Holistic performance analysis for large-scale distributed systems

Francisco Neves and José Pereira
francisco.t.neves@inesctec.pt,jop@di.uminho.pt
HASLab - INESC TEC and U. Minho
Braga, Portugal

1 CONTEXT

The complexity of systems hosted in cloud infrastructures raises challenges in managing resources to ensure the performance stated in Service Level Agreements. The heterogeneity and interdependencies of integrated off-of-the-shelf and custom software, along with virtualization layers and the unknown behavior of those systems, *i.e.*, how their components interact among themselves, make it difficult to pinpoint the root cause of bottlenecks and act accordingly.

Effective resource management is strongly related to the knowledge about the system [8]. Knowing how each system is composed and how its components interact not only eases the task of pinpointing bottlenecks but also allows a better prediction of the effects of resource management on the overall performance, that is, after changing resources allocated to components of the system [1].

This project aims at discovering behavior of complex, multi-tier, black-box large-scale distributed systems. This is suitable for wide range of scenarios: from performance analysis over in-house systems, potentially built along with other opaque sub-systems, to effective resource management in cloud infrastructures that host these systems. Such systems raise challenges not only in pinpointing the root cause of a performance bottleneck but also in predicting the consequences in holistic performance, good or bad, of allocating and deallocating resources assigned to each component of each system. In fact, as the knowledge of a given system increases, better decisions can be made. To this end, three main challenges are addressed:

- Discover the architecture and behavior of software components. Determining how black-box composite systems operate allows to build a representation of the whole system. This should cause a minimal overhead and be applicable to heterogenous components;
- Simulate systems behavior under several scenarios. This should somehow rely on a correlation between performance key indicators and hardware configuration and be easily adapted for various workloads;
- Leverage knowledge about black-box systems and its simulated behavior and apply effective resource management, without harming the overall system performance.

In this project we foresee two main scientific contributions: A method to extract and model the behavior of a given composite system; and a method to quantify scalability of the whole system under a given scenario. These contributions will be embodied in prototype tools that are used for experimental evaluation.

2 STATE OF THE ART

The analysis of distributed systems has attracted a lot of attention and been addressed in various ways for different purposes. A common approach is to add instrumentation to systems to generate events for future logging and analysing. Namely, Google Dapper is a tracing tool used in Google services to record requests flow by annotating messages sent through standard communication protocols [15]. X-Ray applies instrumentation to Java-based data processing systems for monitoring and analysis of distributed database queries [5]. Other approaches are based on low-level traces which significantly increases the amount of collected data [14]. For instance, both X-Trace [4] and Pivot Tracing are suitable for Java-based systems and aim at identifying the root causes of software bugs and misconfiguration. The main difference between both is that Pivot Tracing is able to correlate distributed events across components without expensive operations such as aggregations [12]. It also inspired Facebook Canopy [9], an end-to-end performance tracing and analysis that enables the collection of performance metrics in order to allow a global analysis of performance from the back-end to the final user. What makes it more relevant when compared to Pivot Tracing is the ability to perform analysis over historical data in a near real-time fashion.

Generic monitoring tools make it possible to execute queries over performance metrics thus providing useful filtered and aggregated data. One of the well-known tools is Prometheus, a monitoring tool that collects performance metrics from each running agent deployed on each node and provides a powerful Query Language. Despite the flexibility of the query language and amount of metrics available for analysis, it is limited to performance indicators. Therefore, pinpointing the root cause of a performance bottleneck is still not trivial for engineers.

Current Monitoring-as-a-Service (MaaS) solutions enable monitoring without the need of manual and tedious configuration nor instrumentation [11]. The monitoring agents they provide are capable of discovering the architecture and interactions among processes and services within the system under analysis. The key to make this happen is to provide an agent that is ready to instrument specific points in running processes. When compared to approaches like Prometheus, these MaaS solutions do not provide a so powerful query language or even none at all, which limits the overall understanding of the architecture, interdependencies and performance metrics.

As knowledge about a given system grows, it can then be used to build or instantiate better models that describe its scalability. On one end this can be as simple as direct application of Amdahl's Law, to estimate scalability of a parallel program in multi-processor architectures [2], and Universal Scalability Law, which differs in representing the loss of scalability due to inter-node communication. Both require benchmarks to estimate their parameters, which is unpractical for large scale and dynamic systems [6]. One interesting feature of the Universal Scalability Law is the fact that it can be composed, so it could be used to model holistic performance of a given system by composing the scalability models of its sub-systems. Nonetheless, it only performs well over homogeneous hardware, which is not always the case in distributed systems. On the other end, there are techniques that ease the performance debugging and bottleneck identification by modeling the system as a queueing network and considering latency of inter-component communication. Still, it requires the service level time and arrival rate as parameters, which are not trivial to get [7]. Amdahl's Law and queueing theory were also combined to work around these parameters [3].

Finally, knowledge about the system can be used to act on itself. In autonomic systems, resources (or components) are managed by resource managers, containing the control loop Monitor-Analyse-Plan-Execute, based on knowledge about the system [10]. When multiple resource managers exist, each monitoring and executing actions based on collected information about its managed resource, the system may become dysfunctional [13]. Therefore, orchestration is needed in order to act according to the overview of the system. In black-box systems, this is even more challenging since no overview of the system is available.

