Advanced Tools for Operators at Amazon.com

Peter Bodík[‡], Armando Fox[†], Michael I. Jordan[‡], David Patterson[‡],
Ajit Banerjee[§], Ramesh Jagannathan[§], Tina Su[§], Shivaraj Tenginakai[§], Ben Turner[§], Jon Ingalls[§]

[‡]RAD Lab, UC Berkeley, [†]Stanford University, [§]Amazon.com

Abstract

Despite significant efforts in the field of Autonomic Computing, system operators will still play a critical role in administering Internet services for many years to come. However, very little is know about how system operators work, what tools they use and how we can make them more efficient. In this paper we study the practices of operators in a large-scale Internet service Amazon.com and propose a new set of tools for operators. The first tool lets the operators explore the health of system components and dependencies between them; the other monitors the actions of operators and automatically suggests solutions to recurring problems.

1 Introduction

Large-scale Internet services invest significant amount of time and money to achieve high availability of their service, typically in the form of labor-intensive monitoring and response teams. Despite the high human resource expenditures, software and hardware failures still occur, and the human-intensive approach to monitoring and troubleshooting scales poorly as the service's complexity and workload grow [1]. We hypothesize that the right kind of visualization and automation can reduce the human effort required for monitoring and repairing failures, allow operators to more quickly recognize a problem as recurrent, and facilitate better knowledge transfer in quickly-growing or high-turnover teams.

One of the authors spent three months working alongside the Amazon.com team responsible for real-time monitoring of hardware and software and for providing monitoring tools for the rest of the company. We collected quantitative data via interviews and surveys and analyzed the data in the trouble ticket database that contains information about each failure in the last few years. Based on our observations, we identify three challenges that make failures difficult to find and fix, and describe prototypes and early evaluation of two tools designed to address these challenges.

In particular, we find that the main challenges in troubleshooting and monitoring in the Amazon.com environment are:

1. Complex dependencies among system components can cause failures to propagate to other components, triggering multiple alarms and complicating root-cause determination.

2. Operators would be more effective if they had more *situation awareness*—they have local knowledge of the behavior of a small part of the system, but no individual understands all the dependencies among different parts of the system.

3. The whole system is heavily instrumented at a fine grain, but even though many problems can ultimately be characterized in terms of the behavior of a dozen or so metrics, the total amount of information collected—the combination of metrics multiplied by the number of machines—can be overwhelming and makes it difficult for operators to analyze problems.

We present two tools that aim to help the operators deal with these challenges in diagnosing and fixing problems.

2 Amazon.com Operations

2.1 Failures and Resolvers

Amazon.com's approach distinguishes two major failure types: Severity 1 (*sev1*) and Severity 2 (*sev2*). Sev1 problems affect customers directly and need to be resolved immediately, and rarely recur. Sev2 problems do not immediately affect the behavior of the website, but could turn into sev1 problems if not resolved quickly. Sev2s typically affect only a single system component and tend to recur, so the operators learn to recognize and fix them. However, they are about 100 times more frequent than sev1 problems, in part because of rapid churn of the site software: just in the Monitoring team, the online documentation repository registered hundreds of changes per month during a four-month period last year, and the deployment system registered an average of over a hundred code changes per month being rolled into the production system (excluding any testing or debugging-related deployments). Furthermore, turnover among sev2 responders is higher than among sev1 responders, as we explain below.

Fewer than a dozen operators monitor the health of the whole web site 24×7. Their task is to monitor for sev1 failures, perform a rapid troubleshooting, and immediately page the affected services' primary resolvers, who are expected to respond in 15 minutes. The primaries then perform extensive troubleshooting and recovery actions.

A few tens of software teams are responsible for designing, implementing, deploying, and maintaining one or more of the several services that collectively comprise the site's functionality, both customer-facing services and services that support the site's infrastructure. This means that *most software developers are also problem resolvers* and are part of an *on-call rotation*. During his rotation of a few consecutive days, the developer/resolver becomes the *primary resolver* for services owned by his team. All told, then, there are hundreds of resolvers working at Amazon.com.

