DOI:10.1145/3364684

Exploring the opportunities to use ML, the possible designs, and our experience with Microsoft Azure.

BY RICARDO BIANCHINI, MARCUS FONTOURA, ELI CORTEZ, ANAND BONDE, ALEXANDRE MUZIO, ANA-MARIA CONSTANTIN, THOMAS MOSCIBRODA, GABRIEL MAGALHAES, GIRISH BABLANI, AND MARK RUSSINOVICH

Toward ML-Centric Cloud Platforms

CLOUD PLATFORMS, SUCH as Microsoft Azure, Amazon Web Services (AWS), and Google Cloud Platform, are tremendously complex. For example, the Azure Compute fabric governs all the physical and virtualized resources running in Microsoft's datacenters. Its main resource management systems include virtual machine (VM) and container (hereafter we refer to VMs and containers simply as "containers") scheduling, server and container health monitoring and repairs, power and energy management, and other management functions.

Cloud platforms are also extremely expensive to build and operate, so providers have a strong incentive to optimize their use. A nascent approach is to leverage machine learning (ML) in the platforms' resource management using supervised learning techniques, such as gradient-boosted trees and neural networks, or reinforcement learning. We also discuss why ML is often preferable to traditional non-ML techniques.

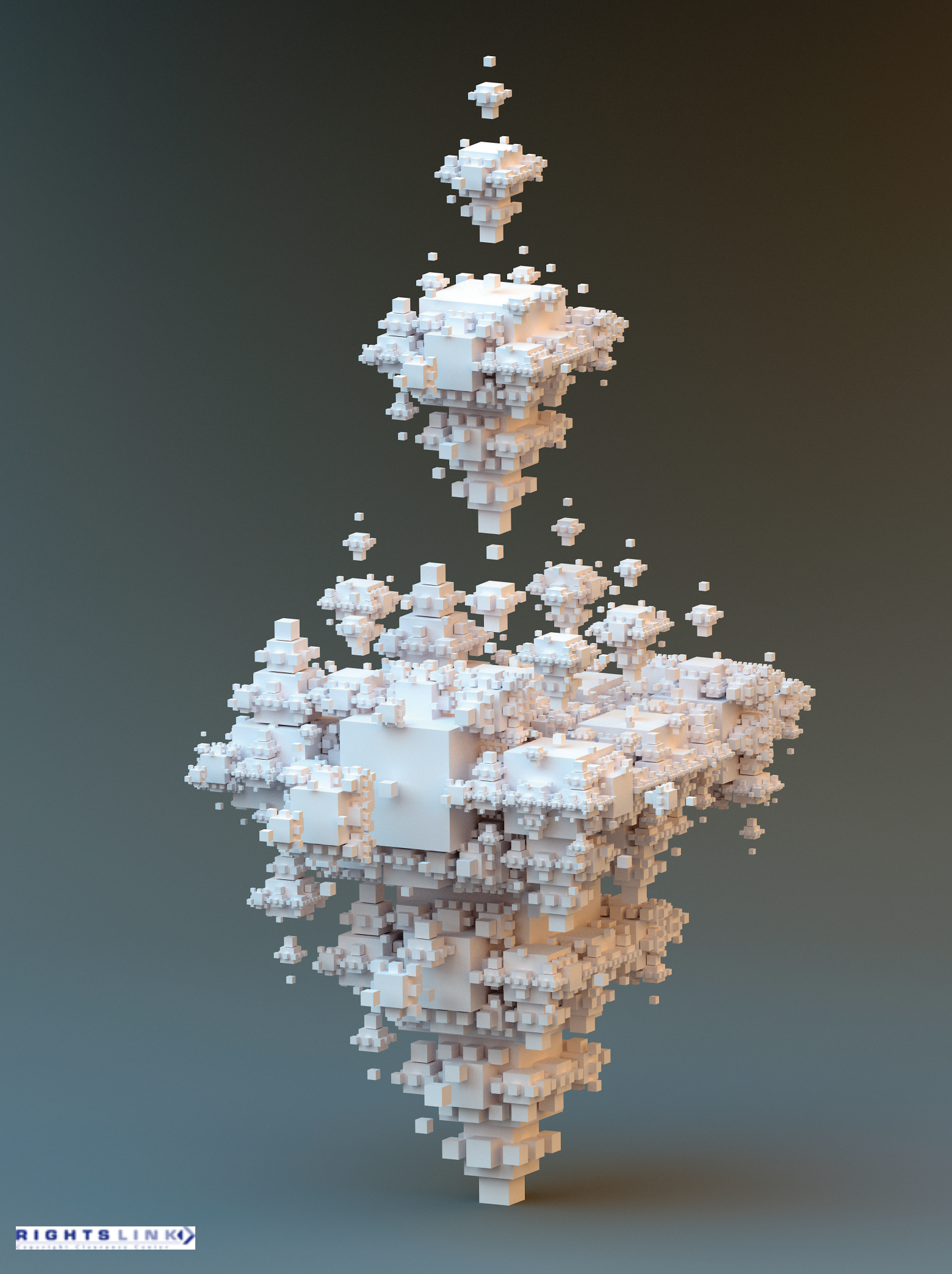
Public cloud providers are starting to explore ML-based resource management in production.[9,14] For example, Google uses neural networks to optimize fan speeds and other energy knobs.[14] In academia, researchers have proposed using collaborative filtering—a common technique in recommender systems—in scheduling containers for reduced with in-server performance interference.[12] Others proposed using reinforcement learning to adjust the resources allocated to co-located VMs.[24] Later, we discuss other opportunities for ML-based management.

