Application Performance Management: State of the Art and Challenges for the Future

Christoph Heger,[1] André van Hoorn,[2] Mario Mann,[1] Dušan Okanović[2]
[1]NovaTec Consulting GmbH, Competence Area APM, Leinfelden-Echterdingen, Germany
[2]University of Stuttgart, Institute of Software Technology, Stuttgart, Germany

ABSTRACT

The performance of application systems has a direct impact on business metrics. For example, companies lose customers and revenue in case of poor performance such as high response times. Application performance management (APM) aims to provide the required processes and tools to have a continuous and up-to-date picture of relevant performance measures during operations, as well as to support the detection and resolution of performance-related incidents.

In this tutorial paper, we provide an overview of the state of the art in APM in industrial practice and academic research, highlight current challenges, and outline future research directions.

1. INTRODUCTION

Business success is directly influenced by the performance of the enterprise application systems that support it. Any performance issue that may arise during the production use of such applications may bring losses in revenue, and even cause customers to turn away. Examples of these losses and their impact are well documented. Google loses 20% traffic if their web sites respond 500 ms slower [9]. Amazon loses 1% of revenue for every 100 ms in latency [8]. Mozilla's study showed that if the page is not loaded within one to five seconds, users will leave the web site [4].

Application performance management (APM), as a core IT operations discipline, aims to achieve an adequate level of performance during operations. To achieve this, APM comprises methods, techniques, and tools for *i)* continuously monitoring the state of an applications system and its usage, as well as for *ii)* detecting, diagnosing, and resolving performance-related problems using the monitored data.

In this paper, we provide a state-of-the-art overview of the common APM activities (Section 2) and tools (Section 3), and highlight selected challenges and future directions (Section 4).

ICPE'17, April 22-26, 2017, L'Aquila, Italy
© 2017 ACM. ISBN 978-1-4503-4404-3/17/04…$15.00
DOI: http://dx.doi.org/10.1145/3030207.3053674

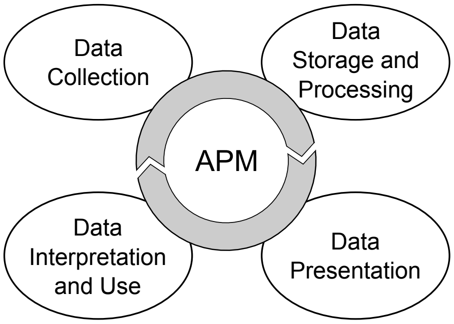


Figure 1: Continuous APM activities

2. APM ACTIVITIES

Regardless of the actual technical realization, APM involves the following four—concurrently conducted and interrelated—activities (Figure 1):

1. *Data collection.* Performance measures are collected from the different system tiers, layers, and locations by a combination of complementary techniques and technologies.
2. *Data storage and processing.* The collected data is combined into higher-level data structures, such as time series or execution traces.
3. *Data presentation.* Data is made available for visual inspection on different levels of abstraction and detail.
4. *Data interpretation and use.* The data is used to manually or automatically reason about and act upon the current state.

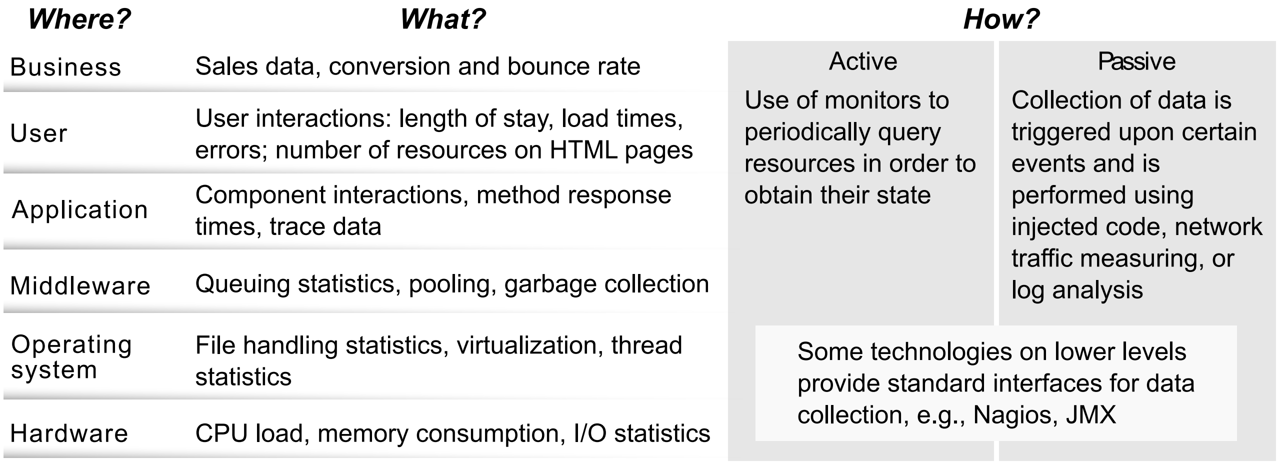
The remainder of this section details these activities.