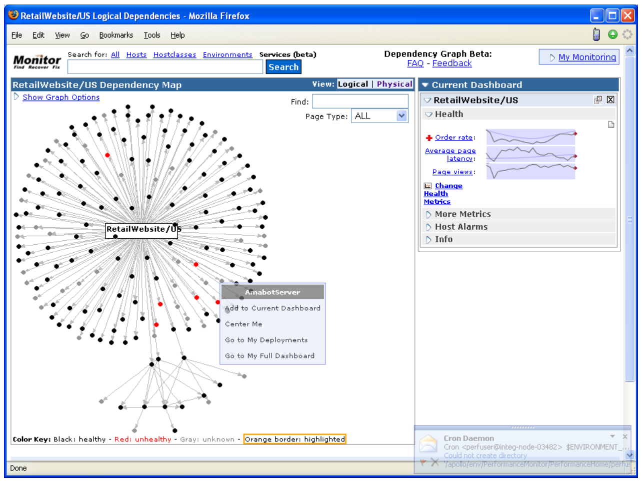
RetailWebsite/US Logical Dependencies - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Monitor Find Recover Fix
Search for: All Hosts Hostclasses Environments Services (beta)
Search
Dependency Graph Beta: FAQ - Feedback
My Monitoring
RetailWebsite/US Dependency Map
View: Logical | Physical
Show Graph Options
Find:
Page Type: ALL
RetailWebsite/US
AmabotServer
Add to Current Dashboard
Center Me
Go to My Deployments
Go to My Full Dashboard
Color Key: Black: healthy - Red: unhealthy - Gray: unknown - Orange border: highlighted
Current Dashboard
RetailWebsite/US
Health
Order rate:
Average page latency:
Page views:
Change Health Metrics
More Metrics
Host Alarms
Info
Cron Daemon
Cron <perfuser@integ-node-03482> $ENVIRONMENT_...
Could not create directory
'/apollo/env/PerformanceMonitor/PerformanceHome/perfu
Done

Figure 1. Maya in action: the directed graph visualizes the system components, their health (red/black), and the dependencies between them. The square in the top-right corner is a dashboard for one of the components showing the most important metrics, alarms, and other information.

2.2 Tools

Most of the data used by operators and resolvers for troubleshooting consists of real-time hardware, operating system, network, and application-level metrics collected from every machine at fine grained intervals and stored in a central database. In-house web-based tools can be used to graph the metrics, offer predictions of metrics based on historic values, and set threshold alarms on all metrics. Other tools correlate various events like sev1/sev2 problems or deployments of new software to hosts. These general monitoring tools are sometimes inappropriate for some special tasks; the monitoring team therefore offers programmatic access to the data via APIs, and resolvers often build custom tools tailored to needs of their team.

Lastly, resolvers sometimes run shell commands manually on individual hosts to examine system logs, look for error/exception messages, or restart processes and other software components, although this practice is receding as the web-based tools mature.

3 Troubleshooting Sev1 Problems

The small group of "level 1" operators is charged with detecting sev1 problems, which usually manifest themselves as anomalous traffic levels to parts of the site whose traffic patterns are well known. The level 1 operators perform initial troubleshooting to identify the affected services, and then page the primary resolvers of each. The resolvers are expected to respond in 15 minutes and join a conference call initiated by the operators to determine which service(s) are actually responsible so the problem can be assigned to the correct team. If the problem is not resolved in a specified period, it is escalated to higher management and more people join the phone call.

As mentioned earlier, the obstacles to level-1 troubleshooting are failure propagation, lack of global dependency knowledge, and overwhelming amounts of low-level information.

