LOG 8371E
Software Quality Engineering

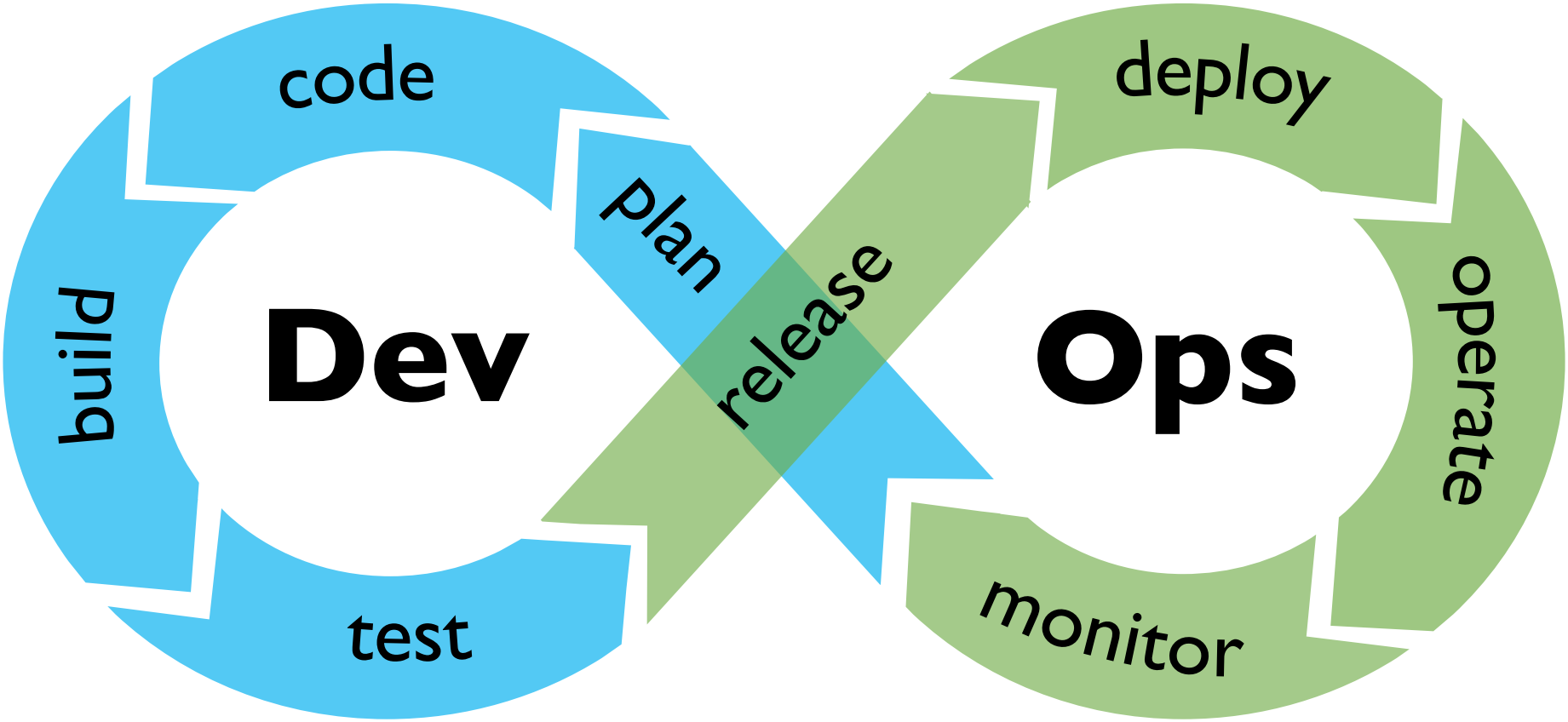
Lecture 11:

Autoscaling and Self-adaptive Systems

Armstrong Foundjem Ph.D. — Winter 2024

Suppose we have a targeted response time of 500ms, but our application responds on average in 1.4s.

What can we do?

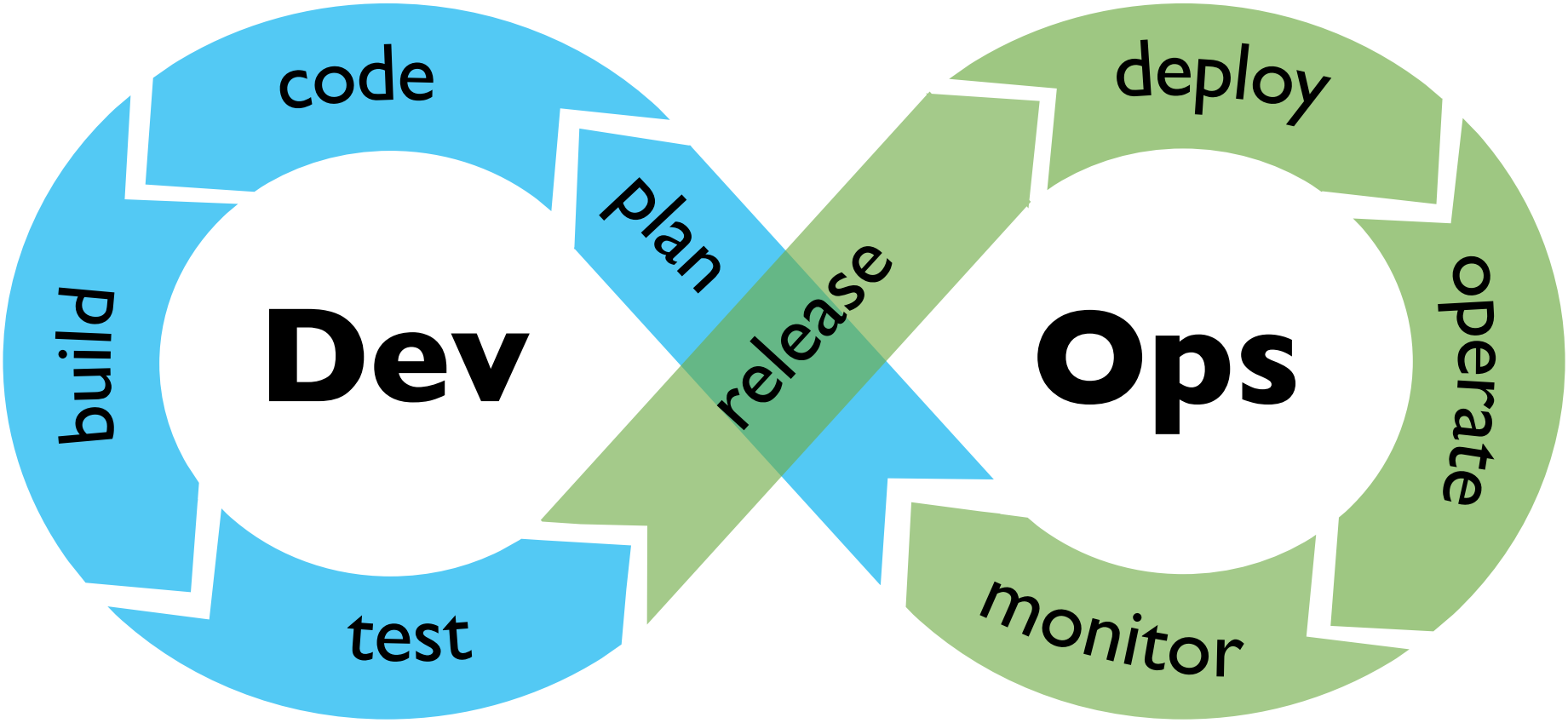


Optimization
(Find bottlenecks)

Scaling
(Add resources)

DevOps philosophy:

The static (development) and dynamic (operations) states of the software must be integrated to ensure the continuity of the software in the face of changes



Optimization
(Find bottlenecks)

Scaling
(Add resources)

Content

- **Autoscaling and self-adaptive systems**
- **Control theory**
- **Chaos monkey (Chaos engineering)**
- **Case studies**

Scaling

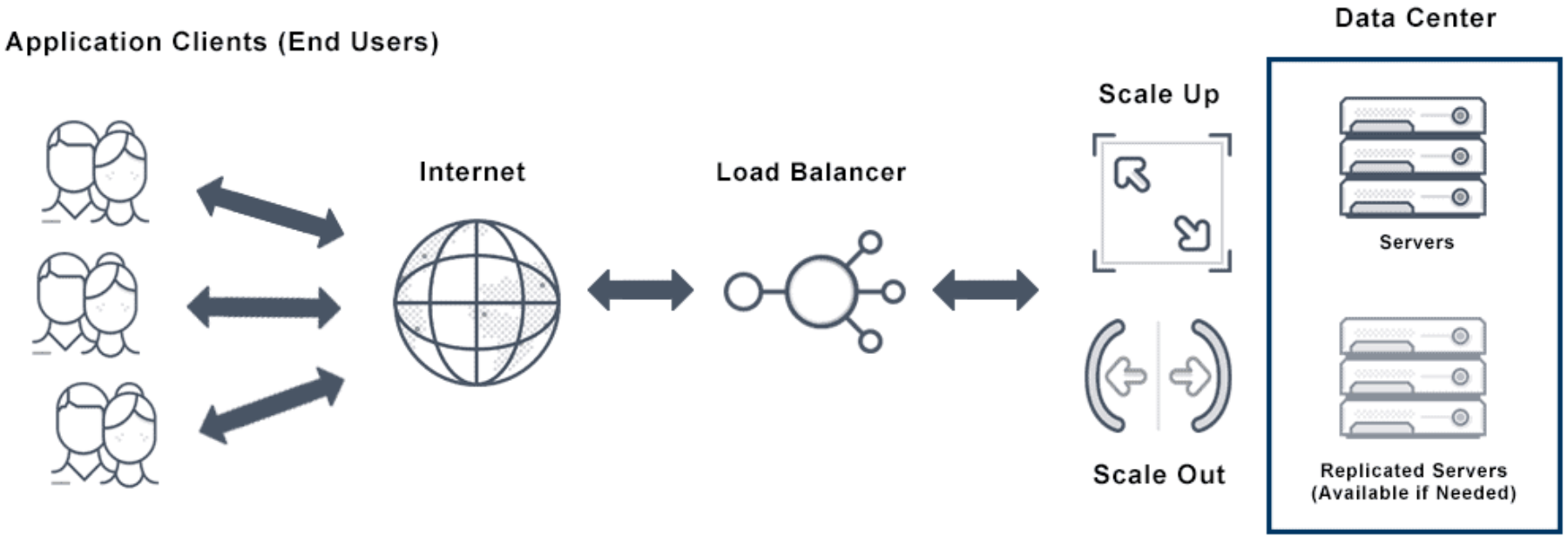


Image source: https://avinetworks.com/glossary/auto-scaling/

Scaling up: making a component bigger or faster so that it can handle more load.

Scaling out: adding more components in parallel to spread out a load.

Amazon DynamoDB Autoscaling

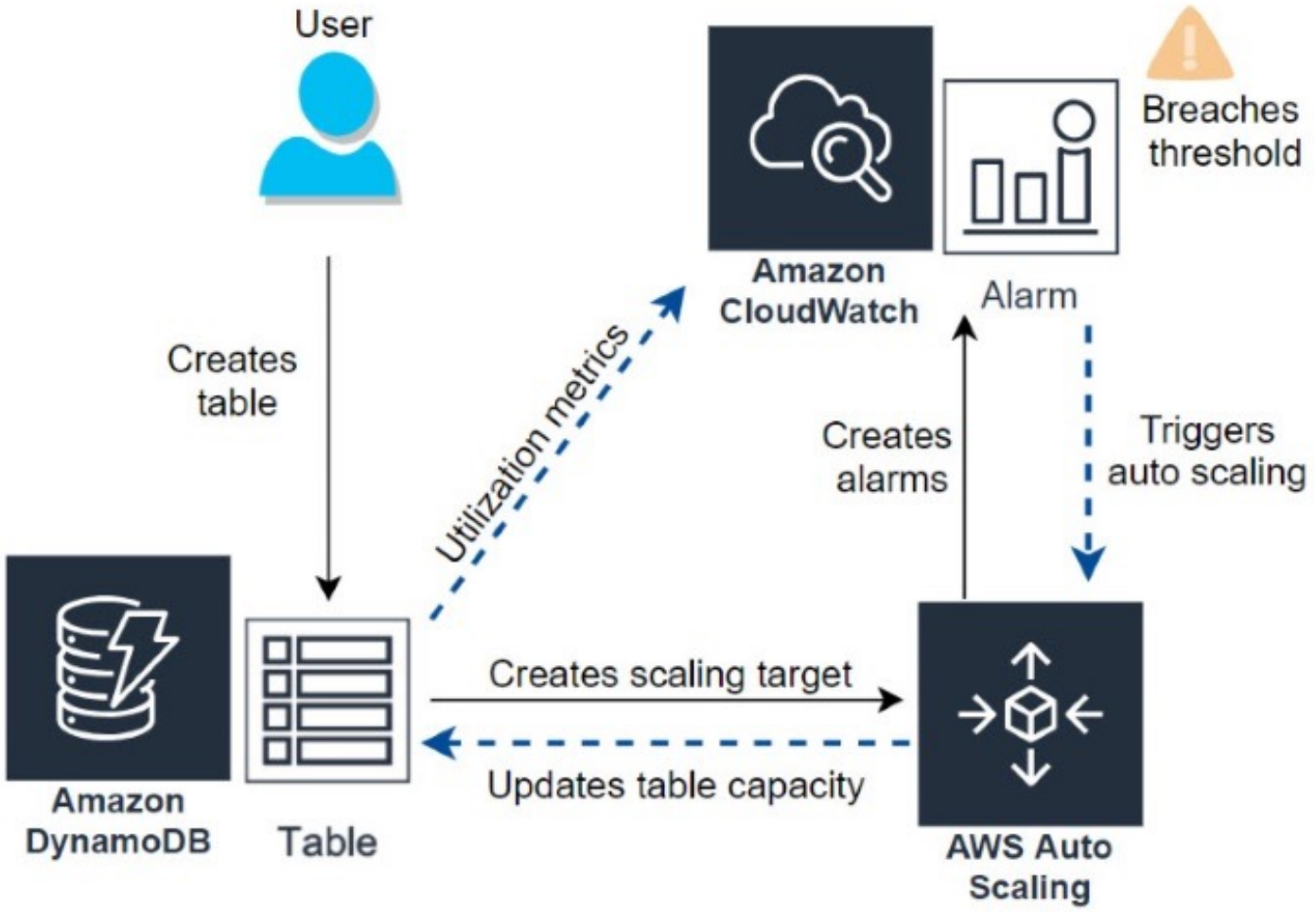


Image source: https://aws.amazon.com/blogs/database/amazon-dynamodb-auto-scaling-performance-and-cost-optimization-at-any-scale/