2.1 Data Collection

Various types of performance-relevant measures can and need to be collected from a system and its clients. A core decision of APM regarding data collection is *where*, *what*, and *how* to collect, as detailed in the remainder of this section and depicted in Figure 2.

2.1.1 Where to Collect Data?

Modern application systems are multi-tiered, highly distributed, multi-layered, and accessed via different types of clients and devices (e.g., third-party systems and humans using desktop or mobile devices).



Where?	What?
Business	Sales data, conversion and bounce rate
User	User interactions: length of stay, load times, errors; number of resources on HTML pages
Application	Component interactions, method response times, trace data
Middleware	Queuing statistics, pooling, garbage collection
Operating system	File handling statistics, virtualization, thread statistics
Hardware	CPU load, memory consumption, I/O statistics

How?

Active	Passive
Use of monitors to periodically query resources in order to obtain their state	Collection of data is triggered upon certain events and is performed using injected code, network traffic measuring, or log analysis

Some technologies on lower levels provide standard interfaces for data collection, e.g., Nagios, JMX

Figure 2: Example measures and techniques for data collection on different system levels

Most application systems are implemented in a way that, in addition to the application logic executed at the provider's site (referred to as the back-end), parts of the application are executed at the client's site. The client site usually constitutes a system tier accessing the back-end via (graphical) interfaces such as fat-clients, thin clients realized in web browsers, or native apps on mobile devices. Communication between clients and the back-end may be conducted by networked, wireless, and/or cellular connections. The back-end involves multiple tiers, for instance concerned with functionality regarding presentation, business, and persistence. The different application tiers and services are usually deployed to a heterogeneous infrastructure of distributed physical and virtual computing resources, each involving stacked layers from hardware to the application.

To provide an end-to-end view on application performance, APM requires the collection of relevant performance measures from all of the mentioned locations.

2.1.2 What Data to Collect?

The types of measures that can and should be collected depends on the previously mentioned locations, but also on the architectural style used by the application system.

As mentioned previously, the primary goal of any application system is to provide support for business processes. Hence, following a top-down perspective of APM, measures related to business and end-user experience are of primary interest. On a business level, these measures include data about completed and uncompleted conversions (e.g., statistics about client sessions with and without buy transactions). Measures about the end-user experience obtained on the client site include end-to-end response times of interactions, (page) load times, errors, and data about the UI usage. On the application level, measures about the application-internal behavior can be collected, including executions of methods, occurrences of exceptions, calls to remote services or databases, etc. On the system level, i.e., middleware, operating system, and hardware, measures about the state of hardware and software resources are collected in particular. Additional example measures for the different levels are included in Figure 2.

2.1.3 How to Collect Data?

There are many approaches for collecting the data, which can be categorized into two main groups: active and passive.

Active data collection is performed by periodic sampling of system services or resources. This includes the emulation of customers using synthetic requests. *Passive data collection*, on the other hand, is performed by collecting the data when certain events, such as method executions, occur. The data can be gathered by, e.g., injecting the measurement logic into the application source or byte code, stack trace sampling, capturing application logs, or mirroring network traffic. Additionally, for some system levels, there are standard interfaces that allow accessing and collecting data.

