ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 7, Issue 3, pp: (21-24), Month: July - September 2019, Available at: www.researchpublish.com

The Evolution of Application Performance Management

TANMAY KELKAR

Abstract: **As the complexity of enterprise systems increases, the need for monitoring and analyzing such systems also grows. A number of companies have built sophisticated monitoring tools that go far beyond simple resource utilization reports. For example, based on instrumentation and specialized APIs, it is now possible to monitor single method invocations and trace individual transactions across geographically distributed systems. This high-level of detail enables more precise forms of analysis and prediction but comes at the price of high data rates (i.e., big data). To maximize the benefit of data monitoring, the data has to be stored for an extended period of time for ulterior analysis. This new wave of big data analytics imposes new challenges especially for the application performance monitoring systems. The monitoring data has to be stored in a system that can sustain the high data rates and at the same time enable an up-to-date view of the underlying infrastructure. With the advent of modern key-value stores, a variety of data storage systems have emerged that are built with a focus on scalability and high data rates as predominant in this monitoring use case.**

Keywords: **application performance management (APM).**

1. INTRODUCTION

The performance of modern software applications has become a critical non-functional requirement. Unfortunately, due to the increased complexity of such applications and the increased number of users, maintaining an adequate level of performance becomes challenging. Performance regressions occur due to software updates that degrade the performance of an application. Regressions have negative consequences such as increased costs of software maintenance and user dissatisfaction. These regressions can even lead to substantial financial losses. Amazon shows that a delay of one second in page load time can decrease Amazon's sales by as much as $1.6 billion yearly [7]. As a result, a large amount of research has focused on studying the causes of performance regressions in software applications and how to efficiently detect such regressions primarily by mining various types of operational data (such as performance counters and logs). In practice, performance regression testing is performed on a software application before its release to detect performance regressions. Performance regression testing is the process of applying a workload on two versions of a software application in order to detect whether the code changes have introduced regressions. Hence, performance regression testing detects regressions when processing a stable workload due to code changes.

A plethora of research that analyzes performance data has been done to detect the performance regressions effectively. In particular, prior research in mining software repositories has shown the effectiveness of applying mining approaches to help developers identify performance regressions in large scale systems. However, most of such research is not easily accessible to practitioners. On the other hand, Application Performance Management (APM) tools are commonly used in practice. By integrating mining approaches on performance data into off-the-shelf performance monitoring tools, APM tools are often used to detect anomalies in the performance, instead of identifying performance regressions. Because of the availability of mining approaches that are already integrated into APM tools and the importance of identifying performance regressions, practitioners often tend to leverage APM tools to identify performance regressions. However, APM tools are not originally designed for that task and there exists no knowledge about their effectiveness for such a task.

2. BACKGROUND WORK

Currently, the Web is used quite differently than the purpose for which it was originally conceived—sharing scientific information among a few scientists. The scope and complexity of current Web applications vary widely: from small scale, short-lived services to large-scale enterprise applications distributed across the Internet and corporate intranets and extranets. Web-based applications can be grouped into the seven categories [1], although a given application may belong to more than one category. As Web applications have evolved, the demands placed on Web-based systems and the complexity of designing, developing, maintaining, and managing these systems have also increased significantly. For example, Web sites such as for the 2000 Sydney Olympics, 1998 Nagano Olympics, and Wimbledon received hundreds of thousands of hits per minute [3]. They provided vast, dynamic information [2,4] in multiple media formats (graphics, images, and video). Web site design for these and many other applications demands balance among information content, aesthetics, and performance. Although numerous Web-based systems are in use now, the manner in which they're developed, deployed, and managed raises serious concerns (see the sidebar, "Problems of Web-Based Systems Development: A Diagnosis"). Web developers often use ad hoc, hacker-type approaches, which lack rigor, systematic techniques, sound methodologies, and quality assurance. The current problems surrounding Web-based system development partially result from the continuing advances in Internet and Web technologies, the increase in commercial Web applications, the frantic rush to be on the Web, and the need to quickly migrate legacy systems to Web environments. Poorly developed Web-based applications that continue to expand have a high probability of low performance and/or failure. Recently, large Web based systems have had an increasing number of failures (many of them weren't publicly acknowledged or documented). In certain classes of applications such as supply-chain management, tendering and procurement, financial services, and emerging digital hubs or marketplaces, a system failure can propagate broad-based problems across many functions, causing a major Web disaster. The cost of bad design, shabby development, poor performance, and/or lack of content management for Web-based applications have many serious consequences. A recent survey on Web-based project development by the Cutter Consortium (reported in its Research Briefs, 7 Nov. 2000) highlighted problems plaguing large Web-based projects:

- Delivered systems didn't meet business needs 84 percent of the time.
- Schedule delays plagued the projects 79 percent of the time.
- Projects exceeded the budget 63 percent of the time.
- Delivered systems didn't have the required functionality 53 percent of the time.
- Deliverables were of poor-quality 52 percent of time.

As a result, developers, users, and other stakeholders have become increasingly concerned about the manner in which complex Web-based systems are created as well as the level of system performance, quality, and integrity.