Despite these prior efforts and opportunities, it is currently unclear how best to integrate ML into cloud resource management. In fact, prior approaches differ in multiple dimensions. For example, in some cases, the ML technique produces insights/predictions about the workload or infrastructure; in others, it produces actual resource management actions. In some cases, the ML is deeply integrated with the resource manager; in others, it is completely separate. In all cases, the ML addresses a single management problem; a different problem requires

» key insights

- **There are many potential uses of ML in cloud computing platforms. The challenge is in defining exactly how and where ML should be infused in these platforms.**
- **Leveraging ML-derived predictions has shown promise for many resource managers in Azure Compute. Having a general and independent ML framework/system has been key to increasing adoption quickly.**
- **Many research challenges remain open, including how to make action-prescribing ML general enough for wide applicability in cloud platforms, how to manage (potentially partial) feedback at scale, and how to debug misbehaviors (especially when the ML is tightly integrated with resource managers).**

IMAGE BY MARCEL CLEMENS



another approach. We discuss these dimensions, the possible integration designs, and their architectural, functional, and API implications.

As one point in this multi-dimensional space, we built Resource Central (RC)[9]—a general ML and prediction-serving system for providing workload and infrastructure insights to resource managers in the Azure Compute fabric. RC collects telemetry from containers and servers, learns from their prior behaviors and, when requested, produces predictions of their future behaviors. We are currently using RC to accurately predict many characteristics of the Azure Compute workload. We present an overview RC, its initial uses and results, and describe the lessons from building it.

Though RC has been successful so far, it has limitations. For example, it does not implement certain forms of interaction with resource managers. More broadly, the integration of ML into real cloud platforms in a general, maintainable, and at-scale manner is still in its infancy. We close the article with some open questions and challenges.

Cloud platforms are extremely expensive to build and operate, so providers have a strong incentive to optimize their use.

ML vs. Traditional Techniques

Resource management in cloud platforms is often implemented by static policies that have two shortcomings. First, they are tuned offline based on relatively few benchmark workloads. For example, threshold-based policies typically involve hand-tuned thresholds that must be used for widely different workloads. In contrast, ML-informed dynamic policies can naturally adapt to actual production workloads.[20,26] For the same example, each server can learn different thresholds for its own resource management.

Second, the static policies tend to require reactive actions, and may incur unnecessary overheads and customer impact. As an example, consider a common policy for scheduling containers onto servers, such as best fit. It may cause some co-located containers to interfere in their use of resources (for example, shared cache space) and require (reactive) live migrations.[23] Live migration is expensive and may cause a period of unavailability (aka "blackout" time). In contrast, ML techniques enable predictive management: having accurate predictions of container interference enables the scheduler to select placements that do not require migrations.[12] In the Resource Central section, we mention our earlier results on the benefit of predictive container scheduling. In fact, non-predictive policies (for example, based on feedback control) are not even acceptable in some cases. For instance, blindly live migrating containers that will incur long blackout times is certain to annoy customers.

ML has been shown to produce more accurate predictions for cloud resource management than more traditional methods, such as regressions or time-series analysis. For example, Cao[6] and Chen[8] demonstrate that ML techniques produce more accurate resource utilization predictions than time-series models. Our results quantitatively compare some ML and non-ML methods.

Opportunities for ML in Cloud Platforms

Cloud platforms involve a variety of resource managers, such as the container scheduler and the server health management system. Here, we discuss some of the ways in which managers can benefit from ML.

Container scheduler. The scheduler selects the server on which a container will run. It can use ML to identify (and avoid) container placements that would lead to performance interference, or to adjust its configuration parameters (for example, how tightly to pack containers on each server). It can also use ML-derived predictions of the containers' resource utilizations to balance the disk access load, or to reduce the likelihood of physical resource exhaustion in oversubscribed servers. Predictions of server health are also useful for it to stop assigning containers to servers that are likely to fail soon. Finally, it can use predictions of container lifetime when considering servers that will undergo planned maintenance or software updates. We have used lifetime predictions to match batch workloads to latency-sensitive services with enough idle capacity for the container.[27]

Server defragmenter/migration manager. As containers arrive/complete, each server may be left with available resources that are insufficient for large containers. As a result, the server defragmentation system may decide to

live-migrate active containers onto a subset of the servers. A migration manager can also live-migrate (for example, low-priority) containers to alleviate any unexpected server resource contention or interference. It can use application-level performance information (when it is available) or ML techniques on lower-level performance counters to identify these behaviors. The manager can use predictions of the containers' expected lifetimes and blackout times to live-migrate only those that will likely remain active for a substantial amount of time and not incur a noticeable blackout time if migrated.

Power capping manager. This manager ensures the capacity of the (oversubscribed) power delivery system is not exceeded, using CPU speed scaling. To tackle a power emergency (the power draw is about to exceed a circuit breaker limit), this manager can use predictions of the performance impact of speed scaling on different workloads to guide its apportioning of the available power budget. Similarly, it can use predictions of workload interactivity as a guide. Ideally, containers executing interactive or highly sensitive workloads should receive all the power they want, to the detriment of containers running batch and background tasks. In this context, the container scheduler can use predictions of interactivity to smartly schedule interactive and delay-insensitive workloads across servers.

Server health manager. This manager monitors hardware health and takes faulty servers out of rotation for maintenance. When a server starts to misbehave, this manager can use predictions of the lifetime of the containers running on the server. Using these predictions, it can determine when maintenance can be scheduled, and whether containers need to be live-migrated to prevent unavailability.

This is only a partial list of opportunities for ML-based resource management. The challenge is determining the best system designs for exploiting these opportunities.

Potential Designs for ML-Centric Clouds

When deciding how to exploit ML in cloud resource management, we must consider: The ML techniques and their inputs and outputs, and the managers and their mechanisms (management actions) and policies. We must also consider many questions: Can we use application-level performance data for learning? How should the ML and the managers interact? Should the ML produce behavioral insights/predictions or actual management actions? How tightly integrated with the managers should the ML be? How quickly does the ML need to observe the effect of the management actions? Is it possible to create general frameworks/APIs that can apply to many types of resource management? Next, we discuss our thoughts along these dimensions.