2.2 Data Storage and Processing

In order to have a central view on the collected data, it is usually transferred by the agents to a data storage for further processing and analysis. Proprietary or standard technologies (e.g., database management systems) can be and are being used. APM usually results in very large data sets that need to be handled efficiently [14].

Two data representations are commonly used: time series and execution traces. While time series represent summary statistics (e.g., counts, percentile, etc.) over time, execution traces [3] provide a detailed representation of the application-internal control flow that results from individual system requests. From this data, architectural information, including logical and physical deployments and interactions (topology), can be extracted.

2.3 Data Presentation

Due to the high quantity, APM information needs to be presented in a meaningful and comprehensible way using different interrelated and navigable views. These views can be categorized using two dimensions: the scope (business vs. technology) and the level of abstraction. Views can contain detailed business information such as the status of user devices, geolocations, as well as the health of the available services. On the other hand, data can be presented in the form of traces, time series, page flows, underlying topologies, server health, etc. These views vary from more abstract to more detailed, depending on what is required to answer the respective concern.

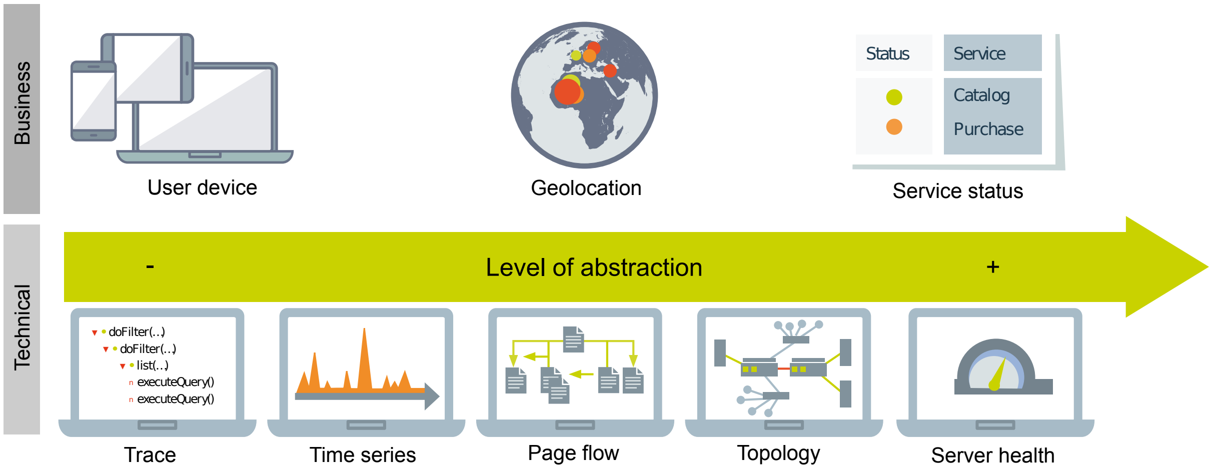


Figure 3: Example views on APM data with different scope and level of abstraction

An example use of the different views is as follows. A service status view shows that all services are healthy. As soon as a service is indicated not to be healthy any more, the incident needs to be analyzed. This is achieved by navigating to the other, more detailed views.

2.4 Data Interpretation and Use

The available data—comprising time series, execution traces, and topology information—can be interpreted and used with different goals, e.g.:

- *Problem detection and alerting.* Statistical techniques can be applied to the data to detect anomalies, which can indicate problems. Particularly time series data is used to detect violations of thresholds, which are either specified manually (e.g., based on service level agreement—SLAs) or learned from the historic data (baselines). In case an anomaly is detected, alerts can be sent out to system operators. To retain trust in the automatic detection and alerting, a high classification quality (e.g., in terms of precision, recall, and related measures) is desired.
- *Problem diagnosis and root cause isolation.* In case a problem occurred, the goal is to isolate its root cause during problem diagnosis—again manually or automatically. Manual analysis is usually conducted by employing the previously mentioned data representations, e.g., by navigating from status lights, via application topologies, to component drill downs, execution traces, and time series. There are also approaches to automatically detect root causes of typical performance problems, also known as performance antipatterns [15]. These approaches usually use trace data [6], but may also add other information, e.g., configuration data [12].
- *System refactoring and adaptation.* Other approaches use the data for automatic reaction in order to minimize the impact of performance issues. In cloud environments, auto-scaling approaches [10] can compare the data with SLAs [7], to determine further actions.