3 APPROACH AND DISCUSSION

This project thus addresses three complementary aspects: Modeling, simulation, and scalability analysis of black-box large scale systems. Black-box approaches can be applied to several layers of a system, from its architecture to application-level knowledge. We start by defining black-box systems as systems whose architecture and interactions among its components, or subsystems, are completely unknown. In order to understand what is the proper function of a system and what and how many resources it needs to operate properly, the first stage of this work is to outline a procedure to discover the behavior of the system. The behavior of the system can be specified as the collected information about three main characteristics: architecture, that shows how system is organized; dependencies between system's components, which hint how changing a given component affects the others; and the system's status under several workloads.

The architectures we target go beyond simple three-tiered systems, tipically used in web services. Instead, several virtualisation layers with containers and virtual machines should be considered as they are introduced to keep systems isolated, specially in multi-tenant systems, where resources are shared. It is a concern of our approach to discover how processes are placed across nodes and virtualised environments and to do so we will first rely on existing libraries for communicating with hypervisors to firstly cover environments where virtual machines are chosen over containers. We achieve this with a single monitoring agent per physical and per virtualised node able to collect this information. The resulting model that comprises the architecture of the system and interactions among its components can be draw as a graph with several types of nodes. Concretelly, in the end nodes represent physical machines, virtualised environments, processes and services. The edges that connect all these nodes represent allocation, in case of virtual machines and physical nodes, and communication relationship, such as processes exchanging messages.

According to the previous *black-box* definition, all this information must be retrieved without requiring any application-level knowledge. To do so, low-level tools are leveraged to collect data for future correlations and thus allow to infer the behavior of the target system. However, choosing the right data and tools to gather relevant data to this end is not trivial, because despite of knowing that several specificities of the system will still be hidden, the procedure that collects and correlates data will determine the accuracy of the modeling process. Moreover, the way these low-level tools work can easily become intrusive to the point of being unpractical to use.

The accuracy of modeling black-box systems is related to the amount and detail of collected data. However, there is a trade-off between the amount of collected data and the overhead imposed by these tools. A large amount of data allows to detail the behavior of system but monitoring may become intrusive. In fact, this is the main challenge with monitoring black-box systems. In the end of this stage, we should be able to fit any distributed black-box system into a model that is representative for scalability analysis. As mentioned in

Section 2, there are two well-known scalability quantification models used in practice, however as their parameters are non trivial to estimate, the main challenge of this stage will fall into understanding what metrics could be useful to estimate them.

After analysing the system and building its model, the second stage is to use the model to simulate the behavior of the system under a given hardware configuration. By using this model in a simulation tool, it should output useful data to analyse if the system under simulation might be under contention or not. Additionally, it should be improved to detect which components are causing performance bottlenecks.

REFERENCES

[1] Marcos K Aguilera, Jeffrey C Mogul, Janet L Wiener, Patrick Reynolds, and Athicha Muthitacharoen. 2003. Performance debugging for distributed systems of black boxes. *ACM SIGOPS Operating Systems Review* 37, 5 (2003), 74–89.

[2] Gene M Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 483–485.

[3] Nuno A Carvalho and José Pereira. 2010. Measuring software systems scalability for proactive data center management. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 829–842.

[4] Rodrigo Fonseca, George Porter, Randy H Katz, Scott Shenker, and Ion Stoica. 2007. X-trace: A pervasive network tracing framework. In *Proceedings of the 4th USENIX conference on Networked systems design & implementation*. USENIX Association, 20–20.

[5] Pedro Guimarães and José Pereira. 2015. X-Ray: Monitoring and analysis of distributed database queries. In *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 80–93.

[6] Neil J Gunther. 2007. Scalability—A Quantitative Approach. *Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services* (2007), 41–69.

[7] Ahmed Harbaoui, Nabila Salmi, Bruno Dillenseger, and Jean-Marc Vincent. 2010. Introducing queuing network-based performance awareness in autonomic systems. In *Autonomic and Autonomous Systems (ICAS), 2010 Sixth International Conference on*. IEEE, 7–12.

[8] Olumuyiwa Ibidunmoye, Francisco Hernández-Rodriguez, and Erik Elmroth. 2015. Performance anomaly detection and bottleneck identification. *ACM Computing Surveys (CSUR)* 48, 1 (2015), 4.

[9] Jonathan Kaldor, Jonathan Mace, Michał Bejda, Edison Gao, Wiktor Kuropatwa, Joe O'Neill, Kian Win Ong, Bill Schaller, Pingjia Shan, Brendan Viscomi, et al. 2017. Canopy: An End-to-End Performance Tracing And Analysis System. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 34–50.

[10] Jeffrey O Kephart and David M Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (2003), 41–50.

[11] Dynatrace LLC. 2005. Dynatrace: Digital Performance and Application Performance Monitoring. (2005). https://www.dynatrace.com/

[12] Jonathan Mace, Ryan Roelke, and Rodrigo Fonseca. 2015. Pivot tracing: Dynamic causal monitoring for distributed systems. In *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, 378–393.

[13] Mohammad Reza Nami and Koen Bertels. 2007. A survey of autonomic computing systems. In *Autonomic and Autonomous Systems, 2007. ICAS07. Third International Conference on*. IEEE, 26–26.

[14] Bo Sang, Jianfeng Zhan, Gang Lu, Haining Wang, Dongyan Xu, Lei Wang, Zhihong Zhang, and Zhen Jia. 2012. Precise, scalable, and online request tracing for multitier services of black boxes. *IEEE Transactions on Parallel and Distributed Systems* 23, 6 (2012), 1159–1167.

[15] Benjamin H Sigelman, Luiz Andre Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. [n. d.]. *Dapper, a large-scale distributed systems tracing infrastructure.* Technical Report.