Application performance vs. counters. Managers must optimize resource usage without noticeably hurting end-to-end application performance. Thus, having direct data on application performance enables precise management with or without ML. Some application metrics are easier to obtain than others. For example, VM lifetimes are "visible" to the platform, whereas request latencies within VMs implementing a service often are not. When containers are opaque to the platform, the way to obtain application performance data is for developers to instrument their codes with monitoring calls into the platform (for example, using AWS's CloudWatch[3] or Azure's Monitor[21]). When they do not, lower-level counters (for example, resource utilization, CPU performance counters) must be used as an imperfect proxy for application performance. Given that most workloads are not instrumented, we expect that ML techniques will most often use counters. Nevertheless, providers also run first-party workloads, which can potentially be instrumented.

Predictions vs. actions. Another dimension concerns the role of the ML techniques. One approach is for them to produce insights (for example, performance, load, container lifetime predictions) that managers can leverage to improve decisions as in Figure 1 (top). This approach gives managers sole control and understanding of the management policies. Another approach is for the ML to produce actual management actions (for example, migrate this container, change this resource allocation) to be taken by managers. In this case, the ML embodies a deeper understanding of the policies (or may itself define the policies). Targeting the ML at producing actions may lead to policies that more easily adapt to the actual

Figure 1. Two designs.

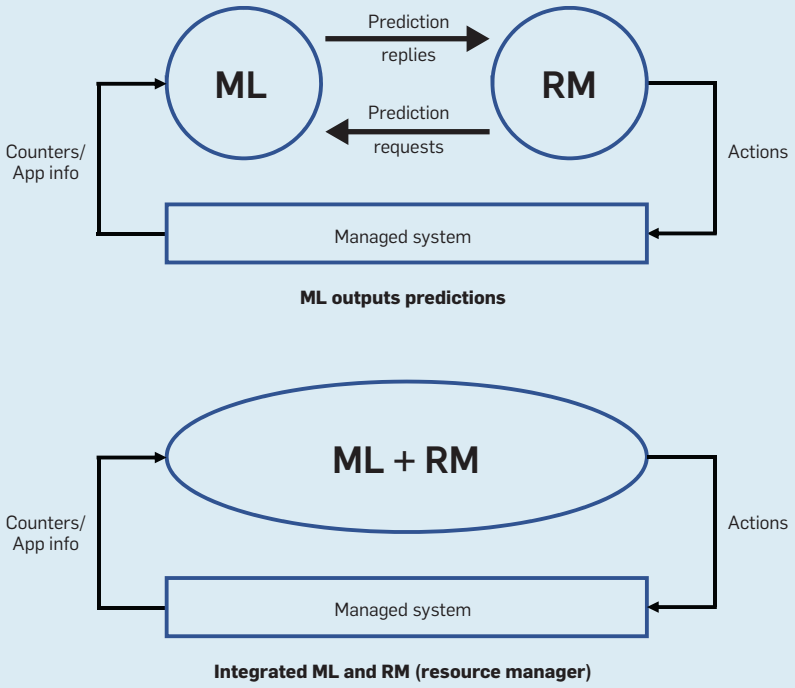
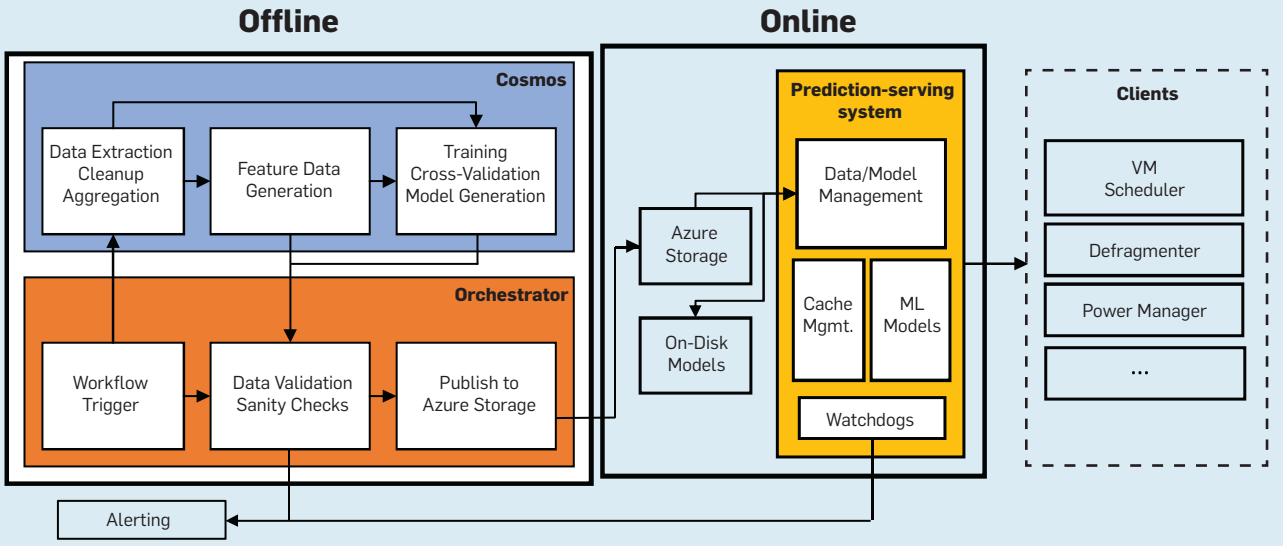


Figure 2. RC architecture comprising offline and online components.



workload and infrastructure behaviors, whereas leaving this responsibility to managers may produce policies that are unnecessarily general. Producing actions may also be the only alternative when it is impractical to collect labeled training data (for example, when fast management decisions must be made at the servers themselves, based on fine-grained performance data). On the other hand, leveraging ML for insights simplifies the managers, making them easier to understand and debug. In fact, relying on insights is less likely to cause negative feedback loops that could potentially degrade customer experience. Insights may also inform multiple managers (for example, container scheduler and power manager), whereas actions are manager-specific.