Provisioning without Autoscaling

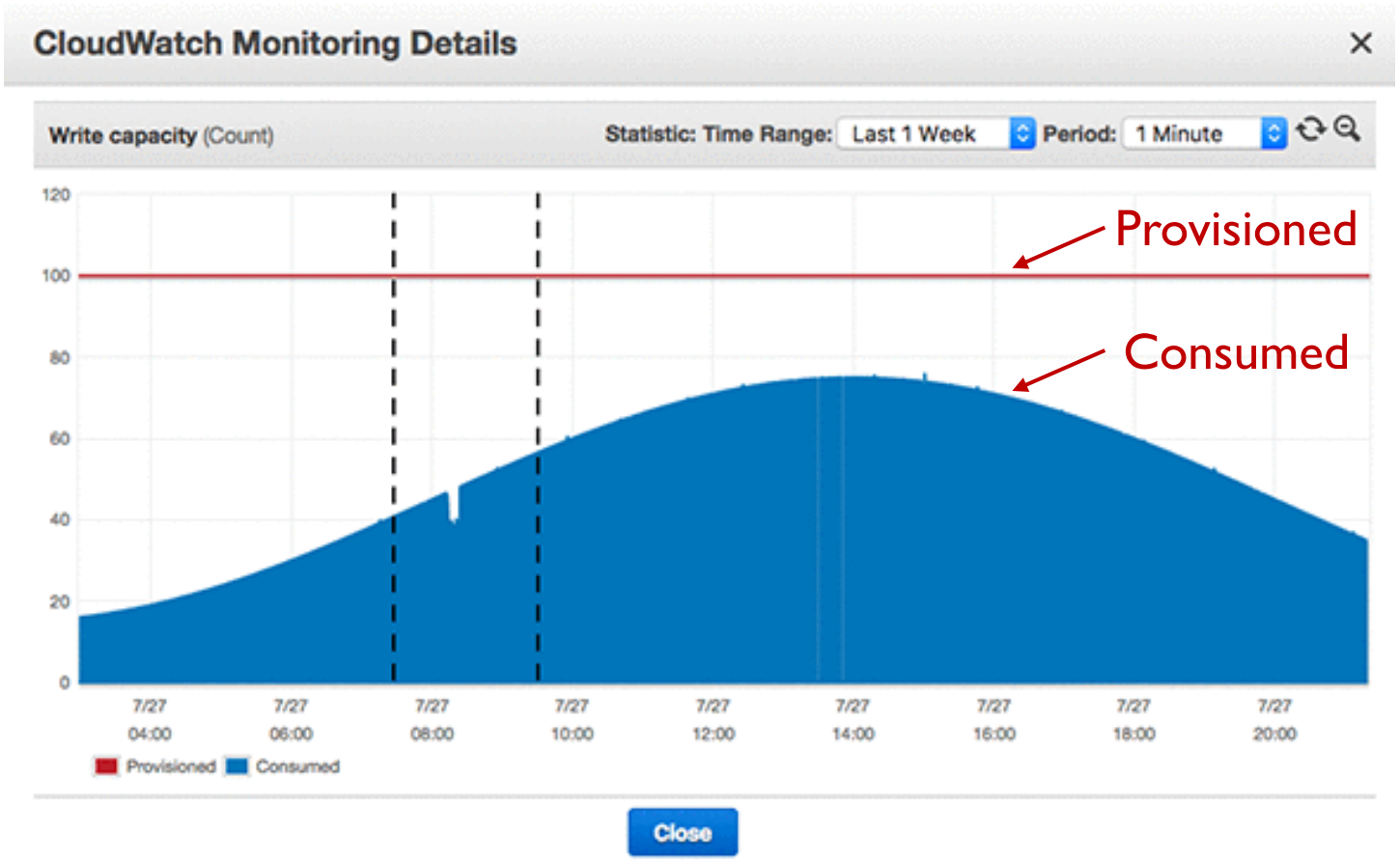


Image source: https://aws.amazon.com/blogs/database/amazon-dynamodb-auto-scaling-performance-and-cost-optimization-at-any-scale/

Provisioning with Autoscaling

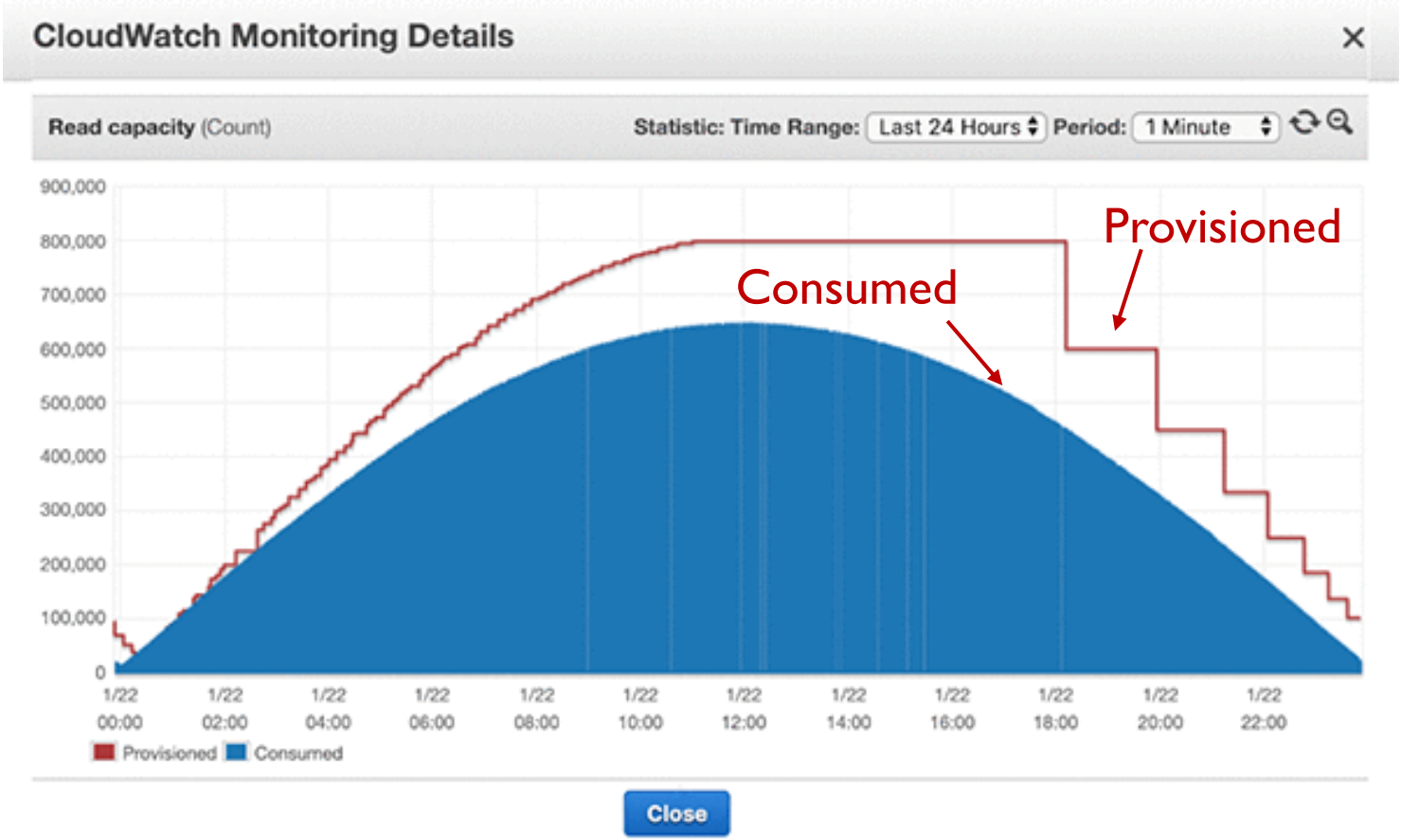


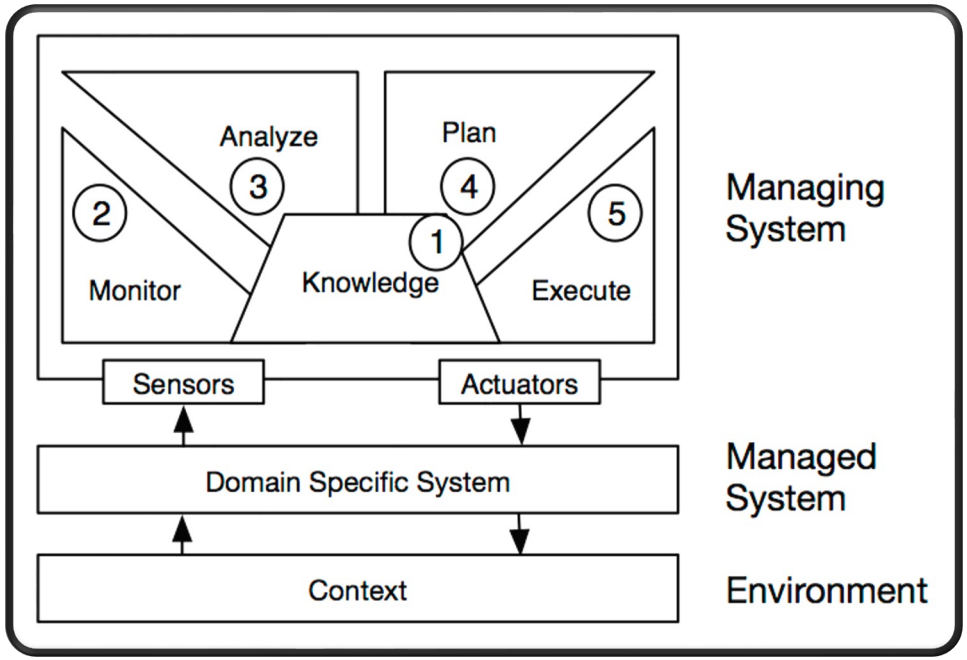
Image source: https://aws.amazon.com/blogs/database/amazon-dynamodb-auto-scaling-performance-and-cost-optimization-at-any-scale/

Self-adaptive systems

- Engineering of systems that can adapt.
- In fact, we can transform software to adapt or we can develop systems-managers that can adapt other software systems.
- Adaptation can refer to any quality.
  - ✓**Performance** - self-optimizing systems.
  - ✓**Security** - self-protective systems.
  - ✓**Reliability** - self-repairing systems.
  - ✓**Functionality** - self-configuring systems
- In 2005, IBM proposed an architectural framework for the design and development of self-adaptive software systems, **MAPE-K**.

MAPE-K

- **M**onitoring: The module to monitor software, infrastructure, and environment.
- **A**nalysis: The module to analyze the measurements and identify the problematic cases.
- **P**lanning: The module for planning adaptive actions and answering problems.
- **E**xecution: The module to apply adaptive actions to the software or its infrastructure.
- **K**nowledge: A knowledge base for past events and data that will be analyzed to improve future adaptive actions.