The aforementioned approaches can be conducted in a reactive or proactive manner, i.e., after a problem occurred or to predict that a problem will occur [13].

3. TOOLING SUPPORT

Tooling support is at the core of realizing the previously described APM activities. An ever increasing number of APM-supporting tools exist, ranging from fully-fledged APM tool suites covering the whole process, to specialized tools focusing on specific smaller problems, e.g., collection of certain measures, tailored database management systems, analytics, and visualization.

Most APM tools provide some basic functions, such as resource monitoring, architecture discovery, component deep-dive, end-user experience monitoring. In addition, they can provide data visualization and some form of analytics, e.g., baseline calculation and anomaly detection. Modern APM tools usually support monitoring in complex environments, that can consist of different platforms, different programming languages, etc. Tools usually consist of two components: *i)* a monitoring agent that collects the data, and *ii)* a storage and analysis component. Depending on the requirements, the analysis component can be further divided. Two usual modes of installation are available [2], *on-premise* and *SaaS-based* (software as a service).

The most mature and feature-rich APM tools are commercial products such as AppDynamics, CA APM, Dynatrace, and New Relic, regularly reviewed by Gartner [5]. As an alternative to commercial solutions, open-source tools are often used to implement the technical APM infrastructure. Mature open-source tools for monitoring on system-level have been around for many years (e.g., Nagios[1]). Open-source application-level monitoring tools (e.g., Kieker [17] and inspectIT[2]) are available. Other tools are available for collecting distributed execution traces (e.g., Zipkin[3]) and are being used together with emerging technologies for data storage and analytics (e.g., logging infrastructure, NoSQL databases, big data).

4. CHALLENGES AND DIRECTIONS

In this section, we share our view on what selected challenges and promising future research directions are.

[1] http://www.nagios.org/
[2] http://www.inspectit.eu/
[3] http://zipkin.io/

Automation of Supporting Activities. APM practice requires expertise and effort. For instance, expertise is required for setting up and maintaining APM configurations (e.g., deciding which parts of the software to instrument), as well as for the analysis and visualization of the data. Even for experts, manual tasks can be error-prone, costly, and frustrating because various tasks and problems are recurring. Automation of these tasks could be performed by formalizing the expert knowledge, and using it to solve these tasks.

Problem Detection, Diagnosis, and Prediction. Regarding the root cause analysis of performance problems, today's tools give little or no support. They are usually limited to alerting and visualization, but the diagnosis and root-cause analysis of performance issues, still has to be performed manually. Systematization of expert knowledge and machine learning approaches could provide key support here.

Interoperability. Currently, all of the tools store the data in their own format. As a consequence, it is a common practice that the same analysis approach has to be re-implemented for different APM tools. Some tools allow export of data in some machine readable format, such as XML, and this data can than be parsed with more or less effort [11]. However there are ongoing works on developing APIs and formats for APM tool interoperability [1, 11].

Development Paradigms and Architectural Styles. Modern development paradigms such as *DevOps* aim for frequent releases— posing additional APM challenges, e.g. how to calculate new baselines when the time span between new releases is extremely short. On the other hand, in these kind of environments, APM data from production can be made available to developers, e.g., for use in IDEs and extracted performance models. Emerging architectural styles and new cloud-based delivery models, e.g., microservice and serverless architectures, further extend the task of cross-platform monitoring and require the tools to cope with new measures and uncertainty in results.

5. CONCLUSIONS

APM allows to provide deep insights into the run-time behavior of application systems, supporting the detection, diagnosis, and resolution of incidents. In this paper, we covered the core APM activities and tooling support, and identified current limitations.

Modern development paradigms and architectural styles provide challenges to which APM practice and research will have to provide solutions. APM is not a purely technical topic anymore, as there is also a need for support of business activities and vice versa. We see promising future research directions in automation of supporting activities and analysis of data. In our current research, we try to tackle selected challenges by including expert knowledge and analyzing performance concerns by a declarative approach [6, 18]. Technology transfer of new APM approaches developed in research would benefit from the availability of real-world APM data for the evaluation of the approaches.