3.1 Dependencies and Propagation of Failures

Because of dependencies between system components, an issue common to many sev1 problems is that a failure of one component may affect other components that will also report alarms and anomalies. Since in most cases all the alarms are triggered within a few minutes, causality cannot be established by analyzing the timestamps of alarms. This complicates the root-cause diagnosis because it's not clear which of the components is actually failing. In these cases, many primary resolvers are initially paged even though not all of them will end up directly involved in problem resolution. Indeed, our analysis showed that many sev1s are misdiagnosed and consequently assigned to a different team.

3.2 Situation Awareness

Members of each team usually know which other services they depend on; i.e., over time they learn the local neighborhood around their service in the dependency graph. But nobody remembers dependencies for all services in the company, and there are no good tools for visualizing the dependencies. As a result, the operators and resolvers don't always see "the big picture", making it hard to diagnose sev1 problems in which one failing component affects other components that depend on it.

Overwhelming Amount of Information

Amazon.com collects a few million metrics from all their datacenters. These metrics include about 100 hardware metrics from every host (CPU and memory utilization, I/O, swap space, and network interfaces) and application-level metrics such as latency, availability, and error rate of each service. On the one hand, the monitoring tools allow operators and resolvers to monitor all hosts in real time. On the other hand, the sheer volume of information can be overwhelming, making it hard to analyze all metrics from a single service running on hundreds of machines.

A similar problem is posed by alarms which are manually set by resolvers. Some of the alarms are too sensitive and fire too often, causing them to be often ignored by resolvers.

3.3 Maya

Maya is a new visualization tool designed to address the issues described in the previous section. Maya displays the dependencies among components as a directed graph. Each component—web page, service, or database—is represented as a black (healthy) or red (unhealthy) dot. The operators thus clearly see which components that are reporting alarms may simply be suffering from a cascaded failure, and defer paging those resolvers.

The other goal of Maya is to let the users easily "zoom in" to explore the important metrics of each component. However, each component is associated with thousands of hardware and software metrics, most of which are not directly useful when troubleshooting that component. To find the useful metrics, each component in Maya is associated with a dashboard that works like a Wiki: any resolver can edit the graphs that are displayed on the dashboard. This architecture makes it very easy for people to collaborate on identifying the useful metrics, as multiple resolvers at different computers can edit the dashboard simultaneously based on their individual knowledge. Resolvers can also specify which of the metrics defines the health of the component and how it maps to the black and red colors. All the important metrics for a given component can now be found on a single page, with the notion of "importance" defined by the collective actions of the resolvers who annotate that component.

3.4 Experience With Early Deployment

Maya is now in beta and has been available for anyone in Amazon.com to use since November 2005. It has turned out to be useful in some situations, but because its user community is still small, it is premature to try to quantify its potential usefulness in day-to-day operations.

However, interviews with the users of Maya identified the following issues:

1. Most of the users think that the UI is still overwhelming and offers too much information. This is largely because the tool displays more than 100 components; we need to experiment with showing only the failing components or only the important ones.

2. Since Amazon.com uses multiple software frameworks, Maya doesn't yet properly cover all services, all hardware, and all dependencies. Further, not all dependencies can be detected automatically so we need to add the ability to manually add a dependency.

3. Even though the dependencies are clearly visible on the graph, a primary of a tangential service that is not at fault may be paged. Maya needs to be integrated with the *automatic alarming system* to suppress alarms in services that depend on a different failing service.

All these features will be incorporated to Maya in the near future.

4 Diagnosing and Resolving Sev2 Problems

Sev2 problems only affect a single system component and the operators are not involved; only the corresponding primary resolver is paged and he needs to respond in 15 minutes. Many sev2 problems are caused by bugs in the source code as new features are added to the web site. Finding and fixing these bugs is usually simple, but in some cases a non-trivial reengineering of code is required, and in the meantime a workaround must be found. Possibilities include restarting the affected software component, rebooting the machine, or deleting temporary files. Since this doesn't actually fix the bug, the problem usually recurs later. For example, in the monitoring team, the majority of sev2 problems are "repeat offenders" resulting from a bug that may not be fixed for a few weeks, and in the meantime, they are handled as described above.