3. WHAT IS AN APM TOOL?

An APM tool is usually used to monitor the performance and the availability of a monitored web-based software application. An APM tool collects several performance metrics (such as response time) from the monitored application and mines these metrics to measure the health of the application (e.g., identify potential performance problems using mining approaches). Most of the metrics that are mined by the APM tools are used in performance regression detection research as well. Hence, APM tools might be effective in detecting performance regressions using these metrics. APM tools follow a typical workflow. After installation, 2 the APM tool discovers the different software artifacts of the application that is being monitored. This discovery step allows the APM tool to visualize the application deployment structure and to collect fine-grained metrics about the transactions that are processed by the monitored application. A transaction usually means a web request originating either from a client browser or from another application using a web service request. Examples of fine-grained metrics include: the amount of time spent in each application component for each request and the total number of transactions processed by each application component. During the analysis phase, the APM tool collects the performance metrics periodically. The APM tool mines a historical repository of the collected metrics to determine whether a transaction is abnormal (e.g., slower than usual). APM then sends alerts to the

practitioners about transactions having slow performance or issues related to the computational resources used by the monitored application. These alerts can be categorized into three main categories:

- Transaction-related alerts: Information about single transactions such as the response time and the stack trace, in addition to aggregated information such as the total number of transactions and the average response time.
- Memory-related alerts: Information about memory usage and possible excessive memory usage.
- Database-related alerts: Information about the executed database queries such as query details, number of executed queries for each transaction and the time consumed to run each query.

We believe that these types of information can be used by practitioners not only to detect performance anomalies in production due to workload changes, but also to identify performance regressions (where the code is changing but the test workload is stable) across versions of a software application.

4. PERFORMANCE ANALYSIS APPROACHES

The studied tools use two general mining approaches to detect performance anomalies: 1) baseline-based and 2) threshold based. Some APM tools use only one approach (usually the threshold-based approach) such as New Relic [8,9] and Pinpoint. Other APM tools support both approaches, such as AppDynamics and Dynatrace.

1. Baseline-based Approach

In baseline-based approaches, APM tools mine historical performance data to establish a baseline. For example, an application is expected to have a higher load during the working hours and a lower load during weekends. Hence the APM tool needs to learn this behavior. In order to detect performance anomalies, APM tools detect metric values that deviate from this baseline. APM tools use different mining approaches to establish the baseline. For example, AppDynamics uses the average value of a metric observed during a specific time range to define the baseline. Dynatrace [5,6] uses other statistical techniques such as the 90th percentile and binomial distribution to calculate the baseline. APM tools use these baselines as indicators to notify practitioners when their application performance deviates from the baselines. The specific details of these statistical techniques are not available in the documentation of the commercial tools. For example, we are unable to identify the parameters of the binomial distribution used by Dynatrace.

2. Threshold-based Approach

A threshold is the value beyond which the performance of the monitored application is considered unacceptable. A threshold can be calculated using simple statistical methods or it can be configured either by the software practitioners or by the APM tool itself. APM tools support the following types of thresholds:

• Percentage deviation threshold: When a metric value exceeds a specific percentage above the metric's average.

• Standard deviation threshold: When a metric value exceeds multiples of standard deviations above the metric's average.

• Fixed threshold: When a metric value exceeds a predefined fixed value. Usually, a threshold is set for the transaction response time metric because it is the APM tool's primary concern. However, some APM tools provide means to define thresholds for other metrics, such as the failure rate and throughput (in AppDynamics and Dynatrace). Pinpoint uses fixed thresholds to decide whether the transaction is slow or not based on its response time. Specifically, Pinpoint has four thresholds for the response time of the transaction: less than one second, less than three seconds, less than five seconds, and more than five seconds.

5. CONCLUSIONS AND FUTURE WORKS

Application Performance Monitoring consists of software that is designed to help application troubleshooters ensure that the multitude of applications run as per performance standards and provide a user experience that is as per the user's requirements. APM tools provide administrators with the requisite information such as a database query or a specific method or file that they need to swiftly discover, isolate and solve problems that can adversely hamper an application's performance. APM tools monitor an application's performance over a fixed course of time and help cloud troubleshooters understand the effect that different dependencies such as the systems that applications rely on to function properly, have on an application's performance. Cloud troubleshooters can use the performance metrics, which an APM tool can measure

from a particular application or multiple applications on the same network, to identify the source of the performance issue. The data APM tools collect includes client CPU utilization, memory demands, data throughput and bandwidth consumption. Most APM tools include the ability to gather application performance statistics from multiple sources and correlate them through log files, hardware statistics and network throughput usage reports. The information is then displayed in a dashboard along with graphs that makes it easier for cloud troubleshooters to read data logs, which removes the need for them to manually delve through all the applications source files and identify the problem.

REFERENCES

[1] A. Chandra, W. Gong, and P. Shenoy, "Dynamic Resource Allocation for Shared Data Centers using Online Measurements," IWQoS, 2003.

[2] A. Chandra and P. Shenoy, "Effectiveness of dynamic Resource allocation for handling Internet Flash Crowds," Technical Report TR03-37, Department of Computer Science, University of Massachusetts Amherst, November 2003.

[3] J. Shahabuddin, A. Chrungoo, V. Gupta, S. Juneja, S. Kapoor, and A. Kumar, "Stream-Packing: Resource Allocation in {Web} Server Farms with a QoS Guarantee," Lecture Notes in Computer Science.

[4] T. Kimbrel, M. Steinder, M. Sviridenko, and A. Tantawi "Dynamic application placement under service and memory constraints," in Proceedings of WEA 2005, pp. 391-402.

[5] Dynatrace instrumentation. https://community. dynatrace.com/community/display/DOCDT61/ Instrumentation. Last accessed Oct 16 2015.

[6] Dynatrace PurePath. https://community.dynatrace.com/community/display/DOCDT60/PurePath+ Explained. Last accessed Sept 24 2015.

[7] How one second could cost amazon $1.6 billion in sales. http://www.fastcompany.com/1825005/ how-one-second-could-cost-amazon-16-billion-sales. Last accessed Oct 4 2015.

[8] New Relic. http://newrelic.com/. Last accessed Sept 9 2015.

[9] New Relic custom instrumentation. https://docs.newrelic.com/docs/agents/java-agent/ custom-instrumentation/java-custom-instrumentation. Last accessed Sept 24 2015.