UNIVERSITY OF
CAMBRIDGE

Deep reinforcement learning in cloud data management

Michael Schaarschmidt

Supervisor: Dr. Eiko Yoneki

First Year Report

Computer Laboratory

June 2016

Contents

Chapter 1

First year report

This chapter provides an overview of research projects and training undertaken during my first year apart from my thesis project. Chapter 2 gives an introduction and background to machine learning in systems research. Chapter 3 outlines the thesis proposal, initial experiments, and a dissertation schedule.

1.1 Research training

In my first year, I aimed to develop both a deeper understanding of machine learning techniques used in systems research as well of the state of the art in (deep) reinforcement learning. To this end, I attended lectures on the topics of statistical signal processing, reinforcement learning and deep learning. I also implemented a number of state-of-the art deep reinforcement learning algorithms as part of a proof-of-concept case study, as discussed in chapter 3.

Further, the researcher development program (RDP) of the University of Cambridge offers a number of courses and seminars on general skills unrelated to particular research topics. I have attended the following RDP seminars:

- Introduction to Research Integrity at Cambridge,
- Building Resilience and Coping with Setbacks,
- The Art of Negotiation and Influence,
- Effective Researcher,
- Getting the Most out of Conferences.

In April, I also attended the *EuroSys 2016 Doctoral Workshop (DW)*. The DW provides a forum for early stage doctoral students to both receive

feedback on their research proposals and to practice the format of conference talks. DW participants were also given the opportunity to present a poster at the main poster session. The discussions at the poster session as well as the paper presentations in the main conference track helped me understand current trends in applying machine learning techniques to improve runtime behaviour of computer systems. These observations are discussed in more detail in chapter 2.

Finally, I will participate in the Microsoft Research Cambridge PhD summer school in July, where I will again present a poster to obtain further feedback.

1.2 Quaestor

In Michaelmas, I primarily worked towards completion of the Query Store (*Quaestor*) project, which is an extension of my MPhil dissertation work [61].

Quaestor is a caching architecture for query results which accelerates load times for websites and applications by utilising existing web caching infrastructure. The project is a collaboration with two PhD students at my undergraduate institution (University of Hamburg, Germany), Felix Gessert and Wolfram Wingerath. Felix and I have developed the distributed caching algorithm, the TTL estimation mechanism and the experiment design. Wolfram contributed the stream processing-based invalidation engine that matches database updates to cached query results to determine invalid cached entries.

The experimental infrastructure was built throughout summer 2015 and took until October to complete. The complexity of the experiment was rooted in the multitude of interacting components with strict latency requirements. The central components are the distributed stream processing pipeline, a DBaaS middle ware containing query processing logic, a custom built message queue for interfacing Storm from the DBaaS, a distributed shared state mechanism implemented on top of a key-value store, an invalidation engine interfacing a third-party content delivery network, a client side API supporting the caching logic, and a benchmarking framework on top of the API generating workloads. Finally, I implemented an orchestration framework for AWS EC2 managing all these components to automate experiment cycles.

Major delays arose from unexpected malfunctioning in the interaction of these components, thus causing us to miss the *EuroSys* deadline. For illustration, the content delivery network (CDN) we utilised as a third-party service (Fastly) advertised to support collapsed forwarding. Our experiments showed unexpected latency spikes up to multiple seconds skewing our results distributions. After a lengthy investigation, it turned out that

Fastly could not collapse requests without changing the processing order significantly, thus introducing latency spikes for random requests.

By the end of Michaelmas term, the *Quaestor* project was completed. Our initial submission to *Very Large Databases (VLDB)*, of which I am a joint first author, was rejected. In part, the reviewers did not recognize a clear narrative on the necessity of each of the components and their precise functionality. A further key criticism was that the reviewer doubted our system could not deliver the rich consistency guarantees the paper claimed. We view this as a failure on our end to not explicitly discuss each consistency mode with a clear comparison to prior work on adaptive consistency. We are currently preparing a revised version of the paper addressing these comments and various other minor issues. An improved version will be submitted to *SIGMOD 2017* in mid July. The most recent (not necessarily final) version of the paper is attached to this document.