Monitoring

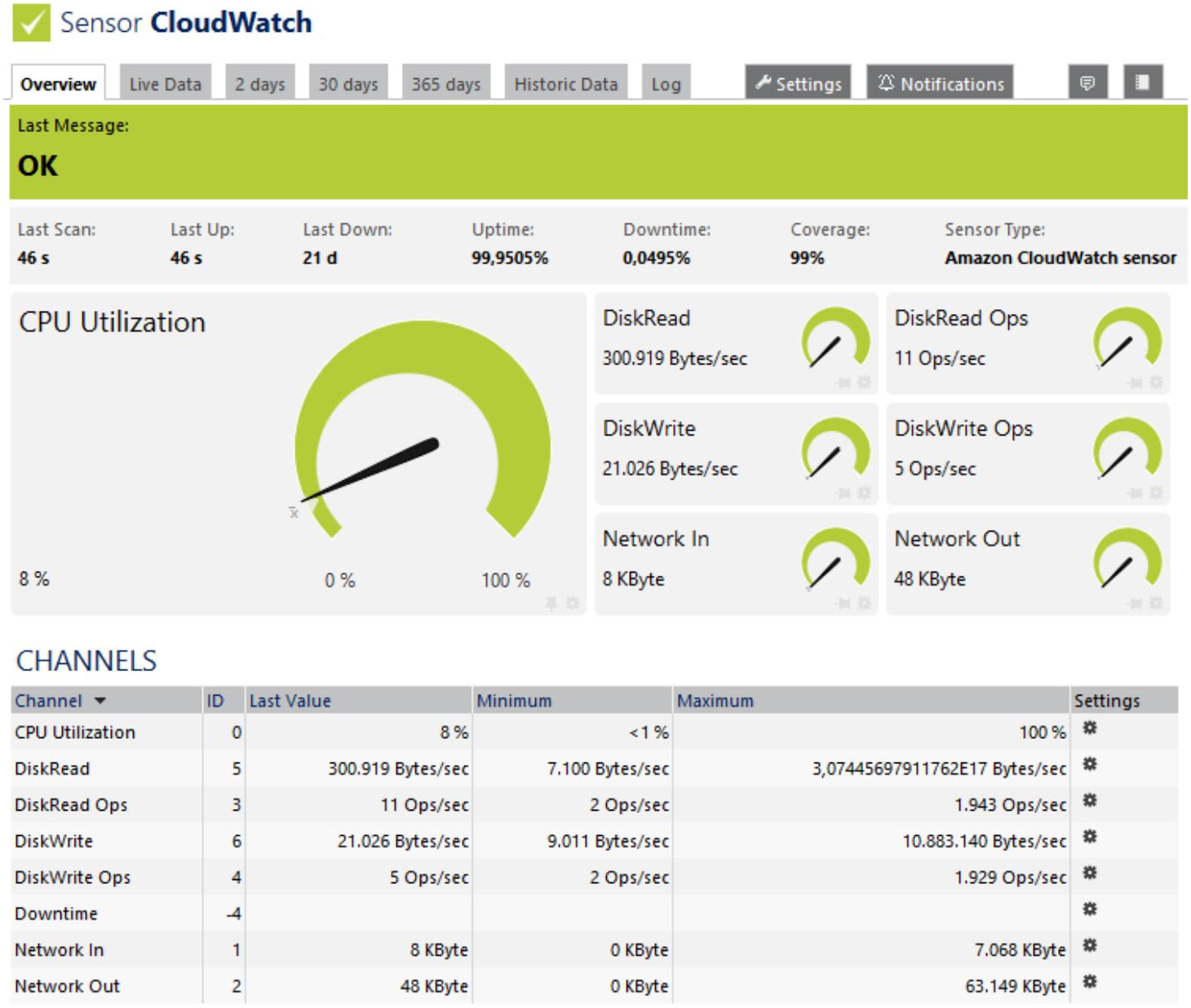
- Monitoring is the activity of measuring the software during the time of its execution, when the system is already deployed.
  - ✓Profiling is the measurement of a sometimes simulated execution during its development.
  - ✓The stress test is the measurement of the system under controlled conditions and before it is deployed to the final infrastructure.
- We can monitor the system at different levels.
  - ✓Application.
  - ✓Operating system
  - ✓Infrastructure (virtual machine, container)
  - ✓Equipment

Monitoring: challenges

The **different levels** have different challenges.

- At the hardware and infrastructure level, multilocation (the situation where multiple users share the same hardware for their applications) affects the reliability of the metrics for our application.
- At the operating system level, the coexistence of several processes also affects the reliability of the measurements.
- At the application level, we can measure the right response time from our side, not from the customer's point of view.
  - ✓ There is also the delay of the network to consider that we can not control.
- Like profiling, monitoring can be intrusive or external.
  - ✓ Live monitoring (the module surveys the system in real time) is intrusive and may affect the measurement, but it is very accurate.
  - ✓ External monitoring uses sampling. The measurements are stored (e.g., in a database) and the analysis module retrieves them asynchronously.

AWS CloudWatch



Channel	ID	Last Value	Minimum	Maximum	Settings
CPU Utilization	0	8 %	<1 %	100 %	
DiskRead	5	300.919 Bytes/sec	7.100 Bytes/sec	3,07445697911762E17 Bytes/sec	
DiskRead Ops	3	11 Ops/sec	2 Ops/sec	1.943 Ops/sec	
DiskWrite	6	21.026 Bytes/sec	9.011 Bytes/sec	10.883.140 Bytes/sec	
DiskWrite Ops	4	5 Ops/sec	2 Ops/sec	1.929 Ops/sec	
Downtime	-4				
Network In	1	8 KByte	0 KByte	7.068 KByte	
Network Out	2	48 KByte	0 KByte	63.149 KByte	

Analysis

- The analysis module is responsible for monitoring the health of the monitored system and identifying problematic situations.
- The simplest analysis is to compare measurements at certain thresholds.
  - ✓ Eg CPU <80%
- A more sophisticated analysis is to compare measures to predictions or estimates of a performance model.
  - ✓ By knowing the workload and resource demands, resource consumption and expected response time can be estimated and compared to current measures.
- In any case, we can define rules (IF-THEN) for the analysis.
- The rules can be integrated into the monitoring module that can specify alerts.

Planning

- Planning is responsible for designing adaptive actions to respond to problematic cases identified by the analysis.
- Depending on the level of monitoring and the results of the analysis, the software or its infrastructure can be adapted.
- One can have a static or dynamic planning.
  - Static: The planner has a fixed correspondence between problematic cases and adaptive actions.
    - ✓ Eg Add a server when the CPU usage exceeds the 80% threshold.
  - Dynamic: According to the results of the analysis, the planner will adjust the adaptive action.
    - ✓ Eg If the difference between the current measurement and the last measurement is more than 30% add two servers.

Execution

- The execution module consists of actuators that apply the adaptive actions designed by the planner.
- If the actions occur at the application level, one should follow the processes of continuous integration.
  - ✓Run all the tests, submit the new version to the repository, update the other artifacts etc.
- For actions that adapt infrastructure, cloud service providers provide actuators through APIs.
  - ✓Or through graphical interfaces for non-automatic scaling.

Scaling types

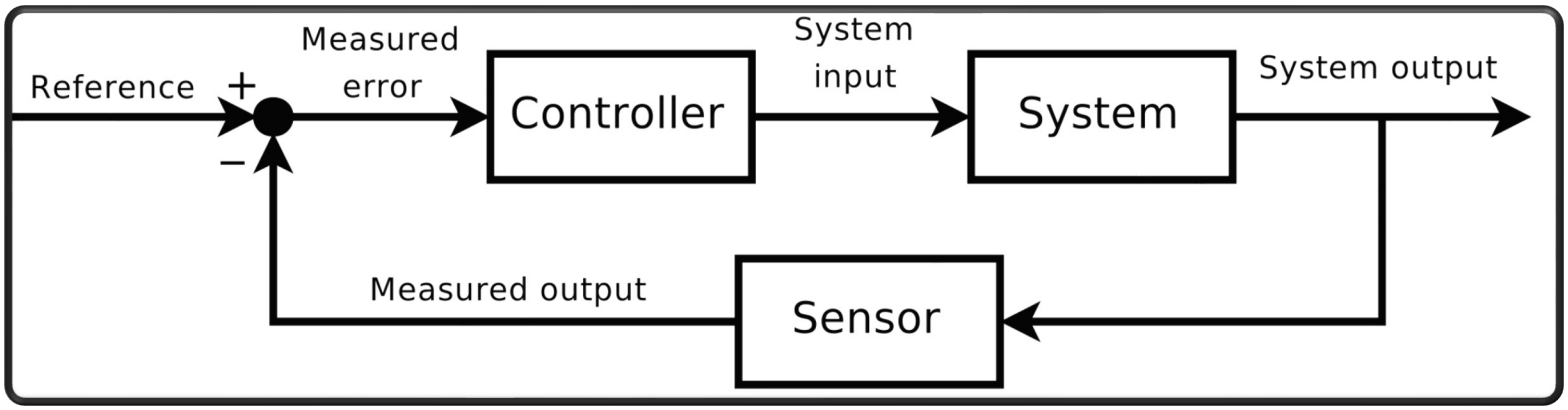
- Scheduled scaling: Workload is planned for a period of time and scaling actions are programmed (eg adding resources during peak hours or during holidays).
  - ✓ Benefit: There is no need to manage the system continuously.
  - ✓ Disadvantage: The system can not react to unforeseen load changes.
- Reactive Scaling: Current system measurements are analyzed and exceptional cases are responded to.
  - ✓ Benefit: The system can respond to sudden and unexpected changes in workload.
  - ✓ Disadvantage: The variation can be temporary and we will change the system infrastructure too frequently.

Scaling types (cont'd)

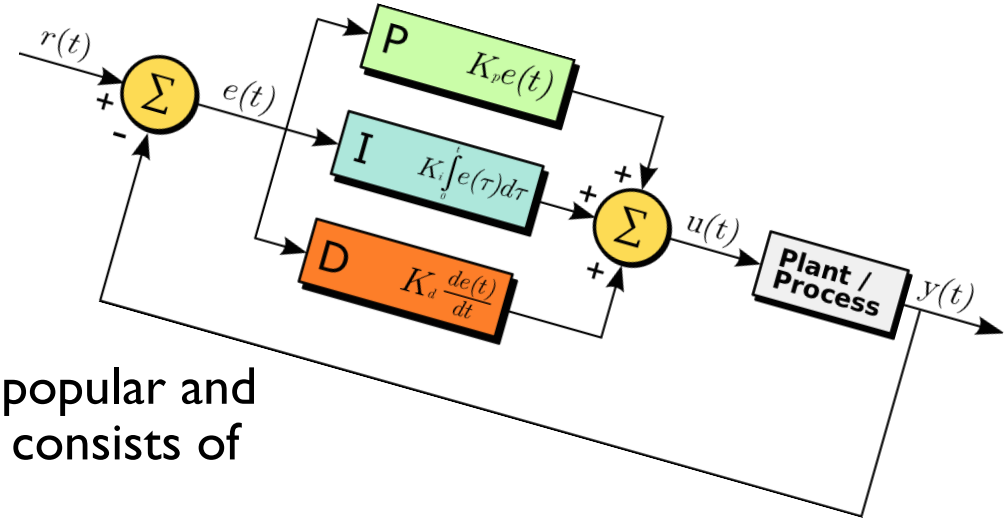
- Cumulative scaling (reactive): Continuous, not instantaneous violations are considered before applying the adaptations.
  - ✓ Advantage: It increases the certainty of the need for scaling.
  - ✓ Disadvantage: It delays the adaptation and temporarily deteriorates the performance.
- Predictive scaling: The workload is predicted for the near future, and the system is adapted before a problematic case occurs.
  - ✓ Advantage: System performance can be guaranteed accurately and quickly.
  - ✓ Disadvantage: It is not always possible to predict exactly the workload. It is very possible that we will lose exceptional cases.

Control theory

- We can implement a management system by following the control theory for physical systems.
- The output of a system is monitored by a sensor.
- The monitored value is compared to a reference value (an objective, a threshold or an estimate).
- The error between the two values is entered into a controller that will calculate an input (an adaptive action) to fit the system.



PID controller



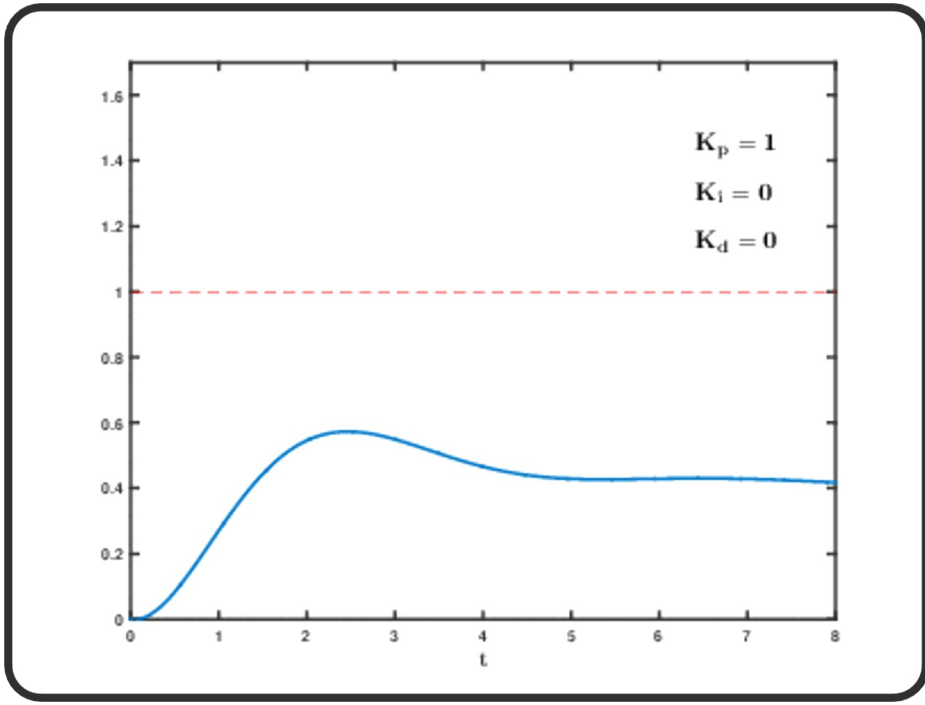
The PID controller is one of the most popular and commonly used types of controllers. It consists of three parts:

- **P**roportional: It measures the current error between the reference value and just the current measurement. It represents the reactive aspect of adaptation.
- **I**ntegral: It measures the cumulative error between the reference value and the past measurements. It represents the cumulative aspect of adaptation.
- **D**erivative: It "measures" the future error. In fact, it is an estimate of the tendency of the error. It represents the predictive aspect of adaptation.

The three parts make up the error according to which the input to adapt the system is calculated.

Controller properties

- **Rise time**: The time to increase the output to the reference value.
- **Overshoot**: The percentage of the error when the output increases more than the reference value.
- **Settling time**: the time it takes for the error to become minimal
- **Stability**: A state of minimal error.
- **Steady-error state**: A non-zero error state that is needed for the proportional part.



What runtime problems should we expect and prepare our self-adaption mechanisms for?

Chaos Monkey

A flat tire

- Is your spare tire inflated?
- Do you have the tools in the trunk?
- Do you even know how to change a tire?
- Solution
  - Have a monkey puncture your tire once a week
  - And fix it
  - So that when you have a flat on the highway you'll be ready

Software is Artificial

- Chaos is expensive in the real physical environment, but
- “but can be (almost) free and automated in the cloud.”