Integration vs. separation. Related to the dimension here is the question of whether the ML should be fully integrated or completely separate from managers. When the ML outputs actions, the fully integrated design is a natural one, as in Figure 1 (bottom). For insights, both integration and separation are viable options. However, for generality and maintainability, cleanly separating the ML and the managers via well-defined APIs is beneficial: multiple managers can use the same ML implementation, which the platform can maintain independently of the managers.

Immediate vs. delayed feedback. A final dimension is whether the ML is able to observe the result of its previous outputs or the manager actions within a short time. Designs that produce ML models offline will likely observe these effects only at a coarse time granularity (for example, daily). Such granularity is a good match when the input feature characteristics also change slowly. However, techniques such as reinforcement learning and bandit learning often benefit from actions being observable much sooner. For such techniques, offline model-learning may not be ideal.

Table 1. Behavior, ML modeling approaches, model and full feature dataset sizes.

Behavior	Approach	#features	Model size	Feature data size
Avg CPU utilization	Gradient Boosting Tree	247	414KB	416MB
Deployment size	Gradient Boosting Tree	41	351KB	296MB
Lifetime	Gradient Boosting Tree	247	438KB	416MB
Blackout time	Gradient Boosting Tree	998	290KB	4.5MB

Table 2. Behaviors and their buckets.

Behavior	Bucket 1	Bucket 2	Bucket 3	Bucket 4
Avg CPU utilization	0–25%	25%–50%	50%–75%	75%–100%
Deployment size	1	>1 and ≤10	>10 and ≤100	>100
Lifetime	≤15 mins	>15 and ≤60 mins	>1 and ≤24 hs	>24 hs
Blackout time	≤0.1 s	>0.1 and ≤1 s	>1 and ≤3 s	>3 s

Resource Central

RC is one point in this multidimensional space. We built it as a general ML and prediction-serving system into the Azure Compute fabric. RC[9] learns from low-level counters from all containers and servers, produces various behavioral models offline, and provides predictions online to multiple managers via a simple REST API.

RC leverages a wealth of historical data to produce accurate predictions. For example, from the perspective of each Azure subscription, many containers exhibit peak CPU utilizations in consistent ranges across executions; containers that execute user-facing workloads consistently do so across executions; tenant deployment sizes are unlikely to vary widely across executions, and so on.[9] In all these cases, prior behavior is a good predictor for future behavior.

RC uses customer, container, and/or server features to identify correlations that managers can leverage in their decision-making. The managers query RC with a subset of the features, expecting to receive predictions for the others. For example, the scheduler may query RC while providing the customer name and type, deployment type and time, and container role name. RC will then predict how large the deployment by this customer may become and how high these containers' resource utilization may get over time.

There are many potential uses and designs for ML in cloud platforms.

Architecture. *Design rationale.* Our design for RC follows several basic principles related to the dimensions we discussed previously and our ability to operate, maintain, and extend it at scale:

1. Since application-level performance data is rarely available, RC should learn from low-level counters.
2. For generality, modularity, and debuggability, RC should be oblivious to the management policies and, instead, provide workload and infrastructure behavior predictions. It should also provide an API that is general enough for many managers to use.
3. For performance and availability, RC should be an independent system that is off the critical performance and availability paths of the managers that use it whenever possible.
4. Since workload characteristics and server behaviors change slowly, RC can learn offline and serve predictions online. For availability, these two components should be able to operate independently of each other.
5. For maintainability, it should be simple and rely on any existing well-supported infrastructures.
6. For usability, it should require minimal modifications to the resource managers.

Design. Figure 2 illustrates how we designed RC based on these principles. The offline workflow consists of data extraction, cleanup, aggregation, feature data generation, training, validation, and ML model generation. RC does these tasks on Cosmos,[7] a massive data processing system that collects all the container and server telemetry from the fabric. RC orchestrates these phases, sanity-checks the models and feature data, and publishes them to Azure storage, a highly available store. RC currently retrains models once a day.

The online part of RC is a REST (Representational State Transfer) service within which the models execute to produce predictions. RC's clients (for example, the container scheduler) call the service passing as input the model name and information about the container(s) for which they want predictions, for example, the subscription identification. The model may require historical feature data as additional inputs, which RC fetches from Azure Storage. As an example of feature data, the lifetime model requires information on historical lifetimes (for example, percentage of short-lived and long-lived containers to date) for the same subscription from the store. Each prediction result is a predicted value and a score. The score reflects the model's confidence in the predicted value. The client may choose to ignore a prediction when the score is too low. It may also ignore (or not wait for) a prediction if it thinks that RC is misbehaving (or unavailable).