1.3 Probabilistic garbage collection optimisation

1.3.1 Background

Throughout Lent and Easter term, I helped Valentin Dalibard, a 4th year PhD student, with a case study on Structured Bayesian Optimisation (SBO) [29]. The purpose of this project was to explore systems use cases for Valentin's work. The case study investigated automated Java virtual machine Garbage Collection (GC).

The key idea of Valentin's doctoral work is to optimise high-dimensional parameter spaces in systems problems by injecting the structure of the problem into the optimisation procedure. I hence spent the majority of Lent and the beginning of Easter term with setting up an experimental infrastructure, interfacing Valentin's framework and analysing GC logs to devise a probabilistic model for a general GC optimiser.

1.3.2 Analysis

A number of performance problems arise from the unpredictable nature of GC interruptions. To avoid running out of heap space, the garbage collector periodically attempts to reclaim memory by removing objects that are no longer used. Garbage collection introduces overhead to applications by requiring resources to scan memory regions and mark objects. Depending on the GC algorithm used, a portion of GC can be executed concurrent to the program by separate threads. However, common GC implementations still require a 'stop the world' phase which completely halts the application

to ensure a consistent object graph.

A typical problem for performance-sensitive applications is 'stop-the-world' collection, which creates latency spikes that last up to several seconds or minutes on large server-class machines. In worst case scenarios, this can cause connection time-outs and cluster rebalancing. In contrast, applications running on machines with smaller heaps do not take as long to reclaim memory but need to ensure efficient object management to avoid frequent 'stop-the-world' or minor interruptions that impact request latency.

Designing and evaluating a GC optimisation tool requires a GC-sensitive application that can be tested under a variety of different workloads. We opted to use Cassandra [34], a popular JVM-based column store in combination with the Yahoo! Cloud Serving Benchmark (YCSB) [22]. YCSB is a benchmark for cloud databases and defines a set of typical web workloads (e.g. read-dominant, scan-intensive, write-dominant). Custom workloads can be specified through typical properties such as request distribution, record count, operation count, read/write/insert/scan/delete mixture, fields per record and field length. After running a workload, YCSB provides throughput and latency histograms for each type of operation.

A Cassandra installation contains a number of JVM configuration files, including scripts for garbage collection, which determine optimal heap and generation sizes depending on machine memory and number of cores. It is a reasonable assumption to expect these default settings to be hand-tuned to perform well on a wide range of typical workloads. The performance of default JVM flags can hence be compared against Cassandra's configuration script and against an SBO-based optimiser.

We constructed a probabilistic model by analysing GC logs and investigating correlations between GC events and YCSB performance metrics. The structure in the model is characterised by mapping critical GC parameters to GC events instead of directly applying Bayesian optimisation to GC parameters. In the next step, a model was trained to predict utility (e.g. 99th percentile latency or throughput) from the occurence of GC events. This is based on our findings that certain GC events (e.g. promotion failure between regions) provide a less noisy estimate of the quality of a GC configuration than the actual benchmark results.

The resulting optimiser can improve 99th percentile latency by up to 60% compared to the Cassandra default configuration and by 70% compared to default JVM flags, as seen in figure 1.1. Implemented in 800 lines of C++ on top of Valentin's framework and probabilistic C++, the optimiser can in principle be supplied with any GC sensitive application.

The results of this project will either be submitted to *ASPLOS* in August as a collaborative paper with Valentin, where the focus of the pbulication will lie on introducing his framework. Alternatively, the project could

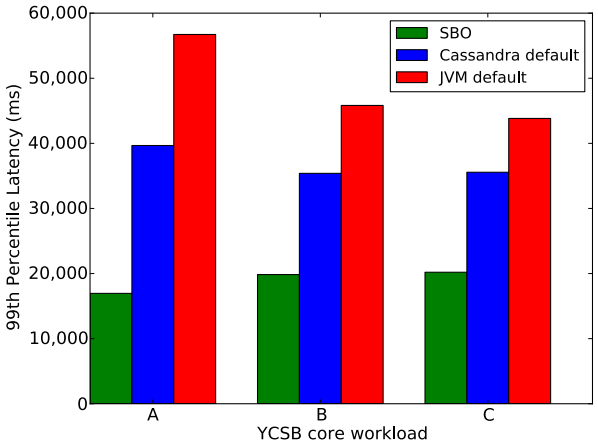

Figure 1.1: Results for YCSB workloads A, B and C.

be extended to be published on its own in the future. There is a number of straight-forward extensions such as the investigation of other applications types, other GCs (e.g. the G1 collector) and other optimisation goals (or to allow for multiple goals). These extensions are primarily engineering tasks and might also be suitable starting points for MPhil projects. A more interesting route would aim to incorporate a much larger variety of runtime information, i.e. comprehensively modelling all GC events as well as heap information available from GC logs rather than relying on simple correlations between GC events and optimisation goals.

