Using reinforcement learning to enable a new class of adaptive service policies

Michael Schaarschmidt

1 Introduction

During the last decade, the advent of cloud computing has given rise to a new class of distributed computing infrastructures. Employing traditional on-premise servers is becoming increasingly undesirable for companies with rapidly expanding data and query volumes. Their expansion relies heavily on scalable and elastic services hosted on infrastructure providers like Amazon EC2 or Microsoft Azure. While elastic load balancing methods enable dynamic provisioning of additional computing capabilities, the inherent configurations of these machines are often not changed during runtime. Typically, such services operate on a convention over configuration paradigm which enables its users to deploy their products quickly. Consequently, productive systems barely use adaptive methods towards the larger part of their configurations. After all, one of the fundamental value propositions of cloud services is their usability, next to horizontal scalability as well as their flexible pricing models [9, 17].

Tenants of these services can express their requirements in the terms of service level agreements (SLAs). However, cloud service providers usually only provide SLAs on simple metrics like availability and durability [4]. Some research efforts have gone into employing online learning approaches towards these systems: Tesauro et al. have demonstrated how high-level resource allocation can be improved trough reinforcement learning (RL) [20, 19]. The proposed research aims to investigate how RL and statistical methods can be applied towards achieving and maintaining much more fine-grained SLAs. First, we will provide a brief overview of current trends in data-centric cloud services. In section 3, we will consider the role of SLAs as a means of communicating requirements between such services. Section 4 defines a concrete set of potential research directions regarding the opportunities and challenges when using RL techniques to create more efficient services. Finally, section 5 demonstrates a provisional work plan for the first year of doctoral work on this promising new field. The proposal concludes with far-reaching questions that will be targeted in the later stages and the final thesis.

2 From hosted infrastructures to managed services

2.1 Infrastructure-as-a-Service

Infrastructure-as-a-service (IaaS) describes the concept of providing computational capacities on demand. These capacities can typically be provisioned and monitored through interactive dashboards or an API. The provided server instances are allocated as virtual machines and not running on isolated physical machines. This enables the providers of

such services to optimally exploit the underlying hardware. For instance, Amazon Web Services's EC2 provides a wide range of instance types with different static configurations. Since tenants pay hourly rates per instance and might decide to provision new instances above some threshold of utilization in order to avoid slowing down their services, tenants rarely fully utilize the full capabilities they rent.

Another problem for tenants of IaaS providers is the fact that after renting out computing capabilities, customers still have to deal with installing, configuring and managing their backend services. Highly sophisticated data driven software products may demand such manually curated infrastructures. However, for a large group of applications, fully managed backends are a more preferable choice: configuring and maintaining a fast, scalable and fault-tolerant distributed backend infrastructure consisting of different types of databases, search servers [1], caching layers, etc. requires in-depth expertise that is often unrelated to the actual application domain a software product is addressing.

2.2 Backend-as-a-Service

Consequently, we have also seen a rise of managed databases, i.e. database-as-a-service (DBaaS) [10, 5]. A common business model (e.g. MongoDB [14]) is to develop an open-source database while commercializing it through managed clusters and integration services. Finally, backend-as-a-service (BaaS) provides not only a database, but tries to satisfy all data related requirements by adding search servers, caches, push notifications, analytics, messaging and so forth (e.g. Parse [8]). Note that features like push notifications are seemingly more targeted at end-customer mobile applications. The trade-off when using a BaaS is to get a "one size fits all" type of solution. For instance, Parse uses MongoDB as its database. MongoDB is a scalable schema-free document database that can provide a reasonable compromise on a lot of features. For example, server-side joins on documents are not supported but can be emulated by storing more data within a single document. Essentially, MongoDB can only provide query performance if the data can be fit into a document structure. Such constraints are not acceptable to large-scale applications, which usually have to manage multiple types of structured and unstructured data sources. So far, BaaS providers have not managed to provide solutions that can capture the multitude of requirements that come with large-scale applications. In the next section, we will demonstrate how these developments relate to service level agreements as a form of abstracting specifications and will argue how such agreements could play a pivotal role in the next generation of backend infrastructures.