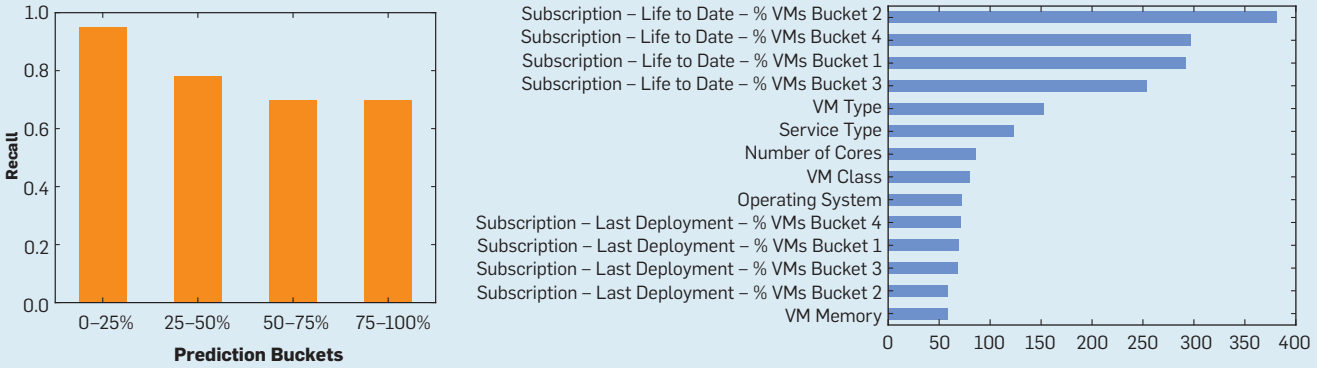
RC relies heavily on caching, as clients may have stringent performance requirements. It caches the prediction results, model, and feature data from the store in memory.

Current ML models. RC acts as a framework for offline training of ML models and serving predictions from them online; RC is agnostic to the specific modeling approach data analysts select. In our current implementation, analysts can select models from a large repository that runs on Cosmos. The three leftmost columns of Table 1 list some of the container behaviors we predict and the modeling approach we currently use: Gradient Boosting Trees (GBTs).[18] We are also experimenting with deep neural networks and plan to start using them in the next version of RC.

For classifying numeric behaviors, we divide the space of possible values into buckets (for example, 0%–24%, 25%–49%, and so on) and then predict a bucket. (As we will discuss, this approach has been more accurate for our datasets than using regression and then bucketizing the result.) When the prediction must be converted to a number, the client can assume the highest, middle, or lowest value for the predicted bucket.

Feature engineering. Each model takes many features as input, which we

Figure 3. Average CPU utilization recall per bucket (left) and attribute importance (right).



extract from the attributes available in our dataset. We split the attributes into three groups: categorical, boolean, and numerical. We model categorical attributes (for example, container type, guest operating system) as categorical features. We represent the features in a vector of pre-defined length. We concatenate the boolean attribute (for example, first deployment, production workload) values to the input feature vector. Similarly, we normalize and concatenate the numerical attribute (for example, number of cores, container memory size) values to the vector. Finally, we place the attributes that describe observed container/subscription behavior (for example, last observed container lifetime) into the buckets of Table 2 and use them as numerical features. We concatenate these features to the vector as well.

Comparison to other systems and techniques. As it focuses on producing predictions, RC fundamentally differs from action-prescribing systems, for example, Agarwal et al.,[2] and Moritz et al.[22] RC currently produces its predictions using TLC, a Microsoft-internal state-of-the-art framework that implements many learning algorithms. However, RC can also leverage recently proposed frameworks, such as TensorFlow,[1] for producing its ML models. RC's online component is comparable to recent prediction-serving systems,[10,11,16] though with a different architecture and geared toward cloud resource management. We are not aware of any ML and prediction-serving frameworks/systems like RC in other real cloud platforms.

The literature on predicting workload behaviors is extensive. These works predict resource demand, resource utilization, or job/task length for provisioning or scheduling purposes.[5,6,8,15,17,19,25] For example, Cao[6] recently explored Random Forests to predict CPU, memory, and disk utilizations, whereas Chen[8] used Residual Neural Networks for predicting VM CPU utilization. We predict a broader set of behaviors (including container lifetimes, maximum deployment sizes, and blackout times) for a broader set of purposes (including health management and power capping). Still, we do not argue that the models we use are necessarily the best. Instead, we show them simply as examples of ML models that we have integrated into the RC framework and work well in practice.

Prediction accuracy. A key requirement for RC is the ability to predict behaviors accurately. Obviously, this accuracy depends on the behavior one is trying to predict and on the modeling approach they use. As such, the best we can do is provide evidence from our experience with RC that many behaviors can be predicted accurately.

For our analysis, we use one month of data about all VMs in Azure. In this dataset, less than 1% of the VMs are from "new" subscriptions, that is, subscriptions that appear for the first time in the set. We trained RC's models with the first three weeks and tested them on the fourth. We provide a similar dataset at https://github.com/Azure/AzurePublicDataset.

We divide the space of predictions for each behavior into the buckets listed in Table 2. Given these buckets, Figure 3 summarizes the RC prediction results for each VM-utilization bucket (left) and the most important predictive attributes (right). Figure 4 shows the overall accuracy, prediction, and recall results (three rightmost bars in each group, respectively) for the VM behaviors in the tables. We measure accuracy as the percentage of predictions that were correct, assuming the predicted bucket is that with the highest confidence score; precision for a bucket as the percentage of true positives in the set of predictions that named the bucket; and recall for a bucket as the percentage of true positives in the set of predictions that should have named the bucket.

Figure 3 (left) shows recall between 70% and 95% across VM-utilization buckets. When we average bucket frequencies and recalls together, we find that overall VM-utilization recall is 89%. Figure 3 (right) shows that the most important attributes in terms of F1-score are the percentage of VMs of the same subscription that fell in each bucket to date. As we discussed in the RC paper,[9] subscriptions show low (<1) coefficient of variation (CoV = standard deviation divided by average) for the behaviors we study. Thus, it is unsurprising that prior observations of the behavior are good indicators. Still, our results show that other attributes are also important: service type (the name of a top first-party subscription or "unknown" for the others), VM type (for example, A1, A2), number of cores, VM class (IaaS vs PaaS), operating system, and deployment time; their relative importance depends on the metric. VM role names have little predictive value, for example, IaaS VMs often have arbitrary role names that do not repeat.