Chapter 2

Machine learning in systems research

2.1 Overview

This chapter provides an overview of how machine learning techniques are currently used in data management, cloud computing and computer systems. Further, an introduction to reinforcement learning and in particular Q-learning is given. The limitations of naive Q-learning, extensions and alternatives are briefly discussed. I will begin by surveying some trends in recent work. Computer systems manage an ever growing amount of physical and virtual resources (software services). In the past decade, the advent of cloud computing has enabled applications to access computational resources with unprecedented flexibility. Further, cloud-based infrastructure, e.g. infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), and database-as-a-service (DBaaS) have significantly shortened application development cycles by omitting the operational complexity of server and database management behind REST APIs and software development kits. These developments have raised a number of research challenges for learning in systems.

Data centres rely on a vast set of configuration parameters governing their runtime behaviour on every level from server power management [51, 52] and network protocol behaviour to higher abstraction levels such as virtual machine provisioning and load balancing. Dynamic parameters can be governed through static thresholds or sets of rules, physical models governed by partial differential equations (e.g. thermal modelling), statistical/machine learning models, or hybrid approaches.

There are several challenges in providing more efficient parameters at runtime. The number of configuration parameters is large and for many parameters, the relevance to system performance might be unclear even to experienced users. What is more, even if a parameter is ostensibly impact-

ing performance, determining a model for a runtime controller might be uneconomical. First, system dynamics might be noisy, high-dimensional and not fully observable (e.g. in a geo-replicated system), thus requiring considerate effort and expertise to devise analytical models. Second, the recent surge in unsupervised and semi-supervised machine learning has provided researchers with powerful abstractions such as various forms of deep neural networks [12, 32, 63]. However, these models often require large amounts of training data as well as significant computational resources and long training periods for complex models, e.g. up to weeks of training on multiple GPUs or CPU clusters [70]. In consequence, deep learning models have found little application in optimising system performance itself. Instead, systems research has focused on providing more efficient computing abstractions to train these models, e.g. through frameworks like TensorFlow [1] and Spark MLlib [53] or research into distributed GPU scheduling and gradient exchange models [28, 40, 47]. In the following sections, I will review some of the current models and use cases.

2.2 Resource management

The current direction of research in optimising runtime configurations for resource management seems to favour comparatively simple parametric models which are tuned through offline analysis. In the following,I will survey some recent work in this area, including some examples from this year's *EuroSys* conference.

Teabe at al. devised a virtual CPU scheduler that adjusts time slices length depending on application type. Application types were identified through their level of CPU usage, I/O intensity and concurrency. The authors identified typical values for each of a set of predetermined application types through offline analysis of benchmark traces. Runtime decisions were made through classifying the current workload according to these characteristics.

A similar approach was employed by Kumar et al. to determine optimal waiting times for aggregation queries [42]. Result aggregators in large scale parallel processing engines (e.g. search engines) wait for outputs from processes for a certain duration before sending them upstream. Longer waiting times increase response quality at the risk of missing upstream deadlines. The authors utilise an offline process to determine the distribution for process durations from workload traces and estimate the parameters of that distribution at runtime to adjust waiting times.

For a third example, also from this year's *Eurosys*, Sharma et al. introduced a framework to run data-intensive processing tasks with Apache Spark on Amazon EC2 spot instances [64]. Their approach helps decrease

runtime and save instance costs by adapting checkpointing policies and server placement locations to the fluctuations of the cloud spot market, where prices and failure rates vary for different instance types and for every region, reflecting current demand levels. Again, these runtime improvements were achieved through a hand-crafted parametric model calibrated through historic price data. In a similar vein, Venkataraman et al. have recently proposed a performance prediction framework for Spark that fits a parametric model based on execution times on small data samples [81]. These task runtime predictions can then be used to find the cheapest cloud server configuration to finish a task within a deadline.