3 SLAs and the BaaS model

3.1 Defining service goals

Customers and service providers usually define the terms of the desired quality of service in the form of a service level agreement [11]. A primary example is an SLA over the availability of a service. For instance, the EC2 service level agreement [2] promises an uptime of 99.95% in a monthly billing cycle and will refund partial credit for failure to deliver on that promise. Other SLAs consider durability or 90th/95th percentiles on response time. On a more abstract level, we can consider SLAs as a part of the specification when integrating an application into a specific platform.

Consider the following example: A BaaS provider manages a MongoDB cluster with multiple replica sets for an enterprise customer. To ensure consistency, all reads on

the database are routed to a single master node. Writes are replicated with eventual consistency, resulting in δ-atomicity. MongoDB offers various settings regarding the acknowledgement of write operations: *replica acknowledged* means that results of a write operation need to be acknowledged on all replicas, which naturally requires extensive locking periods. On the other end, *acknowledged* essentially creates the behaviour of an in-memory database because changes are only immediately applied to the in-memory view of the data. During peak loads, the provisioned capacity in our example cluster might be insufficient to uphold the specified agreement on response times because all operations are bottlenecked at the master. Changing the write concern to achieve higher performance during peak loads would be straight-foward. The fundamental problem of such adaptive policies is that the service provider usually cannot make an informed decision without knowing the preferences of the tenant. Is response time more important than replicating data as soon as possible? Which requirements can be relaxed under which circumstances?

This issue can easily be transferred to various other aspects of data management. Other examples include choice of database, caching policies (trade-offs on staleness), logical organization of the database, encryption levels (e.g. only encrypt sensitive fields under high loads), batching, transactional semantics, network level configurations (e.g. temporarily rate-limiting calls to a REST API) and many more. The interesting observation is that the reason a lot of these services are not as adaptive as they could be is not that there is an inherent technical hurdle. It is rather a lack of information on both ends that poses the problem. Application developers or researchers who are just looking for a tool to carry out their experiments cannot be expected to be aware of intricate tuning mechanics. Conversely, providers of managed services have in-depth knowledge on their systems but not on the preferences of their tenants.

3.2 Fine-grained SLAs

In summary, we arrive at two connected questions: how can tenants of complex systems express their preferences in terms of how the system should react in certain situations when they are not aware of the trade-offs of the backend infrastructure they are using? Second, how could service providers act upon this information? One solution could consider a set of fine-grained SLAs that are managed in an opt-in fashion. As discussed earlier, SLAs are currently used as a high-level specification of service targets. We can thus extend the notion of SLAs into a richer means of communicating preferences in terms of system behaviour, which could entail both functional and non-functional goals. In our earlier example, this translates to assigning different weights to response time, minimal cost and safe writes (i.e. fully replicated writes). Translating this into the current user interfaces of BaaS/IaaS platforms is straight-forward and can be presented to users through sliders in a dashboard. The proposed research will primarily explore how these methods can be carried out on the site of the provider. The challenge lies in the fact that while the provider is generally aware of the trade-offs and configurations of the systems he manages, making ad-hoc decisions for every tenant under changing workloads is still not straight-forward. In the next section, we will propose reinforcement learning as a potential solution to this problem.

4 Reinforcement learning in adaptive configurations

Reinforcement learning concerns the question what action an agent needs to take in a certain environment to maximize his future rewards [18]. The RL agent does not require

any prior knowledge about his environment and instead learns from the rewards (or lack thereof) gained by his actions. Thus, the agent constantly needs to balance exploration (i.e. trying out new actions) against exploitation of his policy (i.e. gaining rewards from his understanding model of the most profitable actions). This includes randomly selecting actions that are deemed to be suboptimal under the current belief set of the agent. Formally, the agent will try to maximize the expected rewards over an infinite series of actions discounting the decreased value of a current action in future states to avoid infinite rewards.

Most RL models operate under the assumption of a Markov Decision Process (MDP), i.e. future states can be determined by exclusively looking at the current state. Techniques of RL are particularly useful in situations when constructing an accurate model is infeasible or undesirable due to high complexity. For instance, predictive statistical models can be employed for many data related tasks by the means of approximating distributions on workloads and access patterns. Building such a model can require extensive experiments, simulations and, depending on the type of model, parametrization. Thus, RL techniques have been used for various highly dynamic tasks like autonomously flying helicopters or optimizing stock portfolios [15, 16]. RL is thus an interesting candidate for tasks in dynamic data management. In the remainder of this section, we will introduce some specific research questions regarding the use of RL in this application domain.