Chaos Monkey

- Randomly disables production instances to make sure that the system is robust enough to survive this common type of failure without any customer impact.

Chaos is added in production

"By running Chaos Monkey in the **middle of a business day**, in a **carefully monitored environment** with **engineers standing by** to address any problems, we can still learn the lessons about the weaknesses of our system, and build **automatic recovery mechanisms** to deal with them. **So next time an instance fails at 3 am on a Sunday, we won't even notice.**"

Types of Chaos

- Original Chaos Monkey: bring down random production instances
- Latency Monkey induces artificial delays in communication layer
- Doctor Monkey taps into health checks to check for system health (eg CPU load) and remove unhealthy instances
- Janitor Monkey find unused resources and gets rid of them
- Conformity Monkey finds instances that don't adhere to best-practices and kill them
- Security Monkey finds security violations or vulnerabilities and kill the associated instances
- 10–18 Localization Monkey detects problems caused by multiple geographic regions and languages
- Chaos Gorilla simulates an outage of an entire availability zone

Latency Monkey

- Latency Monkey induces artificial delays in communication layer
- Simulate service degradation
- Measures if upstream services respond appropriately.
- Large delays simulate node or service being **down**
  - Without actually bringing down the service (ie other parts of system still run normally)
- Good for testing the fault tolerance of a **new service**

Doctor Monkey

- Health checks that run on each instance as well as monitors other external signs of health (e.g. CPU load)
- Goal: check for unhealthy instances.
- Unhealthy instances are removed from service (ie end users don't see them)
- Keep service up, but only for devs, so the service owners can find the problem
- Once the root-cause is found, the service is terminated

Janitor Monkey

- Ensures that the environment is running free of clutter and waste
- Searches for unused resources and disposes of them

Conformity Monkey

- Check for instances that don't adhere to best-practices and shuts them down
- Don't tolerate laziness
- For example, a service that isn't part of auto-scaling will fail when requests surge
- Shut it down now, so that it doesn't happen at a peak period

Security Monkey (special case of conformity)

- Finds security violations or vulnerabilities,
- For example, improperly configured AWS security groups,
- Terminates the offending instances.
- Also ensures that all SSL and DRM certificates are valid and are not coming up for renewal.

10–18 Monkey Localization-Internationalization

- Detects configuration and run time problems in instances serving customers in multiple geographic regions, using different languages and character sets.

Chaos Gorilla

- Simulates an outage of an entire availability zone.
- Want to verify that services automatically re-balance to the functional availability zones without user-visible impact or manual intervention.
- Eg remove North American Amazon instances and run from Europe

Types of Chaos

- Original Chaos Monkey: bring down random production instances
- Latency Monkey induces artificial delays in communication layer
- Doctor Monkey taps into health checks to check for system health (eg CPU load) and remove unhealthy instances
- Janitor Monkey find unused resources and gets rid of them
- Conformity Monkey finds instances that don't adhere to best-practices and kill them
- Security Monkey finds security violations or vulnerabilities and kill the associated instances
- 10–18 Localization Monkey detects problems caused by multiple geographic regions and languages
- Chaos Gorilla simulates an outage of an entire availability zone

Summary

- Chaos is added in production in the middle of the business day
- with careful monitoring
- and engineers standing by to fix problems
- so that
- weaknesses will be identified
- auto-recovery mechanisms will be build
- so that
- a 3 am failure won't need a manual interventions

Case studies

- *Autonomous configuration*
- *Workload prediction*
- *Autoscaling of containers*
- *Comparison between autoscaling methods (rules, model, controller)*

Autonomous configuration 2018

Adopting Autonomic Computing Capabilities in Existing Large-Scale Systems

An Industrial Experience Report

Heng Li
Queen's University, Canada
hengli@cs.queensu.ca

Tse-Hsun (Peter) Chen
Concordia University, Canada
peterc@encs.concordia.ca

Ahmed E. Hassan
Queen's University, Canada
ahmed@cs.queensu.ca

Mohamed Nasser
BlackBerry, Canada

Parminder Flora
BlackBerry, Canada

ABSTRACT

In current DevOps practice, developers are responsible for the operation and maintenance of software systems. However, the human costs for the operation and maintenance grow fast along with the increasing functionality and complexity of software systems. Autonomic computing aims to reduce or eliminate such human intervention. However, there are many existing large systems that did not consider autonomic computing capabilities in their design. Adding autonomic computing capabilities to these existing systems is particularly challenging, because of 1) the significant amount of efforts that are required for investigating and refactoring the existing code base, 2) the risk of adding additional complexity, and 3) the difficulties for allocating resources while developers are busy adding core features to the system. In this paper, we share our industrial experience of re-engineering autonomic computing capabilities to an existing large-scale software system. Our autonomic computing

CCS CONCEPTS

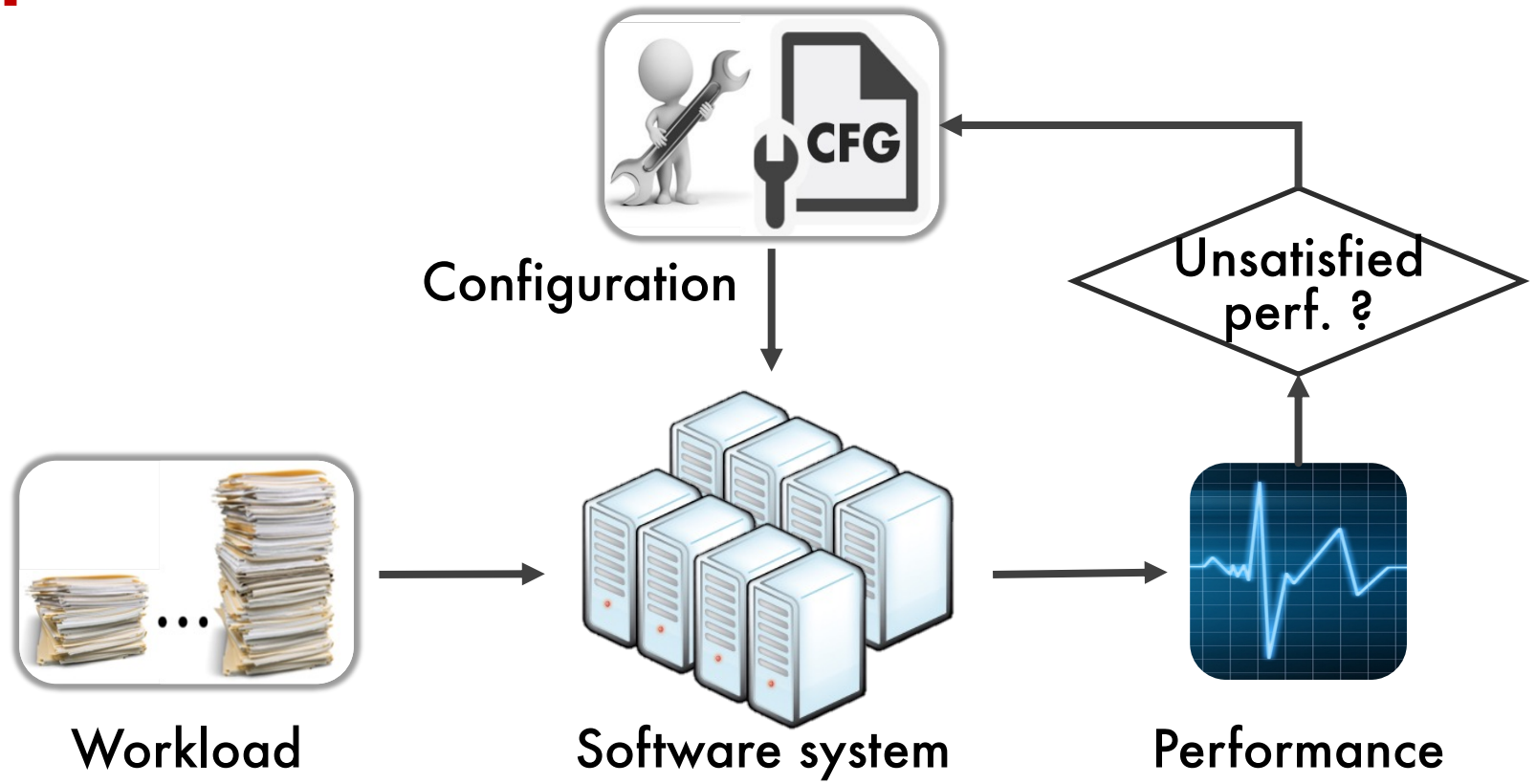
• **Software and its engineering** → *Software development process management*; • **Computer systems organization** → *Self-organizing autonomic computing*;

KEYWORDS

Autonomic computing, software re-engineering, performance engineering, software testing

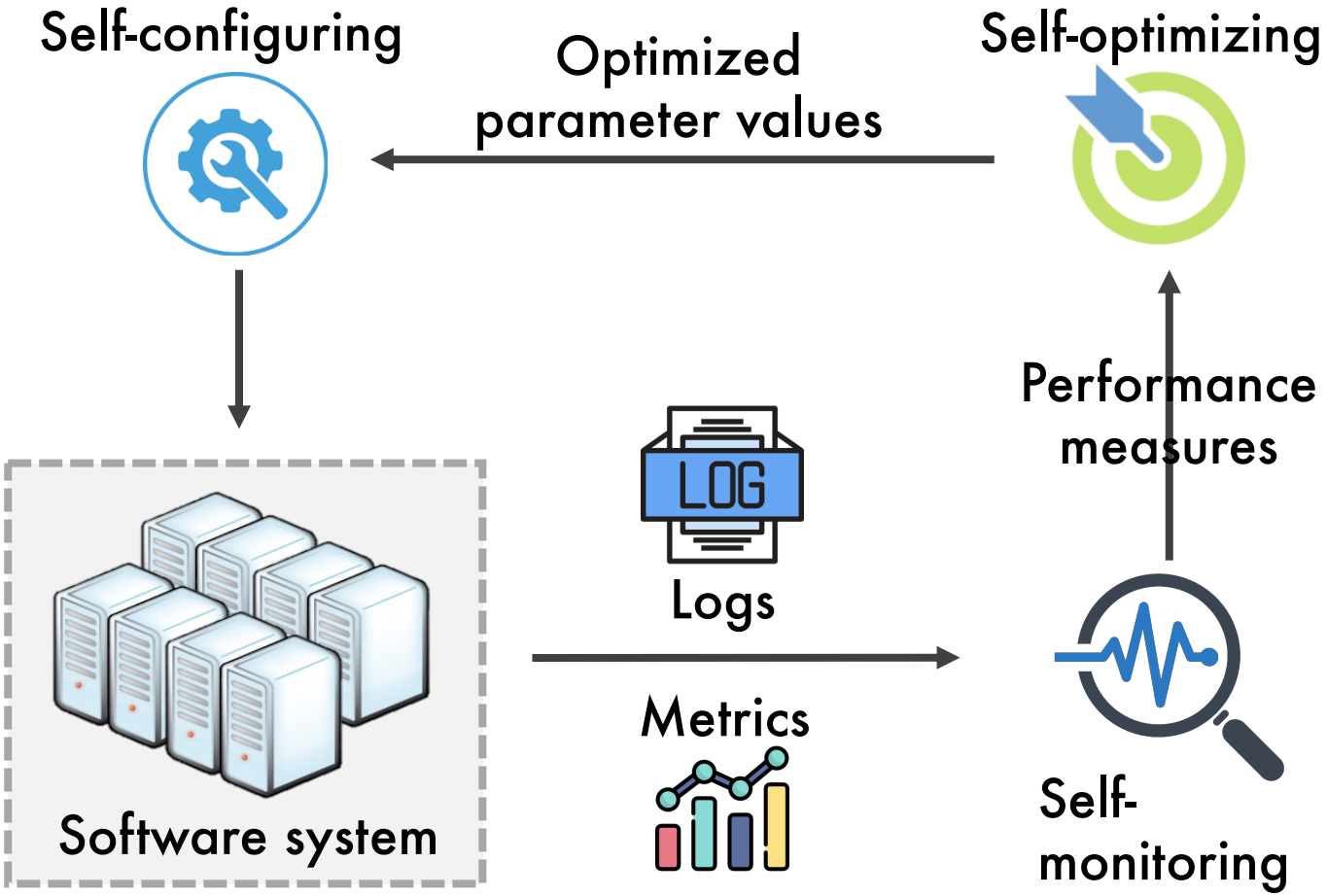
ACM Reference Format:
Heng Li, Tse-Hsun (Peter) Chen, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. 2018. Adopting Autonomic Computing Capabilities in Existing Large-Scale Systems: An Industrial Experience Report. In *ICSE-SEIP '18: 40th International Conference on Software Engineering: Software Engineering in Practice Track, May 27-June 3 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3183519.3183544

Manually configuring large-scale software systems is costly & error-prone



Workloads are constantly evolving, requiring constant human intervention to ensure optimal performance