2.3 Service customisation

A different avenue of research into runtime configurations is concerned with customising cloud services according to specific user goals. Cloud services often operate on a convention-over-configuration paradigm. For instance, a database-as-a-service might provision a database cluster with a variety of default settings regarding allowed incoming connections, replication scheme, journalling, consistency (e.g. allowing stale reads from replicas or not) and so forth. These settings might not necessarily reflect the performance requirements of all customers, but will work well for typical applications. A straight-forward solution to this is to expose a configuration file or a web interface that lets users adjust default settings when needed. However, this defeats the purpose of omitting operational details of the database infrastructure from developers.

An alternative approach suggests the use of service level agreements (SLA) to communicate specific performance requirements (service level objectives) [11, 83]. Cloud service providers can then try to provide performance trade-offs reflecting these requirements by adapting runtime parameters. In return, providers can charge customers on a more granular level for meeting these goals. Terry et al. have explored this notion in the context of consistency SLAs for key value stores [72, 71]. Their model enables users to define utility levels and accepted price for concrete response times and consistencie levels so that for each read operation, a good response-time/consistency trade off can be determined. This is achieved through a client-library that estimates the probability of reading a certain consistency level from a replica based on tracking time stamps retrieved from reach replica. Developers are hence required to deal with different consistency levels in their application code by specifying actions for a given consistency level returned by a read operation.

A key differentiator of SLA-driven models is that they not necessarily result in an absolute improvements for all performance metrics, but rather give more flexibility to clients through declarative trade-offs.

2.4 Challenges

In summary, these approaches can achieve substantial performance or usability improvements at the cost of extensive system analysis to devise models for specific parameters under a number of assumptions about workloads and environment. Further, manual parameter modelling helps researchers and software architects understand system constraints. On the other hand, the volume of configuration parameters in contemporary systems makes manual modelling only feasible for a few critical parameters. What is more, models based on historic workload traces, often designed for a discrete set of situations, can only approximate the dynamic of production systems with constantly evolving conditions. Ideally, offline analysis and simulation should be used to determine prior parameters which should nevertheless be improved at runtime. In my doctoral work, I plan to investigate state-of-the art model free learning techniques to determine efficient runtime parameters. In the following section, I will introduce reinforcement learning as a framework towards this task.

2.5 Reinforcement learning

Reinforcement learning (RL) is a machine learning technique characterised by software agents that interact with an environment and learn optimal behaviour policies through reward signals. At every discrete time step t, the agent is in a state s_t, takes an action a according to its current policy $\pi(a_t|s_t)$, and moves it into a new state s_{t+1}. State transitions are often stochastic and governed by a state distribution $p(s_{s+t}|s_t, a_t)$. Upon taking an action, the agent observes a reward signal $r(s_t, a_t)$. The goal of the agent is to learn a policy π which maximises the cumulative expected reward $R = \mathbb{E}[\sum_t \gamma^t r_t]$, where future rewards are discounted by γ. Markov decision processes (MDP) provide the formal framework for these optimal control problems. A comprehensive introduction to the topic and various RL approaches (value-based, policy-based, model-based) is given by Sutton [67].

The fundamental advantage of RL is the agent's ability to learn good policies without requiring an explicit model of the environment. RL is hence especially attractive in noisy environments with complex dynamics. Variations of RL have been used in various applications, notably navigation in robotics [36, 86] and board games [35, 50, 75, 84]. In the following sec-

tions, I will introduce Q-learning as one of the most popular RL algorithms and will discuss prior attempts of utilising it in systems research.

