Designing an Interactive Dashboard for Automated Cloud Resource Management

Sana Malik
Adobe Research
San Jose, CA 95113, USA
sana.malik@adobe.com

Kanak Mahadik
Adobe Research
San Jose, CA 95113, USA
mahadik@adobe.com

Abstract

Many applications today are deployed on the cloud and require their owners to decide how many resources (*e.g.,* CPUs, memory) to allocate, known as provisioning. Due to the complexities of understanding variations in load, many applications are either under-provisioned or over-provisioned. We developed an automated resource configuration recommender system but found that application owners were hesitant to trust an automated system that may impact their applications' performance. Towards increasing trust, we built an interactive dashboard that allowed them to understand their resource usage, review the automated system's recommendation, and control when the recommendations are applied. We iteratively designed and piloted the system with owners of twenty cloud applications and discuss seven design needs for designing dashboards for automated resource management systems.

Author Keywords

automation; resource allocation; dashboard

CCS Concepts

•Human-centered computing → Interactive systems and tools;

Workshop proceedings *Automation Experience across Domains*
In conjunction with CHI'20, April 26th, 2020, Honolulu, HI, USA

Website: http://everyday-automation.tech-experience.at

Introduction

Performance of an application executing in the public cloud invariably depends on its provisioned resources. Under-provisioning can result in performance degradation and costly application-level agreement (SLA) violations, while over-provisioning leads to low resource utilization and wasted money. Designing cloud applications such that they can deliver on resource efficiency without performance degradation is key to their success. However, deciding these resource requirements is not straightforward for application owners. Cloud applications undergo striking variations in load and application owners don't always have tools to understand how their resources are being used over time.

To alleviate these pains, we developed an automated resource configuration recommendation algorithm that provides recommendations for right-sized resource provisioning. However, during piloting, we found that application owners were hesitant to make changes to their applications' configuration due to a lack of understanding of how their allocated resources are being used and distrust that the automated recommendations would not degrade their applications' performance. Based on previous literature in the trust and automation domain, we decided to develop an interactive dashboard for application owners to build trust in the recommendations.

In this paper, we (1) distill seven design needs for an interactive dashboard for application owners through iterative prototyping and expert reviews, and (2) present our final system for automated cloud resource management(`AutoCRM`).

Background

Resource Management

Previous work to aid right-provisioning falls into *reactive* and *predictive* approaches. Reactive approaches, such as autoscaling with predefined heuristics, are often used to adapt to changes in load [7, 4]. However, these heuristics are difficult to tune and can lead to poor quality-of-application (QoS) if the change in resource demands is quicker than the reconfiguration time. While predictive approaches, such as artificial neural networks and reinforcement learning [7] ease QoS issues, they are only known to capture simpler workload behavior or are not scalable in production environments, respectively. Our approach is predictive but uses a closed-loop approach that not only predicts usage and models the scaling behavior over time to generate configuration parameters.

Trust and Automation

Much work has been done surrounding trust in automated systems, including determining trust [8], building trust [9], and understanding the role of trust [2, 3]. Carlson et al. [2] surveyed factors for trust in the autonomous vehicle and medical domains and provide guidelines for building trust in systems. The authors found similarities and differences in important factors between these two domains, such as the ability to stay up-to-date, past performance, and verification, which we use as a starting point for our system.

Understanding Application Owners' Needs

Method

The initial recommender was piloted with six teams owning twenty applications total. Each team saw CPU and memory line charts showing allocated resources versus actual usage, as well as the recommended allocation over time. Based on feedback, we iteratively developed a prototype in close collaboration with engineering teams and project managers who were responsible for collecting resource usage, identifying pilot teams, and overall coordination. The development phase took about six months. We then synthesized the feedback into seven design needs.