Using an AIOps solution to autonomously tune system configurations



Challenges in autonomous configuration of large-scale software systems

Complex system behavior

Fast response to environment

Minimal footprint

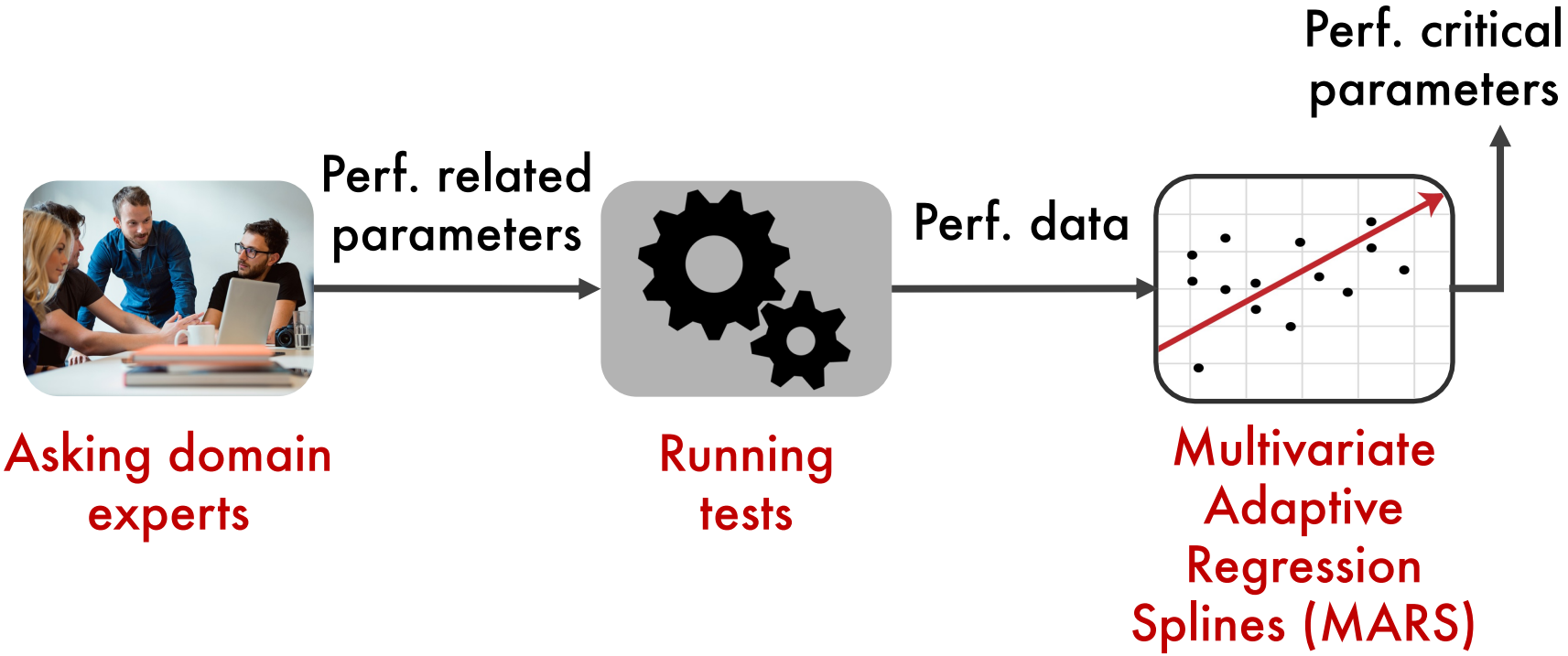
Challenges in autonomous configuration of large-scale software systems

Complex system behavior

Fast response to environment

Minimal footprint

Understanding the relationship between config. parameters and performance metrics



Only a few out of many candidate parameters significantly impact system performance

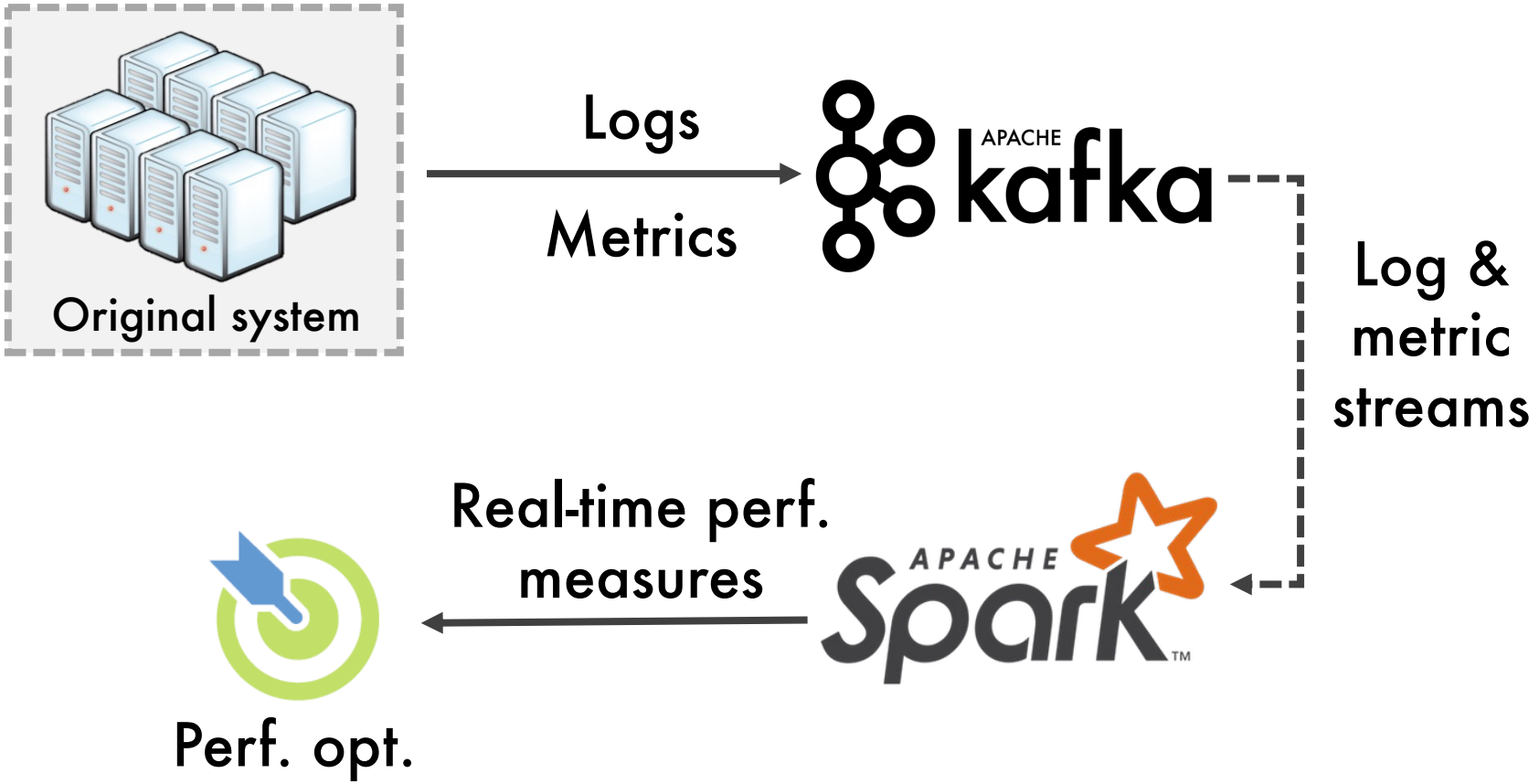
Challenges in autonomous configuration of large-scale software systems

Complex system behavior

Fast response to environment

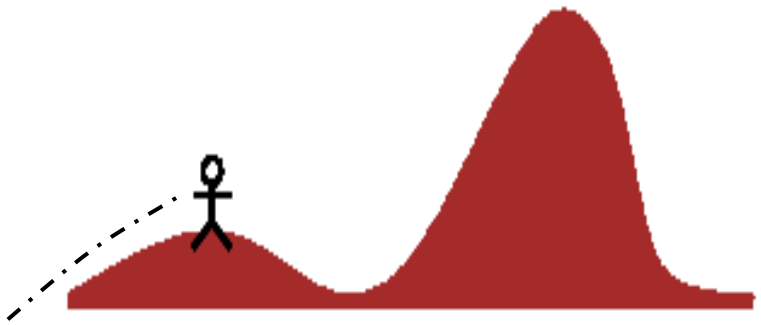
Minimal footprint

Real-time monitoring of system behavior using readily-existing logs and metrics data



Fast search for optimal configurations (Hill Climbing)

"Like climbing Everest in thick fog with amnesia"



- Continually moves in the direction of increasing values (uphill).
- Terminates when it reaches a peak where no neighbor has a higher value.
- Fast & local optimization.

Fast response to workload changes (within seconds)

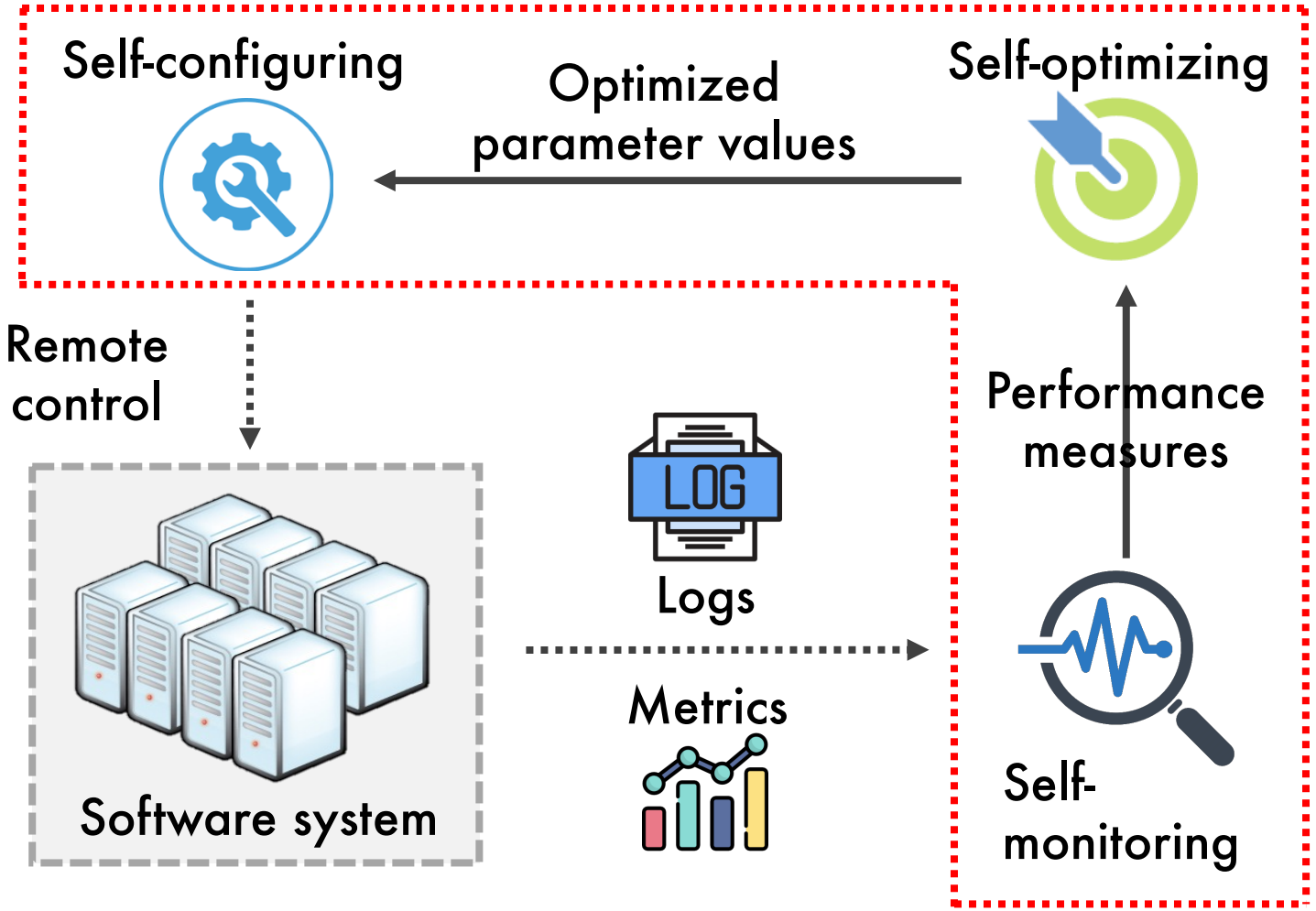
Challenges in autonomous configuration of large-scale software systems