4.1 RL in large-scale systems

Using RL in the context of SLAs seems promising because SLAs define clear goals that can be measured through utility functions. Utility functions map the value of specific metric (e.g. latency) to a (normalised) utility value, which can be understood as a form of reward. For instance, availability might follow a sigmoid-like function for a lot of applications: if a popular web application is available only 95% of the time, this level of service has close to zero utility for a customer because the service outages might cost too much revenue. Utility then rapidly increases up to an availability of 99.95% and only marginally increases after that, because a few minutes of downtime might be tolerable. So far, the of using RL instead of simple threshold-based methods [6] has been adopted specifically in the space of optimizing resource allocation in clusters, i.e. auto-scaling [7, 20, 21, 13].

Translating these models to more complex trade-offs is challenging for multiple reasons. First, as seen in our earlier example of relaxing write concerns in turn for higher troughput, we still need to manually identify all actions that could be involved in these trade-offs and integrate interfaces into multiple application layers to enable the agent to adjust them. Hence, even though RL is a model-free approach, we implicitly require considerable domain expertise to let the agent interact with the system. Second, the number of possible states and actions will grow drastically with an increasing number of such tunable parameters. We must thus also consider function approximators like neural networks [20] instead of large lookup tables [13]. Additionally, the states in the environment of dynamic data management are highly complex, as are the action sequences.

Since a productive system cannot function well on an arbitrary initial state, considerable offline training might also be required. Another issue is the way RL models deal with change in later stages of the model. While the rate of random (suboptimal) actions taken may decrease over time, databases or productive software systems in general will still undergo changes. One the one hand, new versions of existing software as well as significant additions to the system can render learned policies useless. Additionally, even

if the system does not change, the workloads might change so drastically that that the behavioural model cannot react because the learning rate is also decreased. Thus, novel methods must also consider hybrid methods (e.g. turn off learned policy and revert to default mode under some circumstances) and convergence speed-up [7, 12].

4.2 Managing resources and configurations

A principal challenge of embedding any kind of fine-grained model is the modification of applications. For instance, Angel et al. [3] have investigated the issue of guaranteeing throughput in dynamic multi-tenancy cluster environments. This is achieved by utilizing a cost model that differentiates between different types of operations on the level of network request header inspection. The authors specifically refrained from making any changes to applications, operating systems or network infrastructures. The proposed research aims to achieve fine-grained models by interfering with actual configurations and is not restricted to managing predefined units of resources. It is yet unclear whether such an approach is feasible or scalable.

Considering the previously introduced issues of using RL in large-scale data management, the proposed research needs to investigate at what level the overhead of interfacing configurations is justified. Ideally, we would be able to address and manipulate configurations as actions in an RL policy in a generic way. In our earlier example, we considered write-acknowledgement configurations for a specific database that can be changed operation-wise. Some parameters (e.g. partitioning model) cannot easily be changed online. We can identify one approach as looking at application-specific parameters and dealing with them in a bottom-up manner, trying to determine whether there is a common way of learning them. However, it might also turn out that achieving application-specific optimizations like in our example require too much specific work on interfacing and analysing the specific application. This would render the idea of using a RL approach impractical: there is no point to using model-free learning when it requires too much specific domain-expertise and implementation overhead to customize the RL model.

Conversely, we can also consider the previously mentioned top-down approach. Here, we restrict the agent to discover policies by looking at the visible interactions (e.g. network behaviour, throughput etc.) between services and applications that are considered encapsulated units. Ultimately, the proposed research needs to investigate to what extent current learning models are able to interact with data management services on a deeper level.

5 Research goals and provisional schedule

5.1 First year plan

In the previous sections, we have outlined the current state of dynamic data management and have identified RL as a potential path towards novel methods. Research in this relatively new avenue must first survey potential indicators that make themselves available to being modelled by an SLA. Next, a RL model must be designed that is specifically targeted towards challenges in data management and deals with the issues we introduced in the last section. Specifically, it must also be evaluated whether RL methods are superior to model-based approaches and if they can significantly outperform simple threshold models. After devising a model, early implementations will explore different types of

adaptive applications. For a starting point, these implementations can consider dedicated single-tenant DBaaS set-ups.