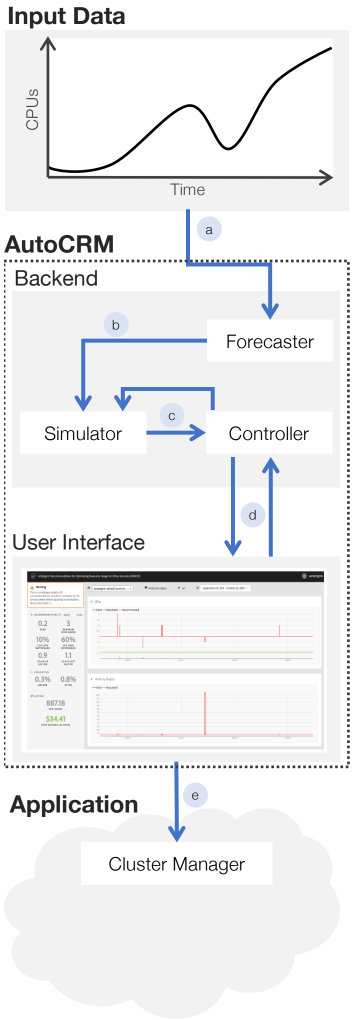


Figure 1: AutoCRM consists of two main components: a backend and user interface. The system takes historical CPU usage data as input and outputs a recommended configuration to the application.

Design Needs for Application Owners

(1) Add automation in stages. Because no similar system existed in a previously fully manual process, it was a large shift in application owners' workflows which resulted in a lack of trust. Developing the system slowly in stages ensured accuracy of the tool and increased users' confidence. For example, though possible, a fully automated version of the system was not immediately deployed. Instead, a mixed-initiative approach where users can manually apply recommendations is used.

(2) Perform formative analysis to model user needs. Because users were primarily concerned with not interrupting application performance, it was essential to be conscious about the recommendations and provide appropriate error margins, since the real future demands of the application are unknown [5]. For example, instead of recommending the optimal configurations, the recommender skews towards slightly lower utilization to balance users' expectations and comfort levels.

(3) Provide accountability for the system. Users needed to easily monitor system behavior, so we provide logging for every change that the system makes and methods for users to override changes (undo) made by the system.

(4) Be aware of coupling between data availability and system automation. Because of the need for accountability, we limited the systems' automation based on the availability of data. For example, there is a one-day delay for usage data, so changes are not made more than once a day so that users can verify what the system is doing in real-time. The system is not able to make multiple changes without the application owner reviewing them, and this can be artificially limited when necessary.

(5) Increase explainability where possible. Because application owners were unable to understand their resource needs, they were hesitant to lower provisioning and compromise application performance. The charts allowed users to see gaps between their allocations and actual utilization.

(6) Build trust through simulation. Similarly, users needed to see the expected benefits clearly and the simulator allows the users to preview the provisioning under the system's recommendations. Additionally, the forecaster allows them to see future usage, and the simulator forecasts the allocations to assure the user that the recommended configurations will not be under-provisioned.