2.6 Q-learning

2.6.1 Algorithm

Q-learning is a value-based RL technique which finds policies by comparing expected cumulative rewards (Q-values) in environments with stochastic transitions and rewards [82]. This is achieved by iteratively constructing a Q-function $Q : S \times A \to \mathbb{R}$ through the following update rule:

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right] \quad (2.1)$$

In a finite Markov decision process with discrete action and state spaces, the Q-function can be conceptualised as a simple look-up table with random initial values. The Q-learning algorithm can be understood as follows: The agent is in state s_t and uses an ϵ-greedy policy to decide what action a_t to take. This implies to choose an a maximising the Q function for the given state s in general and choosing a random action with a small probability ϵ to ensure sufficient exploration. The Q-value $Q(s_t, a_t)$ for the state-action pair is then updated by adding the new observation to the current Q-value adjusted by the learning rate α. The update term (in square brackets) consists of the reward observed r_{t+1} and the Q-value achieved by following a greedy policy from the new state s_{t+1} subtracted by the current Q-value, discounted by γ. Intuitively, this means that Q-learning bootstraps its estimate of the current state value by *its own* estimate of optimal future action values. Q-learning is an *off-policy* RL algorithm: the agent learns an action-value function based on a greedy estimate of future action values even though his policy is not actually greedy. In contrast, on-policy algorithms (e.g. SARSA) update their Q-function by observing the value of the actual policy being followed, i.e. by taking another step and observing the next state. It can be shown that Q-learning converges to an optimal policy for finite MDPs [82].

2.6.2 Function approximation

Q-learning has several limitations and an extensive body of research exists around around addressing them. In this section, I will discuss common extensions and alternative approaches such as policy-based RL. First, very large state and action spaces cannot be conceptualised through look-up tables efficiently since they would be too large to be held in memory. Learning the value of each individual state would also take infeasibly long. This

motivates the notion of *function approximation* [79, 68], i.e. the idea to generalise from seen states to unseen states to approximate their Q-value. The function approximation of the true action-value function $q_\pi(s, a)$

$$\hat{q}(s, a, \theta) \approx q_\pi(s, a) \tag{2.2}$$

is governed by a set of parameters θ which are incrementally adjusted, e.g. by minimising the mean squared error. Given some differentiable function of the parameters $J(\theta)$:

$$J(\theta) = \mathbb{E}_\pi[(q_\pi(s, a) - \hat{q}(s, a, \theta))^2], \tag{2.3}$$

this is often achieved through gradient descent on θ. Many function approximation schemes exist; from simple linear feature combinations to neural networks or Gaussian processes [60]. It should be noted that convergence to an optimal policy is not guaranteed when using a non-linear function approximation scheme. Convergence issues are discussed in more detail in the section on deep reinforcement learning in chapter 3.

2.6.3 Limitations and alternatives

Q-learning requires a maximisation to be performed at every step to determine which action to take. This makes naive Q-learning impractical for large or continuous action spaces since it would require an iterative numerical optimisation to determine the optimal action in the continuous case. Another class of RL are policy-gradient algorithms. Q-learning extracts a policy implicitly through an ϵ-greedy strategy. Instead, the policy can also be directly parametrised by θ [87] and conceptualised as a parametric probability distribution:

$$\pi_\theta(s|a) = \mathbb{P}[a|s; \theta]. \tag{2.4}$$

The quality of a policy can be assessed for instance through the average reward per time-step. The policy is then incrementally improved through the gradient

$$\frac{\partial \pi(s, a)}{\partial \theta}. \tag{2.5}$$

The policy gradient theorem shows that an estimate of the gradient can be obtained from experiences (rewards) [69]. Practically, this can be for instance implemented through a Gaussian policy which uses empirical averages (Monte Carlo estimates) over multiple episodes to estimate $\nabla_\theta J$. An overview of common gradient estimation and variance reduction techniques is given by Zhao et al. [87]. For a more recent example, Silver et al. have introduced deterministic policy gradients (DPG) as an policy-based method to deal with high-dimensional or continuous tasks [65]. The key insight

of DPG is that by using a deterministic (instead of the usually stochastic) policy, no integration over the action space is necessary, thus requiring fewer samples. Further, In the form given above, Q-learning is also not sample efficient since every state-action-reward observation is immediately discarded after the update step. What is more, the utilisation of non-linear state-action value function approximation methods (e.g. neural networks) introduces instability due to correlations in observation sequences [80]. In conclusion, reinforcement learning provides a powerful task learning framework that is constrained by various practical limitations.