Figure 4 illustrates the high accu-

racy of our current GBT models, which ranges between 74% (lifetime) and 87% (blackout time). The GBT prediction quality is even higher when we discard predictions with low (< 60%) confidence: precision ranges between 84% (lifetime) and 90% (average CPU utilization and blackout time) without substantially hurting recall, which ranges between 72% (lifetime) and 99% (blackout time). Again, for all behaviors, the most important attributes are the percentage of VMs classified into each bucket to date.

We expect the prediction quality our current models provide will be enough for most clients. For example, a VM scheduler that oversubscribes CPU cores prevents resource exhaustion as effectively with RC's VM utilization predictions as with an oracle predictor.[9]

Accuracy by VM group. Interestingly, accuracy can be higher for the first VM deployments from new subscriptions than for deployments from subscriptions we have already seen in the dataset. For example, for average CPU utilization predictions, these accuracies are 92% and 81%, respectively. We conjecture that this is because users tend to experiment with their first VMs in similar ways, so feature data accounting for prior subscriptions is predictive of new ones.

We also compare the prediction accuracy for third- and first-party VMs, and for first-party production and non-production VMs. The former comparison shows that accuracy tends to be higher for third-party VMs. For example, for lifetime, the accuracies for third-party and first-party VMs are 83% and 74% respectively, whereas for average CPU utilization they are 84% and 80% respectively. When comparing production and non-production first-party VMs, the results are more mixed. For lifetime, accuracy is higher for production VMs (82% vs. 64%), whereas the opposite is true for average CPU utilization (79% vs. 83%). The wide diversity of production workloads makes utilization more difficult to predict, but at the same time their lifetimes are less diverse and easier to predict as production VM tend to live long.

Comparison to other techniques. As baselines for comparison, we experiment with three techniques: most recent bucket (MRB), most popular bucket (MPB), and logistic regression (LR). MRB and MPB are non-ML techniques. MRB predicts the bucket that was most common for the VMs in the last deployment of the same subscription (lifetime and average CPU utilization), the same bucket as the last deployment of the same subscription (max deployment size), or the same bucket as the last VM migration of a similar size (blackout time). MPB predicts the bucket that has been most popular since the start of the subscription. LR predicts a bucket based on a non-linear probability curve computed using the maximum likelihood method. We train the LR models with the same feature vectors we described earlier. Figure 4 shows that MRB exhibits accuracies between 54% and 81%, whereas MPB stays between 42% and 78%, and LR in the 62%–80% range. Clearly, these accuracies are substantially worse than our GBT results. Compared to MRB and MPB, GBT relies on many features instead of a simple heuristic, giving it a broader context that improves predictions. Compared to LR, GBT performs better for higher dimensional data. In addition, GBT combines decision trees with different parameters to produce higher quality results.

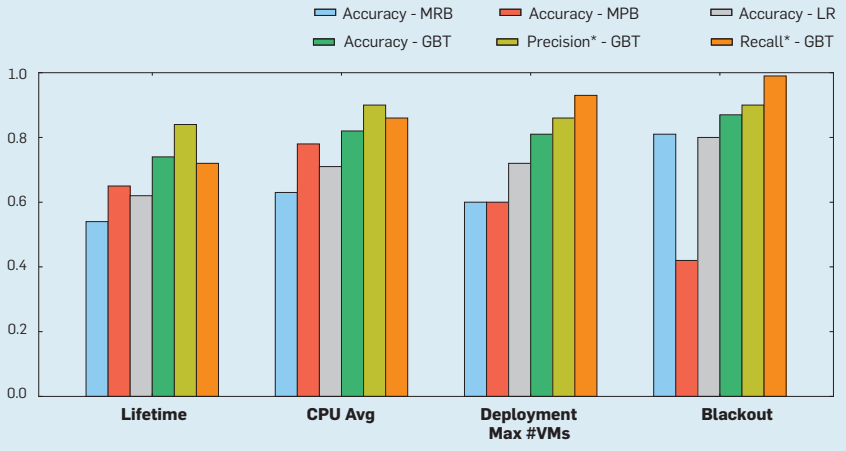
Comparison to regression into buckets. We also compare GBTs as classifiers into buckets with GBTs used for numerical regression and then bucketizing the results. We find that the former approach is substantially more accurate. The reason is that "noise" in the numerical values (for example, few VM deployments that are exceptionally large) throw off the regression models and ultimately produce incorrect buckets.

Initial uses and results in resource management. In its first production instantiation, we have implemented RC's online component as a service in each Azure Compute cluster. A single version of its offline model-generation component runs on Cosmos. The first two major clients to use RC were the server defragmenter, which queries RC for lifetime and blackout time predictions (and VM metadata); and the container scheduler, which queries it for lifetime predictions (and metadata). As of March 2019, RC's clients are directing roughly 1.5 billion queries to it daily. The next major clients we will productize are the power capping manager, which will use RC's workload interactivity predictions; and a new predictive container rightsizing system, which will use RC's utilization predictions to recommend new container sizes. Several other uses of RC are being planned.

Our production results from the server defragmenter show that, from October 2018 to March 2019, RC enabled many tens of thousands of VM migrations, enabling more than 200 clusters (that would otherwise have been considered "full") to continue receiving new VMs. Our earlier simulation study considered the use of RC-produced VM utilization predictions for safe core oversubscription.[9] It showed that an RC-informed oversubscribing VM

Figure 4. Accuracy, precision, and recall for all behaviors.

