The importance of understanding distributed system configuration

David Oppenheimer
Computer Science Division, EECS Department, UC Berkeley
davidopp@cs.berkeley.edu

We recently conducted a study of the causes of failure in three large-scale Internet services. The results pointed to operator error as a key cause of system unavailability—operator error was the largest single cause of failure for two of the services, and it was responsible for at least 19% of failures in the third. Moreover, averaged across the three service, operator-induced failures took longer to repair than did any other cause of failure. In looking for commonalities among these operator-induced failures, we found that system configuration errors *far outnumbered other types of errors (e.g., accidental deletion of data, moving users to the wrong fileserver, etc.). We also found that many operator-induced failures could have been prevented had the operator better understood the system's configuration, and in some cases how the system evolved to that configuration. In this position paper we describe the general configuration problem for Internet services, cite several case studies of configuration-related operator errors, and make suggestions for reducing the number of failures caused by incorrect and misunderstood configurations.*

Internet services are highly complex beasts built from hundreds, thousands, or even millions of interacting components. For better or worse, almost all of these components are configurable. While an application, server, or network switch may offer configuration options to control its own internal behavior (*e.g.,* performance tuning knobs for an operating system or database), the most troublesome configuration issues arise in specifying how components are to interact with one another. Some such cross-component dependencies are specified in configuration files, *e.g.,* a network switch's list of server pool nodes among which it load balances requests. Others are specified only in software, *e.g.,* two communicating software components implementing a certain version of a protocol. And some are specified nowhere, *e.g.,* the knowledge that site A alone or sites B and C together of a three-site service can handle the service load, but that site B or site C alone cannot, meaning that site A should never be taken offline for maintenance at the same time as any other site. Moreover, due to the possibility of cascading failures, configuration options that control cross-component interactions are more likely to have global effects than are single-component ones. Because Internet services are built from many components with configuration options that control complex interactions at the process, node, application, datacenter, and global scales, the tasks of understanding existing configurations, and establishing new ones, are currently complex and error-prone. When an error is made while performing these tasks, a user-visible failure often results, or an existing failure is prolonged.

Incorrectly or insufficiently understanding system configuration influences service availability in two ways:

1. **An operator performing any kind of action on a service** (*e.g.,* establishing a new configuration, adding a new back-end node, moving a component or user from one server to another, deleting files considered unnecessary, *etc.)* **may perform the task incorrectly if she does not understand the existing system configuration.**
2. **When diagnosing a problem** (be it operator-induced or not), **an operator must understand the existing system configuration—and sometimes the history of system configurations before the problem began**--in order to follow cause-and-effect chains back to the problem's root cause.

We recently conducted a study of the causes of user-visible failures in Internet services by reading over 500 problem reports from three large-scale (~10-100 million hits per day) geographically distributed Internet services [2]. Qualitatively we found significant difficulties related to both of the above configuration issues, often leading directly to service unavailability or lengthening the time to diagnose or repair a failure. Quantitative evidence indicates trouble with establishing configurations: we found that (i) operator error was the largest contributor to service failure in two of the three services; (ii) averaged across all three services, operator error had the largest time to repair of any failure cause; (iii) of the operator errors leading to service failure, more than 50% (and for one service 100%) were due to faulty configurations; and (iv) better sanity checking of configuration files would have prevented 9 of 40 failures we examined. On the issue of problem diagnosis and repair, we found that better exposing problem sources and their relationship to the system's configuration would have decreased time to diagnose problems in 11 of 40 failures and would have reduced time to repair problems in 12 of 40 failures.

Three incidents were particularly instructive in demonstrating the importance of operators understanding system configuration. These and other such case studies are described in more detail in [2].

Our first case study demonstrates principles (1) and (2) above. Operations staff were notified that users were complaining that the service was occasionally "losing" their newsgroup postings. In correct operation, a user's posting to a local service newsgroup is intercepted by the service's front-end news daemon, converted to email, and sent by email to a back-end email server dedicated to handling postings to local newsgroups. When email is delivered to the appropriate newsgroup account on that machine, a script is triggered that contacts a daemon on the email server to verify that the posting user is a valid service subscriber. If the user is verified, the message is posted to the newsgroup; otherwise the message is silently dropped. Unfortunately an operator inadvertently removed the user identity lookup daemon from the mail server's configuration because she was not aware of the news posting script's dependence on this component. The script treated the absence of the daemon as a failure to verify the user, and therefore dropped all news postings it received. Moreover, because this error only affected postings to the service's local newsgroups (as opposed to regional, national, and international groups), operators experienced difficulty in tracking down the cause of the problem because they could not readily see that the "failed" postings depended on the misconfigured email server, while the "successful" postings did not.

A second problem we studied also demonstrates principles (1) and (2), and furthermore shows how configuration error causes and effects can span administrative boundaries. One service in our study (an online service we'll call *Online*) uses an external provider (which we'll call *Other*) for one of its services[1]. An operator at *Other* attempted to increase her service's security by restricting the IP addresses from which users could connect. She was unaware that legitimate requests could originate from machines at *Online*, and therefore excluded them, blocking legitimate users whose requests originated at *Online*. Operators at *Online* had a great deal of trouble diagnosing this problem because they were never notified that *Other* was making a change, and they therefore looked for the problem cause within their service. As the number of web services built by composing independently-operated single-function services increases, it will become increasingly important that services understand not only their own internal configuration, but also how their service depends on the configuration of external services.