Complex system behavior

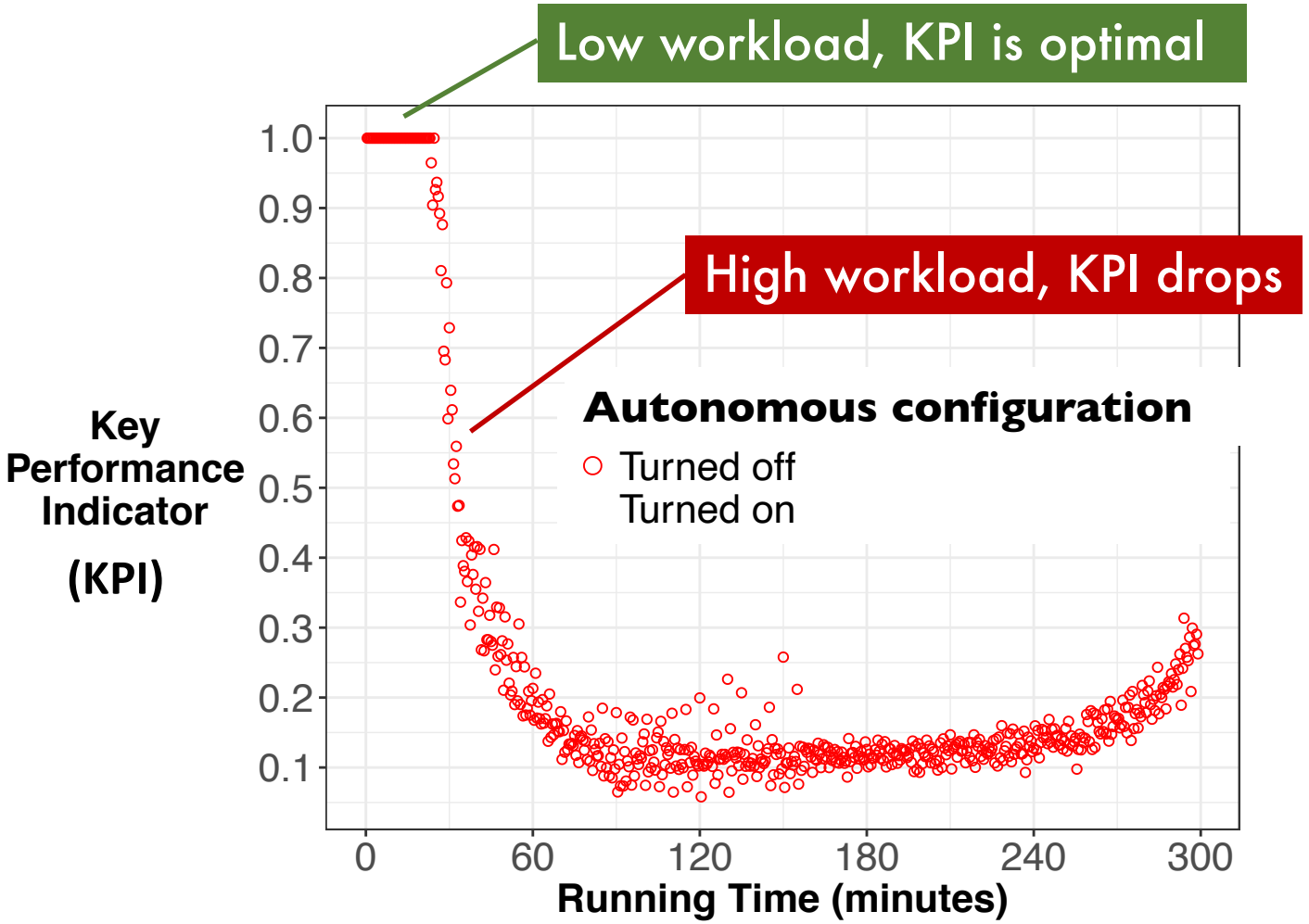
Fast response to environment

Minimal footprint

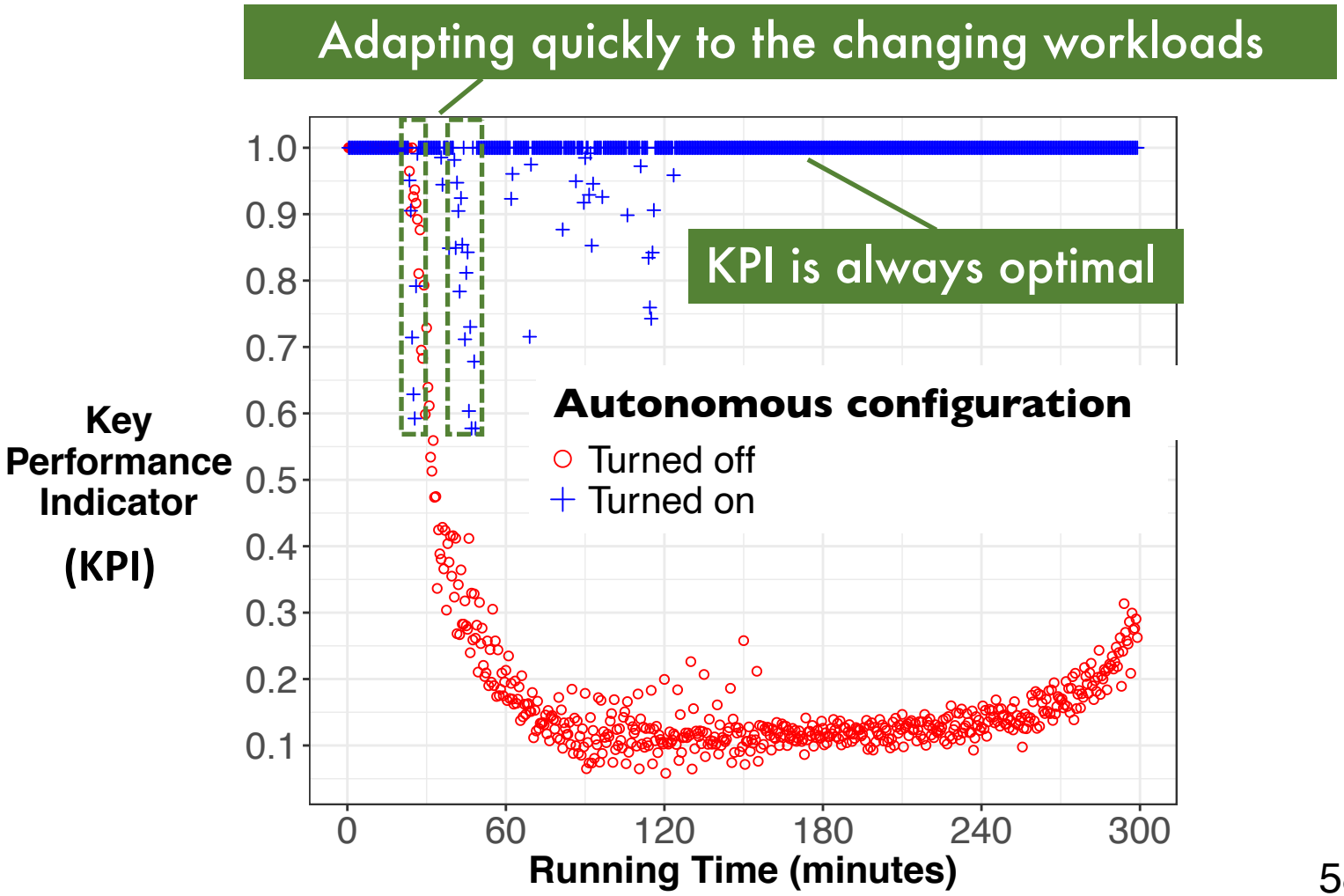
Separating the autonomous configuration capabilities from the original system



Autonomous configuration significantly improves system performance



Autonomous configuration significantly improves system performance



Workload Prediction 2015

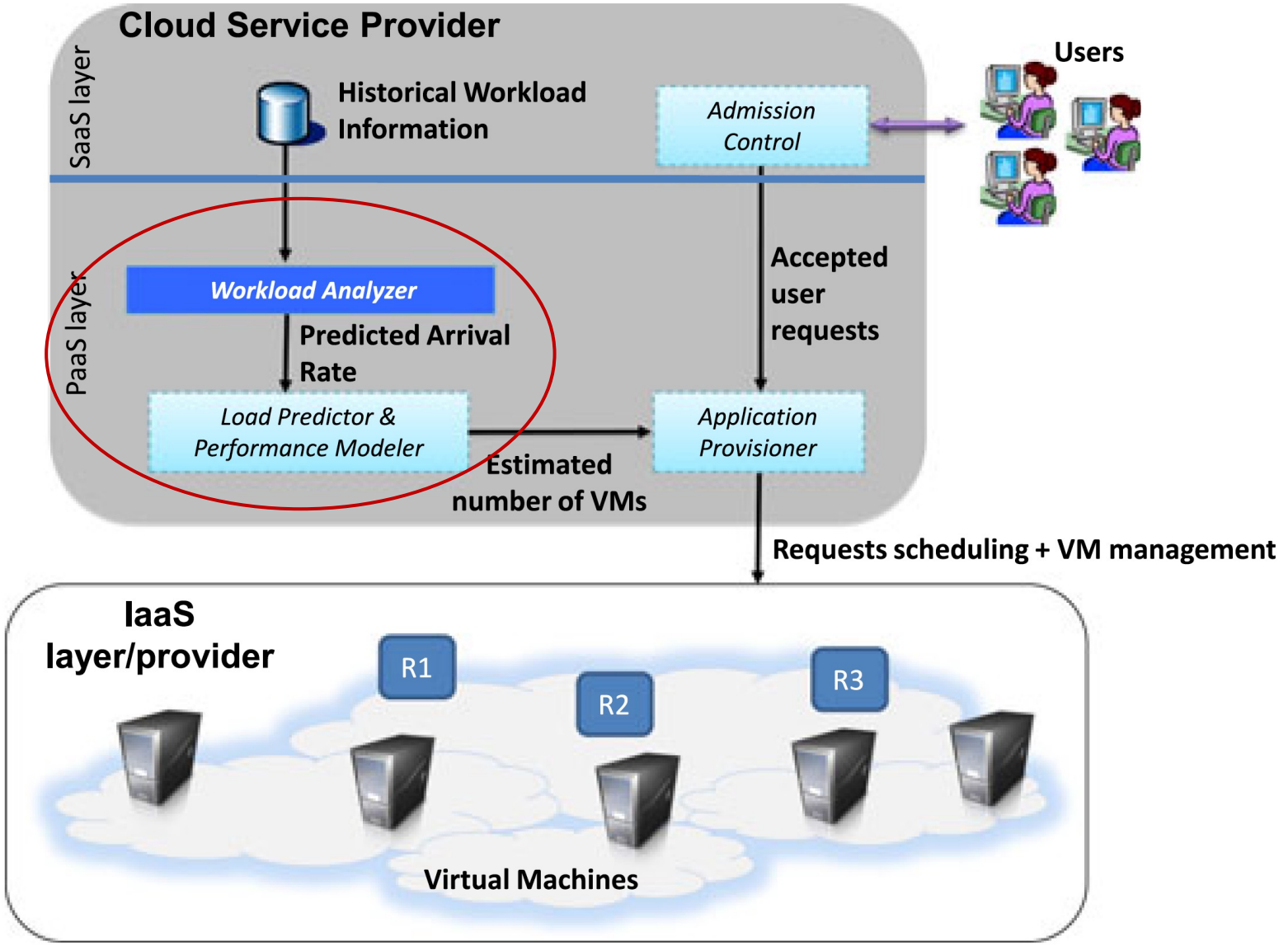
Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS

Rodrigo N. Calheiros, Enayat Masoumi, Rajiv Ranjan, and Rajkumar Buyya

Abstract—As companies shift from desktop applications to cloud-based software as a service (SaaS) applications deployed on public clouds, the competition for end-users by cloud providers offering similar services grows. In order to survive in such a competitive market, cloud-based companies must achieve good quality of service (QoS) for their users, or risk losing their customers to competitors. However, meeting the QoS with a cost-effective amount of resources is challenging because workloads experience variation over time. This problem can be solved with proactive dynamic provisioning of resources, which can estimate the future need of applications in terms of resources and allocate them in advance, releasing them once they are not required. In this paper, we present the realization of a cloud workload prediction module for SaaS providers based on the autoregressive integrated moving average (ARIMA) model. We introduce the prediction based on the ARIMA model and evaluate its accuracy of future workload prediction using real traces of requests to web servers. We also evaluate the impact of the achieved accuracy in terms of efficiency in resource utilization and QoS. Simulation results show that our model is able to achieve an average accuracy of up to 91 percent, which leads to efficiency in resource utilization with minimal impact on the QoS.

Index Terms—Cloud computing, workload prediction, ARIMA

Adaptive Cloud Provisioning



Time Series Forecasting (ARIMA)

ARIMA: Autoregressive integrated moving average

Forecast for y at time t = constant

AR term

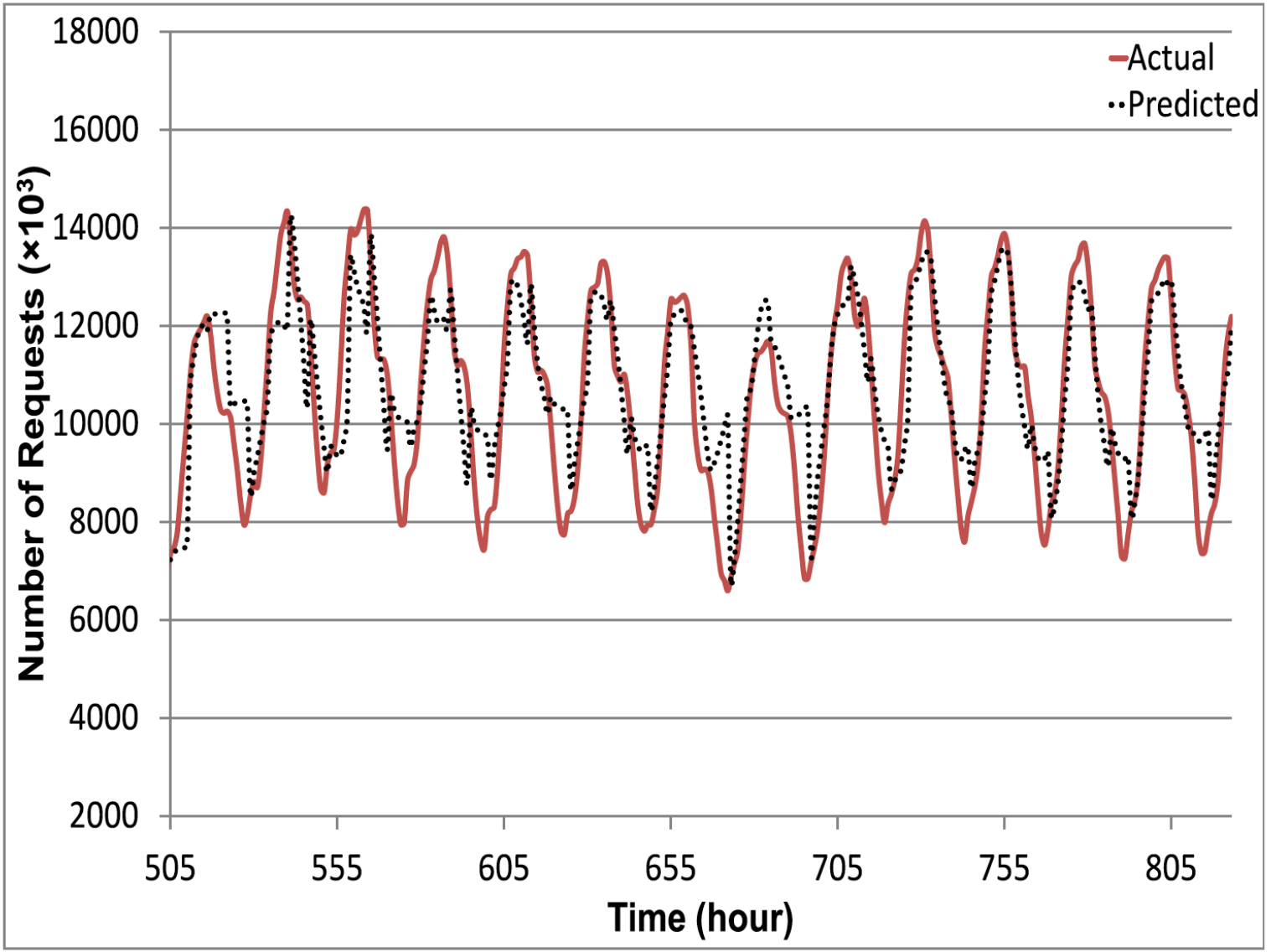
+ weighted sum of the last p values of y