6. ACKNOWLEDGMENTS

This work is being supported by the German Federal Ministry of Education and Research (grant no. 01IS15004, *diagnoseIT*). The authors and this paper have benefited from frequent discussions with Stefan Siegl—including joint work on our APM poster [16] that served as a basis for this paper.

7. REFERENCES

[1] OpenTracing: A vendor-neutral open standard for distributed tracing. http://opentracing.io/, 2016.

[2] T. M. Ahmed, C.-P. Bezemer, T.-H. Chen, A. E. Hassan, and W. Shang. Studying the effectiveness of application performance management (APM) tools for detecting performance regressions for web applications: An experience report. In *Proc. 13th Int. Conf. on Mining Software Repositories (MSR '16)*, pages 1–12, 2016.

[3] G. Ammons, T. Ball, and J. R. Larus. Exploiting hardware performance counters with flow and context sensitive profiling. In *Proc. ACM SIGPLAN '97 Conf. on Programming Language Design and Implementation (PLDI '97)*, pages 85–96, 1997.

[4] B. Cutler. Firefox and page load speed (part I). https://blog.mozilla.org/metrics/2010/03/31/firefox-page-load-speed-part-i/, 2010.

[5] C. Haight and F. D. Silva. Gartner's magic quadrant for application performance monitoring suites, 2016.

[6] C. Heger, A. van Hoorn, D. Okanović, S. Siegl, and A. Wert. Expert-guided automatic diagnosis of performance problems in enterprise applications. In *Proc. 12th Europ. Dependable Computing Conf. (EDCC '16)*. IEEE, 2016.

[7] Y. Kouki and T. Ledoux. CSLA: A Language for improving Cloud SLA Management. In *Proc. Int. Conf. on Cloud Computing and Services Science (CLOSER 2012)*, pages 586–591, 2012.

[8] J. Liddle. Amazon Found Every 100ms of Latency Cost Them 1% in sales. http://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/, 2008.

[9] G. Linden. Marissa Mayer at Web 2.0. http://glinden.blogspot.de/2006/11/marissa-mayer-at-web-20.html, 2006.

[10] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, 12(4):559–592, 2014.

[11] D. Okanovic, A. van Hoorn, C. Heger, A. Wert, and S. Siegl. Towards performance tooling interoperability: An open format for representing execution traces. In *Proc. 13th Europ. Workshop on Computer Performance Engineering EPEW '16*, pages 94–108, 2016.

[12] T. Parsons and J. Murphy. Detecting performance antipatterns in component based enterprise systems. *Journal of Object Technology*, 7(3):55–91, 2008.

[13] T. Pitakrat, D. Okanovic, A. van Hoorn, and L. Grunske. An architecture-aware approach to hierarchical online failure prediction. In *12th Int. ACM SIGSOFT Conf. on Quality of Soft. Architectures*, pages 60–69, 2016.

[14] T. Rabl, S. Gómez-Villamor, M. Sadoghi, V. Muntés-Mulero, H.-A. Jacobsen, and S. Mankovskii. Solving big data challenges for enterprise application performance management. *Proc. VLDB Endow.*, 5(12):1724–1735, 2012.

[15] C. U. Smith and L. G. Williams. Software performance antipatterns. In *Proc. 2nd Int. Workshop on Software and Performance (WOSP '00)*, pages 127–136, 2000.

[16] A. van Hoorn and S. Siegl. Application performance management (APM): Continuous monitoring of application performance (OBJEKTspektrum poster, in german). https://www.sigs-datacom.de/wissen/fachposter/.

[17] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In *Proc. 3rd ACM/SPEC Int. Conf. on Perf. Eng. (ICPE '12)*, pages 247–248, 2012.

[18] J. Walter, A. van Hoorn, H. Koziolek, D. Okanovic, and S. Kounev. Asking "what"?, automating the "how"?: The vision of declarative performance engineering. In *Proc. 7th ACM/SPEC on Int. Conf. on Perf. Eng.*, ICPE '16, pages 91–94, 2016.