Three leftmost bars of each behavior represent the accuracy for most recent bucket (MRB), most popular bucket (MPB), and logistic regression (LR). Two rightmost bars represent precision and recall with gradient boosting tree (GBT), when predictions with <60% confidence are excluded.



scheduler can accommodate more VMs, while producing 6× fewer cases of physical resource exhaustion than a baseline oversubscribing scheduler that does not consider utilization predictions.

Lessons learned. Thinking about ML-centric cloud platforms and through our experience with RC, we learned several important lessons:

Separation of concerns. Keeping predictions and management policies separate has worked well in RC. This separation is making policies easier to debug and their results easier to reproduce, as they are not obscured by complex ML techniques. In Azure's current managers, policies tend to be rule-based and use RC predictions as rule attributes (for example, if expected lifetime is short, then place container in one of these servers). The rule-based organization has made it easier to integrate predictions into existing managers.

Reach and extensibility. The ML framework must act as a source of intelligence for many resource managers, not all of which will be known on day-one. Thus, it is critical to be able to easily integrate new data sources, predict/understand more behaviors, include multiple models for each behavior and implement versioning per model, among others. RC's modular design has made these extensions easy.

Model updates. We designed RC to produce models and feature data offline, and then serve predictions and feature data online until it produces (in the background) an updated version of them. However, one resource manager we have come across requires models to be updated online, so that each prediction accounts for the effect of the previous one. As this scenario seems to be rare, we opted for RC's more general design.

Performance. Many of the managers do not require extremely fast predictions. For example, the server defragmenter can easily deal with slow predictions. However, other managers require substantially higher performance. For example, the entire time budget for the container scheduler is less than 100 milliseconds. In these scenarios, prediction result, model, and feature data caching can be critical in prediction-serving, especially if models are large and complex. Caching also helps maintain operation even when the data store is unavailable.

Integration with clients. We explored two versions of RC's online component: a first one was implemented as a runtime library to be linked with clients, and another as an independent service (as described). Interestingly, there is still no consensus on which implementation is ideal for Azure. Some teams like the ability to get predictions without leaving the client's machine, which the library approach enables via RC's caches. Other teams prefer the standard and higher-level interface of a service, and do not want to manage an additional library. Ultimately, we expect to build multiple online component implementations that will consume models and feature data from the same back-end source.

RC is by no means the only possible approach to ML-centric platforms. In fact, RC cannot currently accommodate certain types of ML integration that can be potentially useful.

Open Challenges and Research Avenues

As should be clear by now, RC is by no means the only possible approach to ML-centric platforms. In fact, RC cannot currently accommodate certain types of ML integration that can be potentially useful. Moreover, there are potential additional areas for ML integration that nobody has explored yet. Clearly, there is a need for more research on this topic. The following paragraphs identify some research challenges and avenues going forward.

Broadly using application-level performance data. As mentioned, low-level counters are an indirect measure of application performance. For resource management without performance loss, extracting high-level information from applications is key. Today's extraction methods require effort from developers, who do not always have a strong incentive to provide the data. The challenge the cloud provider faces is creating stronger incentives or extraction methods that are automatic, privacy-preserving, and non-intrusive.

Using action-prescribing ML while being general. Increasingly popular ML techniques such as reinforcement and bandit learning prescribe actions. In the resource management context, this means the ML must understand the acceptable management mechanisms and policies (these techniques could define the policies themselves, but this would make manager debugging very difficult), and be adjusted for every manager that can benefit. Moreover, it must be safe/cheap to explore the space of available actions. The challenge is creating general designs for these ML techniques, perhaps via frameworks/APIs that take mechanism

and policy descriptions as inputs, so that they are easier to adopt.

Quick feedback at scale. When operating at scale, it is difficult to observe the result of predictions or management actions within a short time. For example, scheduling a large group of containers may impact the resource utilization and performance interference at many servers. Worse, other containers are constantly starting and finishing on these same servers. In this context, obtaining feedback from all the servers and isolating the impact of each prediction/action is extremely difficult. The problem is even harder when the feedback will only be available at an undetermined future time. For example, a container lifetime prediction can only be verified when the container finishes, which may take a long time. Thus, when quick feedback is needed, the challenges become determining when it is available, and accurately deriving it from the massive amount of collected data.

Debuggability. The cloud provider must be prepared for scenarios where an ML technique (and the managers that use it) suddenly starts to misbehave, producing poor predictions or actions. In these cases, regenerating models with more recent data may not be enough to fix the problem, because model features may have changed semantics or been eliminated altogether. Debugging misbehaviors is difficult for certain ML techniques, for example, neural networks, especially when they are tightly integrated with the manager. More research is needed on making debugging easier, if such techniques are to be used broadly in public clouds.

Other aspects of cloud platforms. Finally, our focus has been using ML in resource management for high efficiency and performance. However, ML can also benefit network traffic, availability, reliability, security, configuration management.[4,13,28] An open question is whether the ideas, designs, and trade-offs we discussed also apply to these other areas. In our future work, we will explore how to extend RC for these types management in a real platform.

Conclusion

Public cloud providers invest billions of dollars into their software and hardware infrastructures. Maximizing the use of these expensive resources, while maintaining excellent performance and availability, is critical for profitability and competitiveness. Infusing ML into cloud platforms has the potential to achieve these goals. In fact, we envision many opportunities and designs for bringing ML into cloud resource management. Taking advantage of some of these opportunities, we have been transforming the Azure Compute fabric to leverage predictions of container and server behaviors. This transformation has relied on Resource Central, our general ML and prediction-serving framework and system. Our initial experience shows that one can produce simple yet accurate ML models for many behaviors and enable resource managers to make smarter decisions. However, more research is needed on this topic, so we have made traces of the Azure Compute workload available for researchers to use at https://github.com/Azure/AzurePublicDataset. C