+ weighted sum of the last q forecast errors

MA term

$$\hat{y}_t = \mu + \varphi_1 y_{t-1} + \cdots + \varphi_p y_{t-p} - \theta_1 e_{t-1} - \cdots - \theta_q e_{t-q}$$

Predicted vs. Actual Workload



Controller SEAMS 2017

Delivering Elastic Containerized Cloud Applications to Enable DevOps

Cornel Barna, Hamzeh Khazaei, Marios Fokaefs and Marin Litoiu
Department of Electrical Engineering and Computer Science
York University
Toronton, ON, Canada
{cornel,mlitoiu}@cse.yorku.ca, {hkh,fokaefs}@yorku.ca

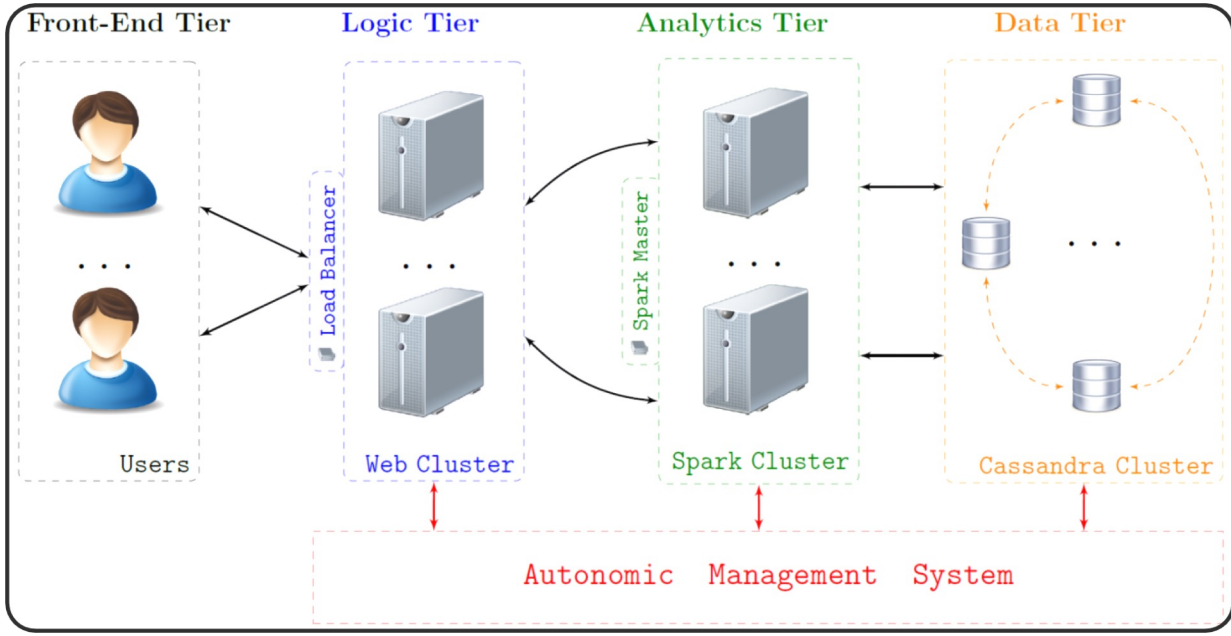
***Abstract*—Following recent technological advancements in software systems, like microservices, containers and cloud systems, DevOps has risen as a new development paradigm. Its aim is to bridge the gap between development and management of software systems and enable continuous development, deployment and integration. Towards this end, automated tools and management systems play a crucial role. In this work, we propose a method to develop an autonomic management system for multi-tier, multi-layer data-intensive containerized applications based on a performance model of such systems. The model is shown to be robust and accurate in estimating and predicting the system's performance for various workloads and topologies, while the AMS is capable of regulating the application's behaviour by taking independent actions on its various parts.**

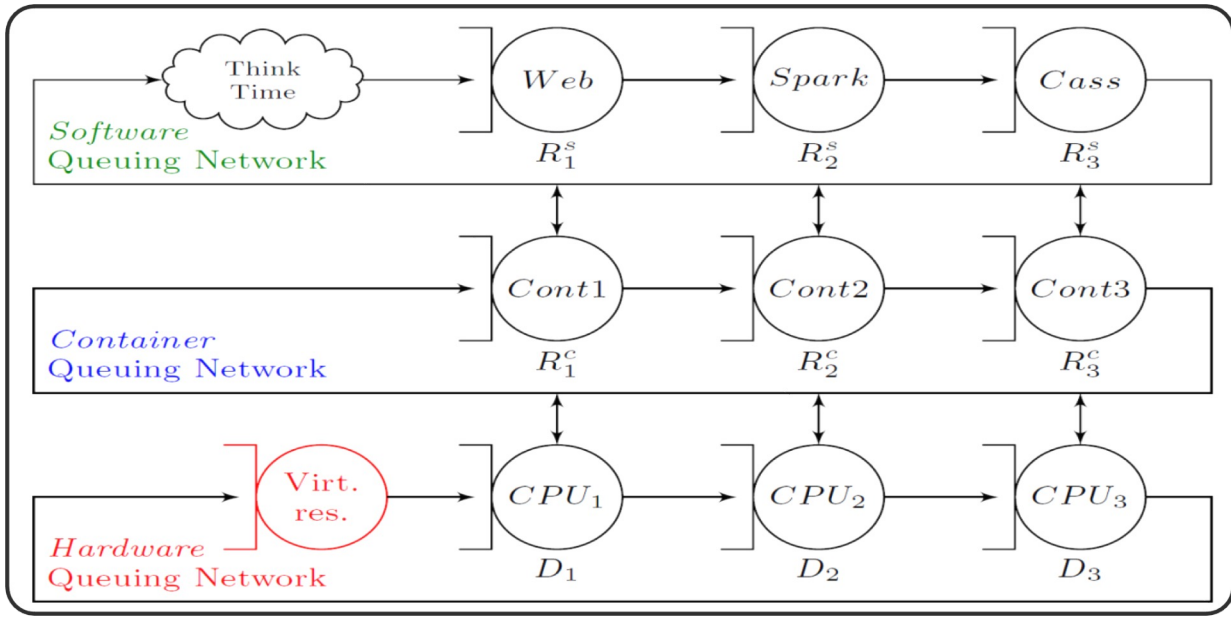
***Keywords*-devops; cloud computing; autonomic management systems; performance models; continuous delivery; containers; scaling; multi-tier big data applications;**

available solutions to solve recurring performance problems may actually create additional challenges for the developers. One important decision is about the functionality and the architecture of the AMS. For complex software systems that consist of many microservices and utilize a number of virtualization technologies, one has to wonder whether there is enough control and knowledge over the module to design a single, overarching AMS or multiple independent ones for each module. Another question is if there are multiple adaptive actions that can achieve the same goal, which one we pick and according to what criteria. In this situation, *models* can be of particular help as they give the opportunity to consider a number of different actions and solutions, evaluate them and, eventually, through systematic decision-making processes pick the optimal one. The models are part of the analysis and

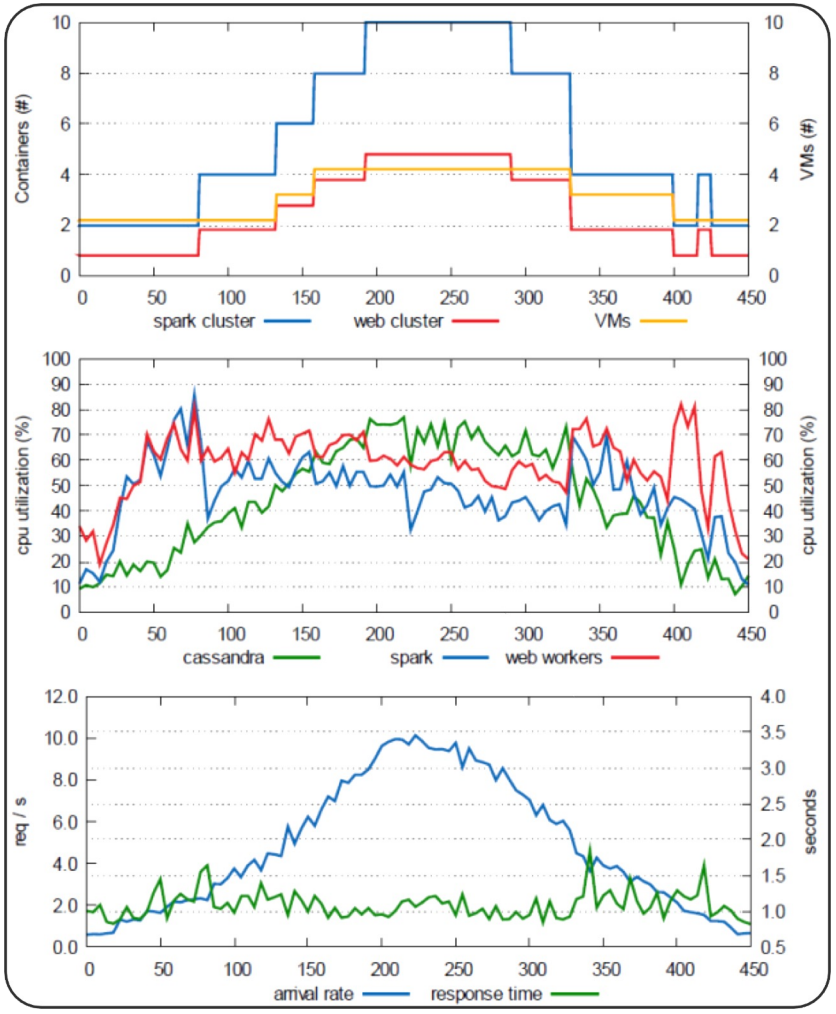
Context

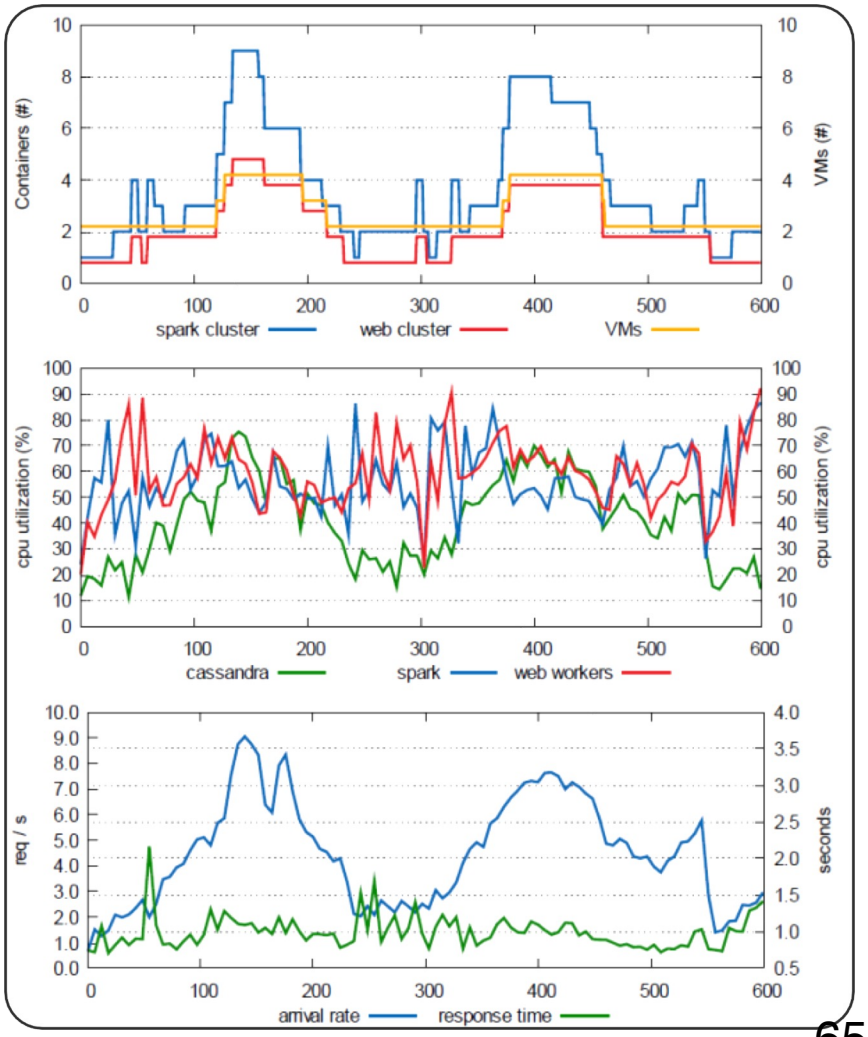
- A multilevel application that consists of a functional level, a level of analysis, and a level of data.
- A management system based on a QN model.
- The goal is to manage the performance of the functional level and the level of analysis by applying Docker container scaling.