Chapter 3

Thesis proposal

3.1 Introduction

This thesis proposal describes the work planned on the topic of learning runtime parameters in computer systems via deep reinforcement learning. In particular, my doctoral work seeks to demonstrate how the practical gap between prior proof-of-concept experiments and utilising recent advances in RL in large scale computer systems can be closed. To demonstrate the feasibility of learning request level parameters in data-processing systems, one must address several challenges.

First, current deep reinforcement learning research is primarily focused on learning episodic tasks with static goals. In contrast, cloud services or more generally computer systems are dynamic in multiple dimensions. Not only do request patterns and data volume change, the infrastructure itself is elastic and will add and remove nodes over time. Algorithms and training methods hence need to be adapted to leverage a changing infrastructure, e.g. by managing fluctuating numbers of distributed learners exchanging gradients or value function parameters [57]).

Second, the reinforcement learning community relies on a common set of tasks and tools (task simulators) to evaluate algorithms and training mechanisms, as will be discussed later. To reflect the specific problems encountered in computer systems, new tools have to be developed. Finally, several cases studies need to be carried out to demonstrate the feasibility of the approach in optimising request flows in practical data processing systems.

The proposal is structured as follows. First, I will discuss some prior applications of RL in computer systems research. Second, an overview to the topic of deep reinforcement learning is given. The remainder of the proposal contains a description of the simulation tool I developed as well as the results of an initial case study. The proposal concludes with a research plan and a task schedule for the remaining terms.

3.2 Reinforcement learning in computer systems

Tesauro et al. have investigated RL for dynamic resource allocation in data centres [73, 74, 77].Their work provides early insights (from around 2006) into attempting to connect data centre load metrics to SLAs through non-linear function approximation. Specifically, their method attempts to maximise the expected sum of SLA payments while minimising SLA penalties for unmet service level objectives. Their state model consists of the mean arrival rate of HTTP requests and the number of servers currently allocated as well as the server count from the previous step. A two-layer Multi-Layer Perception (MLP) maps these inputs to a single linear output. This output in turn is interpreted to make allocation decisions such as a assigning a number of servers to an application, or to add and remove servers in batch processing mode. The MLP is trained offline through back-propagation with data generated from allocation decisions from a custom-designed queueing model. This somewhat bypasses the problem of learning good policies in environments where such hand-crafted models are unavailable. Tesauro's work pre-dates both the proliferation of cloud computing as well as the advances in more sophisticated training methods for deep neural networks at the end of the last decade. Nonetheless, it illustrates the principal viability of non-linear function approximation to make online configuration decisions in computer systems. The authors have also successfully applied their hybrid approach to power management in web servers [76].

Other early applications of RL in computer systems can also be found in adaptive routing. Q-routing is a dynamic routing strategy based on Q-learning [16, 43, 44]. Routing is an appealing problem to explore RL since each node only has a finite number of neighbours and a simple tabular value function representation can be used. Q-routing estimates the delivery time to a specific node if packages are sent over a certain neighbour. Q-learning based algorithms are not mentioned in recent survey papers on routing, nor in Cisco's documentation of routing protocols [3, 58, 20]. This is likely because the discrete, finite nature of routing problems make them accessible to a large variety of mature low-overhead solutions, e.g. by relying on metrics such as reliability, bandwidth, load and communication cost or through greedy distance-vector protocols.

Nonetheless, the work on Q-routing has recently been extended to Q-caching in the context of information centric networking (ICN) [13, 17]. Q-caching is based on the notion of utilising routing information to cache the items with the highest expected cost to retrieve in case of a cache miss. Chiocchetti et al. have also considered RL for ICN to introduce a hybrid request forwarding scheme which combines deterministic request forwarding

towards known data copies with exploring the network via request flooding [19, 18].

For a more recent example, RL has been employed towards auto-configuration of Xen VMs [59, 85]. The authors have identified a number of performance-critical parameters such as the number of allowed clients, threads or connection time-outs and employ RL to periodically reconfigure VMs based on response times. They similarly use an offline model of the environment to train an MLP and have further simplified the problem by only allowing increase, no-op and decrease actions for integer-type parameters.