These repeating problems are an important subcategory of sev2s since they comprise the majority of problems resolvers have to deal with. To simplify handling of such repeating failures, the resolvers create "notes for the primary" that describe how to detect, diagnose, and fix such problems. However, sev2s are still hard to manage for the following three reasons:

1. The high rate of code churn creates a high rate of new sev2 problems appearing, so any documentation aimed at solving any particular sev2 problem is incomplete or short-lived.

2. Although the primary informs the rest of the team during meetings or via email about the types of problems he had to deal with and how he resolved them, this information is not systematically stored or managed, so a primary often needs to search through old emails or documentation or ask more experienced colleagues for help (sometimes waking them in the middle of the night).

3. Resolvers often move between teams and need to be trained to handle operations for that team. They usually *shadow* the primary for a few weeks after which they would join the on-call rotation as regular primary resolvers. However, this time is not enough to learn everything; again, new resolvers often find themselves searching documentation and asking colleagues much more often than the other resolvers.

4.1 Recommendation Service

Many Web sites already employ clickstream tracking and analysis as the basis of recommending to the user some new product for purchase. The insight behind clickstream analysis is that the user's actions provide implicit information about their interests.

We propose to apply this approach to sev2 troubleshooting. When an experienced troubleshooter is resolving a sev2, it is likely that his troubleshooting actions—which metrics he looks at, which commands he uses, etc.—represent implicit knowledge about resolving that problem. We propose to automatically populate database of past sev2 problems and their corresponding "solutions" by capturing the actions of the resolvers as they troubleshoot and fix the problems. The sequence of actions performed by a resolver describes a possible solution to that problem. When a new sev2 problem appears, we automatically find "similar" problems in the past and suggest their solutions to the resolver.

To implement such a system, for each problem we need to know: 1) its type, 2) who worked on this problem and when, and 3) actions performed by the resolvers.

4.1.1 Types of Problems

All of the repeating sev2 problems are detected through automatic alarming system. The mapping between types of alarms and types of problems is many-to-many because the same alarm can be caused by different problems and vice versa. However, the type of alarm is a useful indicator of the type of problem and we use it to classify problems into categories.

4.1.2 Failure Database

To determine which resolvers worked on each sev2 problem and when, we use the trouble-ticket database that contains the following information about each problem: start and end times, changes of severity, and *worklog* where resolvers working on this problem post their progress reports. We consider the list of resolvers who contributed to the worklog as the resolvers who worked on the problem. It's impossible to precisely determine when each of them worked on the problem. However, after discussing this issue with a few resolvers, we came up with the following heuristic: each involved resolver usually works on the problem the first thirty minutes after its initiation and then thirty minutes before posting each comment to the worklog.

4.1.3 Operator Monitor

To monitor the actions of resolvers during sev2 problems, we need to monitor two types of tools: web-based and command-line tools. The web-based tools are used for analyzing graphs of various metrics, alarms, source code, and documentation. They are all hosted on internal web servers and the HTTP access logs contain time, login name and URL for each access to these tools. Command-line actions such as restarting processes and applications, rebooting machines, or searching logs are usually logged in the *sudo* logs; all of them could be logged by using *history*. Given the resolvers and time intervals when they worked on a certain problem, we can extract the performed actions by parsing the logs of all internal tools.

4.2 Evaluating a Prototype

We have implemented a prototype of this tool and evaluated it on the nine most frequent types of sev2 problems in the Monitoring team which represent majority of all their sev2s. The type of each problem was determined by the alarm that generated it; examples of these alarms are the following: "size of the `/var` directory above limit," "CPU utilization above limit," "host unreachable," or "number of errors in a log file above limit". We analyzed all instances of these alarms during a period of two months. For each problem we determined who worked on it and when and extracted all actions performed by each involved resolver. Unfortunately, since we only had access to logs for various metric-monitoring tools, our system can suggest only which metrics and alarms are useful for each type of problem.