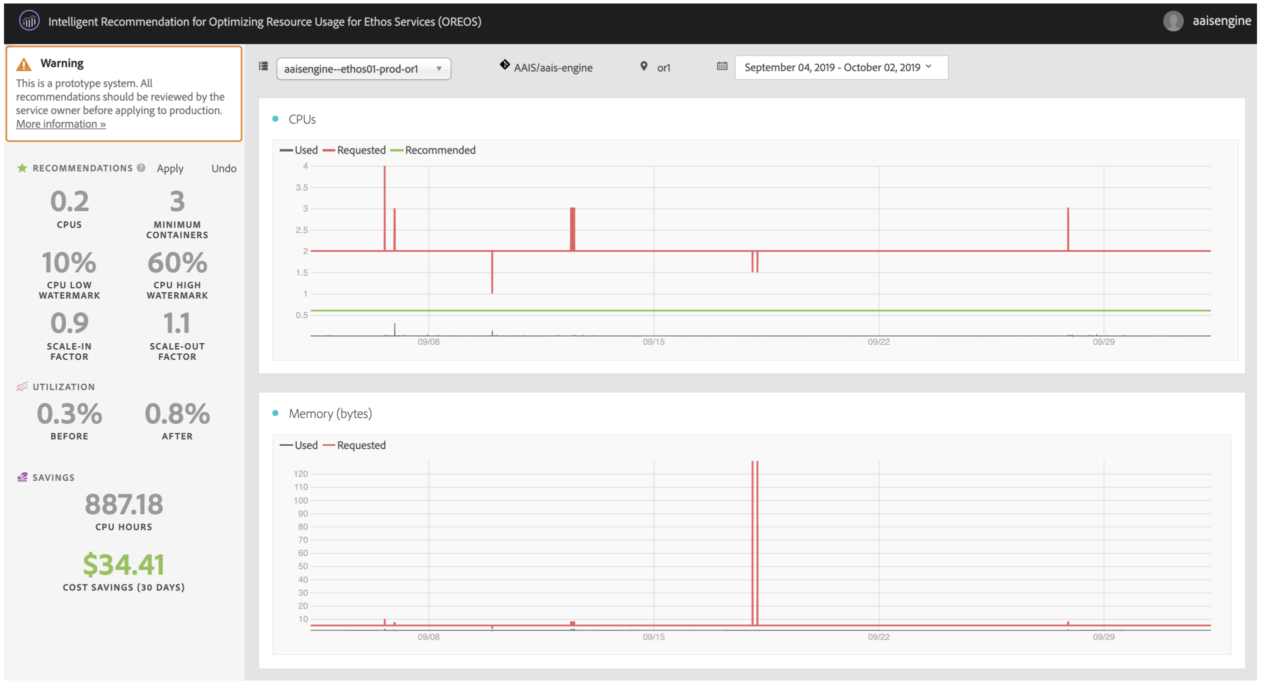
(7) Provide appropriate user controls In its first iteration, the algorithm recommended only the number of containers based on the total CPUs as selected by the user in the sidebar. The CPU selection was presented as a slider, so users could experiment with different CPU amounts and understand how it affected utilization and cost, thinking it would allow users to interact with the recommender and increase trust. However, most application owners relied on the pre-configured number of CPUs and did not experiment. The next iteration removed the slider and instead showed the projected utilization for the 5 nearest configurations, however, this was crowded and confusing to users. Hence, we refined the controller and simulator algorithms to directly generate optimal CPU values which could be visualized.

Description of System

Here we describe the final version of the system in two parts: (1) the recommender backend and (2) the dashboard UI.

Recommender Backend

The main components of AutoCRM (Fig. 1) are an ARIMA-enabled forecaster, a simulator, and a controller that em-



Warning
This is a prototype system. All recommendations should be reviewed by the service owner before applying to production.
More information »

aaisengine--ethos01-prod-or1 | AAIS/aais-engine | or1 | September 04, 2019 - October 02, 2019

RECOMMENDATIONS Apply Undo

0.2 CPUS
3 MINIMUM CONTAINERS
10% CPU LOW WATERMARK
60% CPU HIGH WATERMARK
0.9 SCALE-IN FACTOR
1.1 SCALE-OUT FACTOR

UTILIZATION
0.3% BEFORE
0.8% AFTER

SAVINGS
887.18 CPU HOURS
$34.41 COST SAVINGS (30 DAYS)

Figure 2 (cont'd): The sidebar displays the recommendations with cost and efficiency benefits. The line charts display historically allocated (red) and used (grey) CPUs and memory, so users can understand their applications' resource needs. The simulated CPU allocation based on the recommended configuration is also shown on the CPU chart (green).

Figure 2: The UI was designed to give an overview of the applications' recommended configuration, historical usage, and simulated resources.

ploys a carefully designed optimization function to arrive at efficient application resource sizing values.