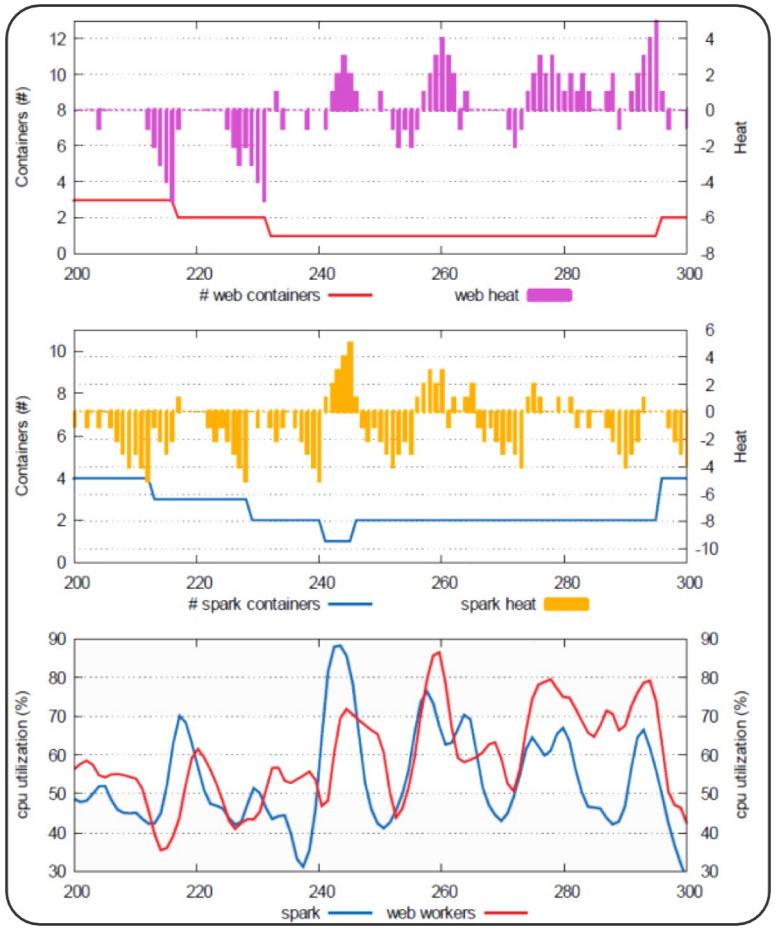
Results


Containers (#)
VMs (#)
spark cluster
web cluster
VMs
cpu utilization (%)
cassandra
spark
web workers
req / s
seconds
arrival rate
response time


Containers (#)
VMs (#)
spark cluster
web cluster
VMs
cpu utilization (%)
cassandra
spark
web workers
req / s
seconds
arrival rate
response time

HEAT algorithm

- The HEAT algorithm is an example of cumulative adaptation.
- A "heat" index is increased each time the measurement violates the reference value (the threshold).
- When the heat value is 5 we add a server.
- If the measurement does not violate the threshold, do not reset the heat value to zero, but decrease it by one.
- Over here, past mistakes are considered.



Cloud 2017

Evaluating Adaptation Methods for Cloud Applications: An Empirical Study

Marios Fokaefs, Yar Rouf, Cornel Barna and Marin Litoiu
Department of Electrical Engineering and Computer Science
York University
Toronto, ON, Canada
Email: fokaefs@yorku.ca, yarrouf@my.yorku.ca, cornel@cse.yorku.ca, mlitoiu@yorku.ca

***Abstract*—Web software systems generally reside in highly volatile environments; their incoming traffic may be subject to sharp fluctuations from reasons that cannot always be captured or predicted. Cloud computing provides a solution to this problem by offering flexible resources, like containers, which can be quickly and easily scaled according to the current workload needs. Automating this process is a key aspect for the management of modern web software systems, and there is a plethora of methods to implement autonomic management systems. In this work, we review three of these methods, a threshold-based approach, a control-based approach and a model-based approach. We design and run a number of experiments for all three systems with different workloads to evaluate their ability to manage the software system and how well they do so. Our experiments were conducted on the Amazon EC2 cloud with Docker containers.**

***Keywords*-self-adaptive systems; cloud computing; containers; control theory; performance models;**

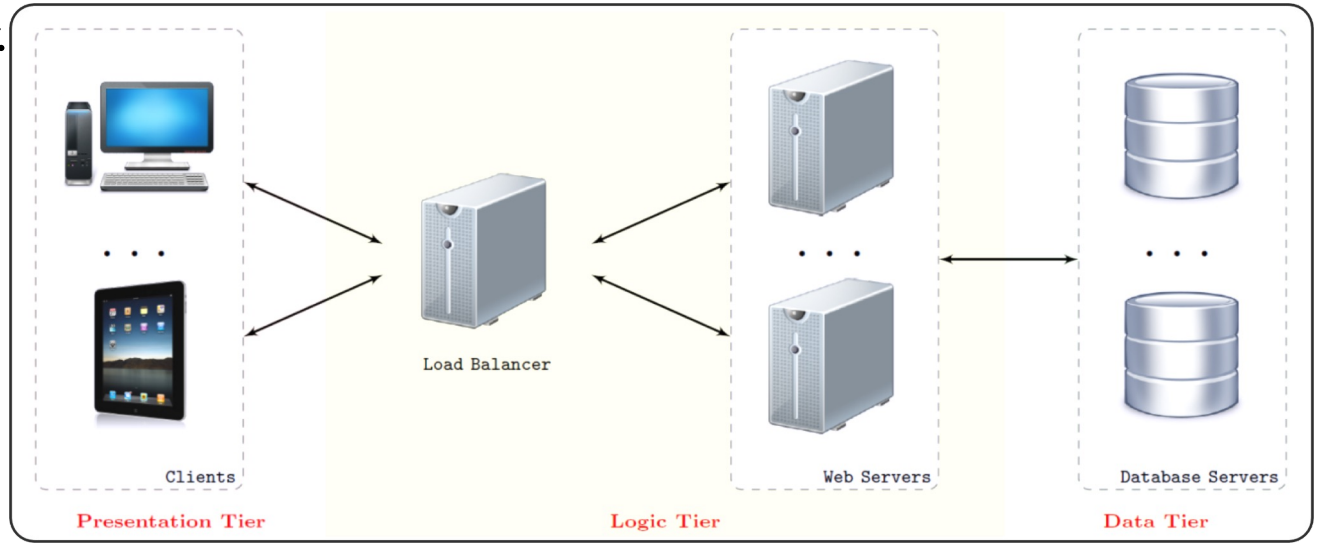
and eventually applies the corrective actions. Different AMS implementations and adaptation methods may focus on different modules from the MAPE-K architecture, usually on the analysis and planning as the monitoring and execution modules are third-party tools available by the cloud or the software provider.

In our work, we consider three types of AMS:

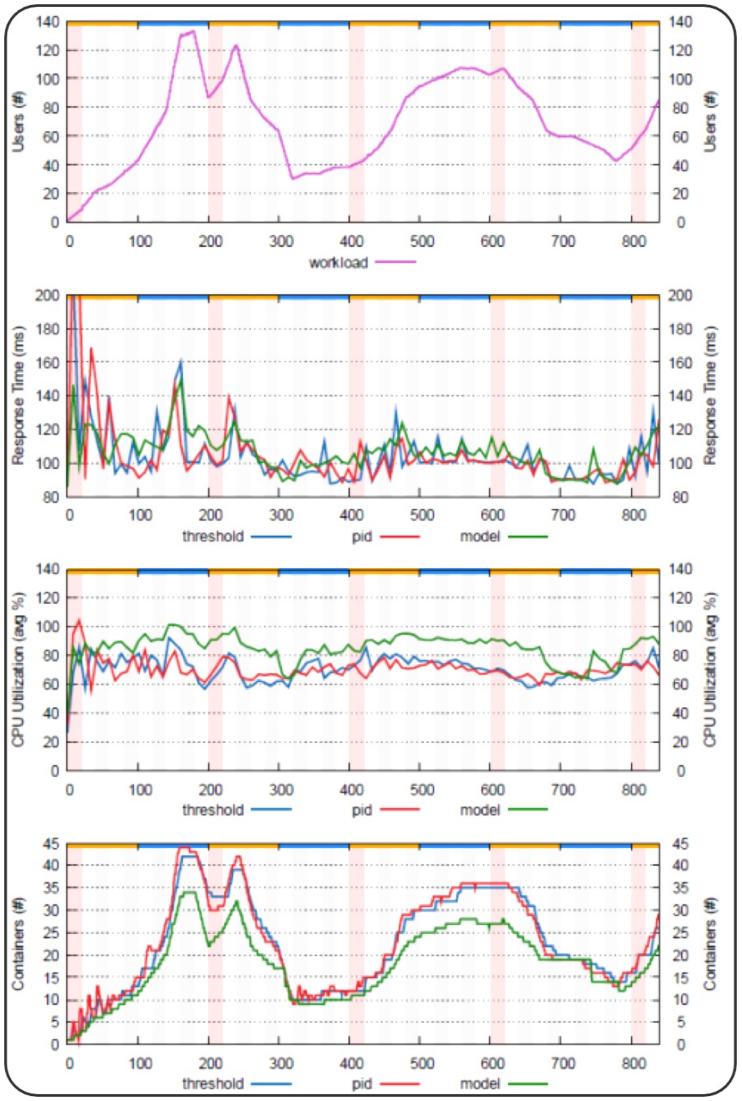
1) A **threshold-based** AMS implemented using predefined thresholds on observed metrics and predefined corrective actions. This is one of the most popular methods currently in practice, mainly thanks to its simplicity, its general applicability and its basic efficiency. In practice, the developer defines *a priori* thresholds on performance metrics. If current measurements deviate from these thresholds, it is an indication for a problem. As a response, the AMS takes a predefined

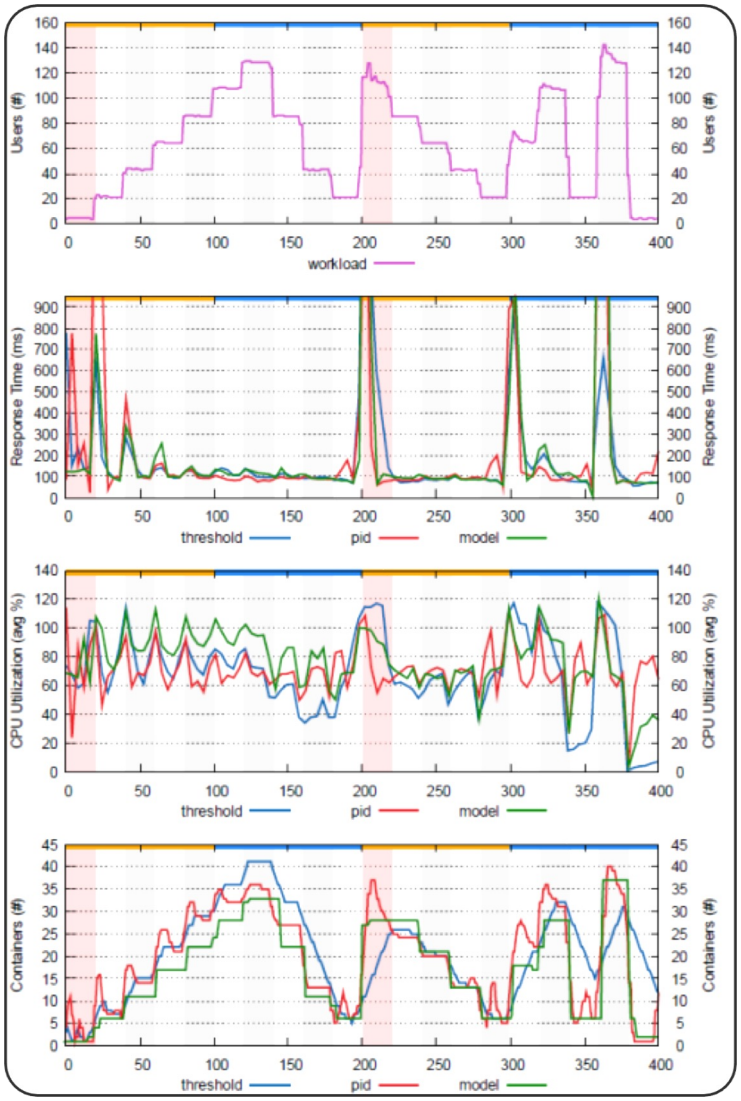
Context

- An application of three levels.
- The goal is to manage the performance of the functional level by applying Docker container scaling.
- Three management systems have been designed.
- One that uses rules according to thresholds of performance indices (CPU, response time).
- One that uses a performance model to estimate the impact of workload and available infrastructure.
- A PID controller.



Results


Users (#)
workload
Response Time (ms)
threshold
pid
model
CPU Utilization (avg %)
Containers (#)


Users (#)
workload
Response Time (ms)
threshold
pid
model
CPU Utilization (avg %)
Containers (#)

Results: settings

Method	CPU (moy)	RT (moy)	MAPE	Rise time	Overshoot	Settling time
Rule	67.98	184.25	22.24	8.7	13.61	14
PID	70.73	211.17	8.95	3.9	18.89	9.85
Model	77.72	222.86	14.71	9.25	17.82	13.7

References

- Li, Heng, et al. "**Adopting Autonomic Computing Capabilities in Existing Large-Scale Systems**." *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. IEEE, 2018.
- Calheiros, Rodrigo N., et al. "**Workload prediction using ARIMA model and its impact on cloud applications' QoS**." *IEEE transactions on cloud computing* 3.4 (2014): 449-458.
- Barna, Cornel, et al. "**Delivering elastic containerized cloud applications to enable DevOps**." *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2017.
- Fokaefs, Marios, et al. "**Evaluating adaptation methods for cloud applications: An empirical study**." *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE, 2017.