References

1. Abadi, M. et al. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symp. Operating System Design and Implementation* (2016).
2. Agarwal, A. et al. Making contextual decisions with low technical debt. *arXiv preprint arXiv:1606.03966* (2016).
3. Amazon Web Services. Amazon CloudWatch; https://aws.amazon.com/cloudwatch/.
4. Bodik, P., Griffith, R., Sutton, C., Fox, A., Jordan, M., and Patterson, D. Statistical machine learning makes automatic control practical for Internet datacenters. In *Proceedings of HotCloud* (2009).
5. Calheiros, R. N., Masoumi, E., Ranjan, R., and Buyya, R. Workload prediction using ARIMA model and its impact on cloud applications' QoS. *IEEE Trans. Cloud Computing 3*, 4 (2015).
6. Cao, R., Yu, Z., Marbach, T., Li, J., Wang, G., and Liu, X. Load prediction for data centers based on database service. In *Proceedings of the 42nd Annual Computer Software and Applications Conf.* (2018).
7. Chaiken, R., Jenkins, B., Larson, P., Ramsey, B., Shakib, D., Weaver, S., and Zhou, J. SCOPE: Easy and efficient parallel processing of massive data sets. In *Proceedings of the 34th Intern. Conf. Very Large Data Bases* (2008).
8. Chen, S., Shen, Y., and Zhu, Y. Modeling conceptual characteristics of virtual machines for CPU utilization prediction. In *Proceedings of the Intern. Conf. Conceptual Modeling* (2018).
9. Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., and Bianchini, R. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the Intern. Symp Operating Systems Principles* (2017).
10. Crankshaw, D. et al. The missing piece in complex analytics: Low latency, scalable model management and serving with Velox. In *Proceedings of the 7th Biennial Conf. Innovative Data Systems Research* (2015).
11. Crankshaw, D.,Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E., and Stoica, I. Clipper: A low-latency online prediction serving system. In *Proceedings of the 14th Symp. Networked Systems Design and Implementation* (2017).
12. Delimitrou, C. and Kozyrakis, C. Paragon: QoS-aware scheduling for heterogeneous datacenters. In *Proceedings of the 18th Intern. Conf. Architectural Support for Programming Languages and Operating Systems* (2013).
13. Fox, A., Kiciman, E., and Patterson, D. Combining statistical monitoring and predictable recovery for self-management. In *Proceedings of the 1st Workshop on Self-Managed Systems* (2004).
14. Gao, J. Machine Learning Applications For Datacenter Optimization, 2014.
15. Gong, Z., Gu, X., and Wilkes, J. Press: Predictive elastic resource scaling for cloud systems. In *Proceedings of the Intern. Conf. Network and Service Management* (2010).
16. Google. TensorFlow serving; http://tensorflow.github.io/serving/.
17. Islam, S., Keung, J., Lee, K., and Liu, A. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems 28*, 1 (2012).
18. Ke, G. et al. LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems 30* (2017).
19. Khan, A., Yan, X., Tao, S., and Anerousis, N. Workload characterization and prediction in the cloud: A multiple time series approach. In *Proceedings of the Intern. Conf. Network and Service Management* (2012).
20. Mao, H., Alizadeh, M., Menache, I., and Kandula, S. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (2016).
21. Microsoft Azure. Azure Monitor; https://azure.microsoft.com/en-us/services/monitor/.
22. Moritz, P. et al. Ray: A distributed framework for emerging AI applications. In *Proceedings of the 13th USENIX Symp. Operating Systems Design and Implementation* (2018).
23. Novakovic, D., Vasic, N., Novakovic, S., Kostic, D., and Bianchini, R. DeepDive: Transparently identifying and managing performance interference in virtualized environments. In *Proceedings of the USENIX Annual Technical Conf.* (2013).
24. Rao, J., Bu, X., Xu, C.-Z., Wang, L., and Yin, G. VCONF: A reinforcement learning approach to virtual machine auto-configuration. In *Proceedings of the 6th Intern. Conf. Autonomic Computing* (2009).
25. Roy, N., Dubey, A., and Gokhale, A. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Proceedings of the Intern. Conf. on Cloud Computing* (2011).
26. Yadwadkar, N.J. *Machine learning for automatic resource management in the datacenter and the cloud.* Ph.D. thesis, UC Berkeley, 2018.
27. Zhang, Y., Prekas, G., Fumarola, G. M., Fontoura, M., Goiri, I., and Bianchini, R. History-Based harvesting of spare cycles and storage in large-scale datacenters. In *Proceedings of the Intern. Symp. Operating Systems Design and Implementation* (2016).
28. Zheng, W., Nguyen, T.D., and Bianchini, R. Automatic configuration of Internet services. In *Proceedings of the 2nd European Conf. Computer systems* (2007).

Ricardo Bianchini is a Distinguished Engineer at Microsoft Research, Redmond, WA, USA.

Marcus Fontoura is a Technical Fellow at Microsoft Research, Redmond, WA, USA.

Eli Cortez is a Principal Engineer at Microsoft Research, Redmond, WA, USA.

Anand Bonde is a senior engineer at Microsoft Research, Redmond, WA, USA.

Alexandre Muzio is a software engineer Microsoft Azure, edmond, WA, USA.

Ana-Maria Constantin is a software engineer Microsoft Azure, Redmond, WA, USA.

Thomas Moscibroda is a partner research scientist at Microsoft Azure, Redmond, WA, USA.

Gabriel Magalhaes is a Ph.D. student at the University of Washington, and was an intern at Mucrosoft Azure during this work.

Girish Bablani is corporate vice president of Microsoft Azure, Redmond, WA, USA.

Mark Russinovich is a Technical Fellow and CTP at Microsoft Azure, Redmond, WA, USA.

Watch the authors discuss this work in the exclusive *Communications* video. https://cacm.acm.org/videos/ml-centric-cloud-platforms