For each of the nine types of alarms, we generated a list of metrics that were accessed most often. We presented these ranked lists to two resolvers in the Monitoring team and asked them to mark the metrics that they find useful for troubleshooting each type of problem and also whether there are any metrics missing. The two resolvers received lists of metrics based on two months and three weeks worth of data, respectively. In both cases, we missed almost no important metrics (recall close to 100%). Precision, the fraction of reported metrics that were indeed useful (in the opinion of the experienced resolvers), was approximately 75% and 60% for two months and three weeks of data, respectively. Although the precision is not perfect, most of the useful metrics were ranked close to the top of the lists, suggesting that some postprocessing could further improve the precision. The higher precision for the two-month dataset confirms the intuition that the precision is higher for problems that repeat more often. To put these numbers into some context, Amazon.com collects a few million metrics used for long-term analysis and correlation, whereas the number of metrics actually useful for immediate troubleshooting is usually around ten.

5 Ongoing Work

As mentioned above, we are continuously improving Maya to make it more useful for operators. An important aspect of deploying this new tool lies in convincing operators and resolvers that it's useful and that they should use it. To achieve this, Maya needs to provide the features that resolvers care about most; we're also planning to organize training sessions.

To improve the usefulness of the recommendation system for operators, it should be extended in a few ways. First, we should instrument all operator tools so we can monitor all their actions; this is feasible since most of the tools are built in-house. Second, actions depend on previous actions and their results; we should incorporate this into our suggestions, however, monitoring the *results* of actions is difficult. Finally, the operators and resolvers themselves could improve the precision of such a tool by inspecting the actions they performed and marking the important ones.

One way to evaluate the usefulness of both tools is to compare the efficiency of two groups of operators and resolvers: those who use these tools versus those who don't use them. Since not everybody will choose to use these tools, we should have enough samples for such evaluation. For a more precise evaluation of the recommendation system, we're planning to compare the suggested actions with the actual actions of the resolvers.

6 Related Work

Internet sites detect and localize failures using various commercial monitoring tools such as IBM Tivoli or HP OpenView. These tools, however, don't provide the functionality needed to address issues of sev1 and sev2 problems and usually don't scale to the thousands of machines used at Amazon.com.

The most extensive study of operators and system administrators known to us is [1]. The authors surveyed one hundred operators and collected and analyzed approximately two hundred hours of videotaped administrators in action. They found that ”the available sysadmin tools do a relatively poor job of supporting sysadmins in several important areas”. In particular, they found the following problems: lack of support for planning, rehearsal, and collaborative work, low situation awareness of administrators working with complex systems, and low support for multitasking of operators. The most significant difference between this study and our work is the environment in which the operators work; the operators described in [1] work in large corporate data centers, usually administering third-party software that doesn't change very often. However, as described in our work, the environment in Amazon.com is quite different: hundreds of software developers, working also as operators, administer rapidly changing software behind Amazon.com.

7 Conclusion

In this paper we presented two promising tools for operators. Maya, a tool for visualizing the health of system components and dependencies between them that allows operators to collaborate on finding the most important metrics for each system component. The recommendation system for resolvers automatically monitors the actions of resolvers and later suggests actions that might help in diagnosing and resolving the current problem.

Acknowledgments

We would like to thank George Borle, Pieter De Temmerman, Monty Vanderbilt, Chris Whitaker, John Zook of the Monitoring team as well as others who contributed to this work.

References

[1] R. Barrett, E. Kandogan, P. P. Maglio, E. M. Haber, L. A. Takayama, and M. Prabaker. Field studies of computer system administrators: analysis of system management tools and practices. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer Supported Cooperative Work*, pages 388–395, New York, NY, USA, 2004. ACM Press.