Our final example demonstrates principle (1). One of the services we studied stores its problem tracking database in a commercial database. This database was to be backed up to tape on a regular basis, and to be mirrored nightly on a machine in a remote datacenter. Unfortunately the backup program had not been running for over a year because of a configuration error related to how the database server is to contact the backup host. That issue was considered low priority because the nightly mirroring ensured a second copy of the data. One night, the disk with the problem tracking database's primary copy failed, leaving the mirror as the only copy. Unfortunately, later that night an operator reformatted the mirror host, not realizing that it held the (now only) copy of the database. This incident underscores the importance of operators understanding the configuration, dependencies, and state of a service, particularly before performing a destructive operation. The operator should have copied the database from the mirror before reformatting that host—but knowing that this was necessary required understanding what function the mirror host was serving, and that in the current system configuration it held the only copy of the problem tracking database. Like this service, many systems use redundancy to mask failures--but it is particularly important in such systems that operators know when a system's configuration has changed to one with a reduced margin of safety.

Having argued the importance of understanding configuration in maintaining system availability, we ask what types of tools might help operators understand configurations and reduce the incidence of misconfiguration. We believe that building operator tools that interface with existing systems provides an incremental path to improving maintainability (and thus availability) until the day when systems are truly built from the ground up considering operators as a first-class user. Such tools should not be mere wrappers around command-line installation and configuration utilities as are many of today's system administration tools. Instead, the complexity of component relationships and configuration options suggests that radically new tools are needed for tasks such as visualizing current and past configurations, and predicting the global impact of changes in component configurations.

[1] Unfortunately we cannot be more specific here, as the services whose failure data we studied required us to eliminate all information that could possibly be used to identify them.

In addition to tools that help operators visualize configurations and dataflow relative to those configurations, we believe that operators would benefit from tools that check to make sure that per-component configuration settings match the desired high-level service architecture and desired high-level service behavior. In our study, we found that such tools could have prevented faulty configurations in 9 of 40 end-user-impacting failures. In keeping with our incremental philosophy, such a tool could at first check simple sanity constraints (*e.g.,* that one or a set of configuration files should not be self-contradictory), and could then be extended over time to check operator-defined rules that reflect high-level service architecture and goals. Such a specification is a form of semantic redundancy, a technique that is useful for catching errors in other contexts (types in programming languages, data structure invariants, and so on). Extending this idea, given a high-level description of desired service architecture and behavior, a more ambitious tool could attempt to automatically generate appropriate per-component configuration files. For incremental deployment of this idea, tools of the sort described in the previous paragraph are also needed, to help the operator understand the existing configuration of a properly-functioning system so that she can write a correct abstract specification of "correct" system behavior. Such tools would also help the operator to understand an automatically-generated configuration, building operator confidence in automated configuration tools.

We further believe that tools for understanding and establishing configurations should be built to support cooperative work by multiple individuals within and among administrative domains. Today's Internet services are operated by distributed teams of administrators and depend on coordination among those teams and with a service's software developers and vendors, collocation facility staff, network providers, and customers. When establishing configurations or diagnosing problems, all of these entities may be involved. Today this coordination is largely handled manually, via telephone calls among responsible personnel. Tools for cooperative work could make the coordination process run much more smoothly. Indeed, simply sharing a unified problem tracking and bug database among all entities involved in a service's operation would be a significant improvement over today's separate per-organization bug and operations problem tracking databases. One can imagine more sophisticated distributed workflow tools customized specifically to support service deployment, upgrade, problem diagnosis, and problem repair.

Finally, we believe that operators would benefit from tools that track configuration history and system health over time. The problem tracking databases used by many Internet services are an important step in this direction, as they give operators a history of all important actions performed on the system in response to a problem, along with annotations of why the actions were performed. Integrating such databases with health monitoring data, bug database data, and configuration and change management data would provide a unified mechanism to clearly expresses the entire history of a system. The history could include parameters during normal and abnormal operation, configuration and system state before and after a change, who or what made the change, why they made the change, exactly what they did to make the change, and what problems may have resulted from a change. Such a history would be an invaluable aid in problem diagnosis, post-mortem failure analysis, and system evolution.

In conclusion, the task of understanding system configuration is crucial to reliable Internet service operation, but it is difficult and error-prone using today's tools. Quantitative and qualitative data we collected from three large-scale Internet services indicates that improving such understanding would reduce service time to failure as well as the time needed to diagnose and repair problems. Our data leads us to suggest several potential techniques for easing the task of understanding and establishing system configurations.

References

[1] David Oppenheimer, Aaron B. Brown, Jonathan Traupman, Pete Broadwell, and David A. Patterson. *Practical issues in dependability benchmarking. Second Workshop on Evaluating and Architecting System Dependability (EASY)*, 2002.

[2] David Oppenheimer, Archana Ganapathi, and David A. Patterson. Why do Internet services fail, and what can be done about it? To appear in *4th USENIX Symposium on Internet Technologies and Systems, 2003*.