Forecaster The Forecaster predicts future usage of an application applying the ARIMA model [1] to the input data (Fig 1a), chosen for its lower error in prediction than other time-series models. The Forecaster employs the Hyndman-Khandakar algorithm [6] to implement the fitting process.

Simulator The simulator models resource resizing over time using the forecasted usage data (Fig 1b) and computes the cost function value for a configuration and interval specified by the controller (Fig 1c).

Controller The controller receives the predicted resource usage values (Fig 1c) and compares two sets of resource configurations for their utility in terms of utilization and overheads using a cost function. The cost function is a weighted sum of resource wastage and overheads (number of scaling events) for an application. The configuration that minimizes the cost function is recommended.

User Interface
The UI (Fig. 2) was designed to display the recommendations with cost and efficiency benefits and help application owners understand and compare their prior resource utilization against the simulated utilization.

Recommendation Display The sidebar displays the recommended configurations and projected utilization and cost savings. We provide both "Before" and "After" measurements so application owners can directly gauge any cost- and resource-saving benefits.

User Controls Many application owners have multiple applications, so a dropdown menu allows them to choose the application of interest. Its repository name and region are shown to the right. The date picker allows users to choose from pre-defined date ranges (last week, last month, last three months, last six months) or select a custom date range. Most importantly, users can manually Apply recommendations or Undo the last application.

Utilization Charts As discussed, the most important aspect was for users to examine their utilization history. Thus, we provide CPU and memory charts that display historical allocated versus actual usage for each resource. Next, it was important for users to see how the resources would scale given their utilization for a particular period, so in the CPU chart, the simulator results are also shown, which made users more comfortable in reducing allocation without sacrificing performance or uptime. Users can pan and zoom on the charts to inspect the data closer.

Conclusion

In this paper, we present design needs and a system for automated resource management. We developed a backend recommender and an accompanying dashboard UI to increase trust in the UI and share our most important lessons learned that can benefit the community. Future work includes deploying the system to the company at large and evaluating the adoption rates for the recommendations, as well as fully automating the system in the long-term.

Acknowledgements

We would like to thank Israel Derdik, Travis Borovatz, and Chandler Allphin for their valuable feedback and support during designing, deploying, and evaluating the system.

REFERENCES

[1] George E.P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. 2015. *Time series analysis: forecasting and control*.

[2] M.S. Carlson, M. Desai, J.L. Drury, H. Kwak, and H.A. Yanco. 2014. Identifying factors that influence trust in automated cars and medical diagnosis systems. *AAAI Spring Symp. - Tech. Report* (01 2014), 20–27.

[3] Sylvain Daronnat, Leif Azzopardi, Martin Halvey, and Mateusz Dubiel. 2019. Human-agent collaborations: trust in negotiating control. *CHI 2019* (2019).

[4] Xavier Dutreilh, Aurélien Moreau, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. 2010. From data center resource allocation to control theory and back. In *IEEE Intl. Conf. on Cloud Computing*. 410–417.

[5] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. 2010. Press: Predictive elastic resource scaling for cloud systems. In *Intl. Conf. on Network and Service Management*. 9–16.

[6] Rob J. Hyndman, Yeasmin Khandakar, and others. 2007. *Automatic time series for forecasting: the forecast package for R*. Number 6/07. Monash University.

[7] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A Lozano. 2014. A review of auto-scaling techniques for elastic applications in cloud environments. *J. of grid computing* 12, 4 (2014), 559–592.

[8] Alexander G. Mirnig, Philipp Wintersberger, Christine Sutter, and Jürgen Ziegler. A Framework for Analyzing and Calibrating Trust in Automated Vehicles. In *Adjunct Proc. of the Intl. Conf. on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI '16 Adjunct)*. 33–38.

[9] Holly A. Yanco, Munjal Desai, Jill L. Drury, and Aaron Steinfeld. 2016. Methods for Developing Trust Models for Intelligent Systems.