.
- [15] S. Nath, H. Yu, P. Gibbons, and S. Seshan. Subtleties in tolerating correlated failures in wide-area storage systems. *Proceedings of NSDI 3rd Symposium on Networked Systems Design & Implementation*, 6, 2006.
- [16] K. Park and V. Pai. CoTop: A Slice-Based Top for PlanetLab. <http://codeen.cs.princeton.edu/cotop/>.
- [17] K. Park and V. Pai. Comon: a mostly-scalable monitoring system for planetlab. *ACM SIGOPS Operating Systems Review*, 40(1):65–74, 2006.
- [18] C. Verbowski, E. Kıcıman, A. Kumar, B. Daniels, S. Lu, J. Lee, Y. Wang, and R. Roussev. Flight Data Recorder: Monitoring Persistent-State Interactions to Improve Systems Management. *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, 2006.
- [19] C. Yuan, N. Lao, J. Wen, J. Li, Z. Zhang, Y. Wang, and W. Ma. Automated known problem diagnosis with event traces. *Proceedings of the 2006 EuroSys conference*, pages 375–388, 2006.
- [20] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang. PlanetSeer: internet path failure monitoring and characterization in wide-area services. *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation*, pages 12–12, 2004.