5.2 Further efforts

Some configurations may affect the behaviour of a system for all its tenants, depending on the hosting model. For instance, one physical database server can host multiple tenants on one instance of the database using access control lists or similar measures regarding user privileges. Changes in partitioning, replication etc. would naturally affect all tenants. Conversely, each tenant can have a dedicated database process on a single machine, which in turn often incurs higher overhead. Further efforts in the proposed doctoral work will thus need to consider how these aspects of dynamic data management behave in changing multi-tenant systems. This specifically needs to consider what kind of generalisations can be made in terms of learned policies: can tenants be classified to share the policy of the same agent? Can the policy for one client be used as the starting point of the policy of newer clients that indicate similar priorities in terms of SLAs? If multiple isolated agents learn in parallel, are there ways to share intermediate results to accelerate convergence?

Finally, how can the required parameters in large scale systems be collected efficiently, i.e. using streaming and sampling techniques. We will also consider how this work can be applied to a wider scope of applications, i.e. not only configurations of databases or data management services but also search servers, execution engines etc. The final thesis will aim to deliver new practical ways of both negotiating requirements and preferences for data management as well for achieving optimal tradeoffs autonomously.

References

[1] Elasticsearch.org open source distributed real time search & analytics | elasticsearch.

[2] Amazon Web Services. Aws ec2 sla terms. http://aws.amazon.com/de/ec2/sla/.

[3] Sebastian Angel, Hitesh Ballani, Thomas Karagiannis, Greg O'Shea, and Eno Thereska. End-to-end performance isolation through virtual datacenters. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 233–248, Berkeley, CA, USA, 2014. USENIX Association.

[4] Salman A. Baset. Cloud slas: Present and future. *SIGOPS Oper. Syst. Rev.*, 46(2):57–66, July 2012.

[5] Carlo Curino, Evan PC Jones, Raluca Ada Popa, Nirmesh Malviya, Eugene Wu, Sam Madden, Hari Balakrishnan, and Nickolai Zeldovich. Relational cloud: A database-as-a-service for the cloud. In *Proc. of CIDR*, 2011.

[6] X. Dutreilh, N. Rivierre, A. Moreau, J. Malenfant, and I. Truck. From data center resource allocation to control theory and back. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 410–417, July 2010.

[7] Xavier Dutreilh, Sergey Kirgizov, Olga Melekhova, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow. In *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*, page 67–74, 2011.

[8] Facebook. The parse backend platform. https://parse.com/.

[9] A. Gohad, N.C. Narendra, and P. Ramachandran. Cloud pricing models: A survey and position paper. In *Cloud Computing in Emerging Markets (CCEM), 2013 IEEE International Conference on*, pages 1–8, Oct 2013.

[10] H. Hacigumus, B. Iyer, and S. Mehrotra. Providing database as a service. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, page 29–38, 2002.

[11] Wolfgang Lehner and Kai-Uwe Sattler. *Web-Scale Data Management for the Cloud*. Springer, New York, auflage: 2013 edition, April 2013.

[12] Michael L Littman. Algorithms for sequential decision making. Technical report, Providence, RI, USA, 1996.

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

[14] MongoDB, Inc. MongoDB. http://www.mongodb.org/.

[15] J. Moody and M. Saffell. Learning to trade via direct reinforcement. *Neural Networks, IEEE Transactions on*, 12(4):875–889, Jul 2001.

[16] AndrewY. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In Jr. Ang, MarceloH. and Oussama Khatib, editors, *Experimental Robotics IX*, volume 21 of *Springer Tracts in Advanced Robotics*, pages 363–372. Springer Berlin Heidelberg, 2006.

[17] Sakr, S. et al. A survey of large scale data management approaches in cloud environments. *Communications Surveys & Tutorials, IEEE*, 13(3):311–336, 2011.

[18] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[19] G. Tesauro, R. Das, W.E. Walsh, and J.O. Kephart. Utility-function-driven resource allocation in autonomic systems. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 342–343, June 2005.

[20] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *Proceedings of the 2006 IEEE International Conference on Autonomic Computing*, ICAC '06, pages 65–73, Washington, DC, USA, 2006. IEEE Computer Society.

[21] Jianxin Yao, Chen-Khong Tham, and Kah-Yong Ng. Decentralized dynamic workflow scheduling for grid computing using reinforcement learning. In *Networks, 2006. ICON '06. 14th IEEE International Conference on*, volume 1, pages 1–6, Sept 2006.