Overall, RL has been utilised for various control problems in computer systems research. However, rarely do these projects seem to have surpassed the stage of proof-of-concept experiments. No generic framework for model-free online learning (e.g. in an ecosystem like Hadoop) exists, which is potentially the consequence of too narrow modelling assumptions as well as the requirement of having high-quality offline training data and a software framework for training. Further, many of these approaches pose impractical limitations with regard to memory overhead, stability, robustness and scalability towards the request volume and level of parallelism found in contemporary data centres. For practical use of RL in data management, both the inherent algorithmic limitations of RL algorithms as well as implementation issues need to be addressed. In the following section, I will discuss deep reinforcement learning as the most recent direction in RL research which alleviates some of these problems.

3.3 Deep Reinforcement learning

3.3.1 Algorithm

In late 2013, Mnih et al. presented the first version of a class of algorithms that have since become known as deep reinforcement learning (DRL) [55]. Their algorithm achieves superhuman performance in a number of Atari games (e.g. breakout) while only requiring raw pixel values of the game simulator as input, thus providing an end-to-end task learning model. A deep convolutional neural network is utilised as a value function approximator to automatically extract visual features. Deep reinforcement learning combines several insights on how to address the above-mentioned limitations of previous Q-learning approaches. Further, it relies on the recent research efforts into efficient deep neural network training methods for powerful value function approximators.

The first key component is to train the value function from a replay memory, a concept first discussed by Lin in the context of robot controllers [49]. Instead of updating Q-values after each sample and thus training from

correlated sequences, observation sequences (i.e. state-action-reward-state tuples) are stored to a buffer. Periodically, sequences from the replay memory are sampled uniformly at random to batch-update the value function through gradient descent. First, this randomisation breaks correlations between consecutive samples. Second, observed sequences can be used multiple times for training. Third, training variance is reduced since the observations are not dominated by the maximisation step in the Q-learning update any more. Naive Q-learning can often run into feedback loops where training is dominated by samples from one state region. Off-policy training from the replay memory averages out experiences from many uncorrelated observations. The structure of the experience replay buffer itself has since been the topic of further research into the question of how to prioritise the most relevant experiences when sampling [62, 30].

The second fundamental insight concerns the so-called Q-target (i.e. $r + \gamma\ max_{a'} Q(s_{t+1}, a'; \theta)$. As noted in the introduction of Q-learning, the algorithm bases its estimates for the current Q-value of a state based on its own estimate of future values. This can cause divergence and oscillation if the agent gets stuck in feedback loops, e.g. due to noise in the environment. Deep reinforcement learning hence utilises a *fixed Q-target*. The value function parameters θ_{i-1} from previous iterations are hence stored and used for the maximisation step. Practically, this means two value functions are maintained in memory. One which is being updated through sampling the replay memory, one with fixed parameters which is used in the update step to determine the action to take. The parameters of the fixed value function are periodically updated by copying the parameters of the trained function, i.e. the weight matrix of the neural network. The combination of experience replay and fixed Q-targets provides learning stability and accelerates convergence [56, 57].

Algorithm 1 shows the control flow as described by Mnih et al. Training is performed in episodes where one episode is one Atari game played until won or lost. Further, there are some mechanisms specific to Atari games. i.e. a function ϕ that combines multiple frames from the simulator into a single state as a pre-processing step (to ensure the Markov property in the state representation). The value function takes as input the state and has one output neuron for every possible joystick action in the Atari domain (up to 18). Actions can hence be selected efficiently by maximising over a relatively small output array. The network is updated by back-propagating the loss for the action taken and setting the loss to 0 for all other actions. The same method can hence not be applied straightforwardly to continuous control tasks. Overall, deep reinforcement learning provides a framework to utilise recent advances in training powerful deep neural networks within classic RL algorithms.

Algorithm 1 Initial deep reinforcement learning algorithm as introduced by Mnih et al.

Initialise replay memory D to capacity N
Initialise action-value Q-function with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = x_1$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select random action a_t
    Otherwise select $a_t = max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t and observe reward r_t and image x_t
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition in replay memory
    Sample uniformly at random from D: (s_j, a_j, r_j, s_{j+1})
    Set

$$y_i = \begin{cases} r_j, & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \; max_{a'} Q(s_{t+1}, a'; \theta), & \text{for non-terminal } \phi_{j+1} \end{cases}$$

    Perform an gradient descent step on $(\gamma_j - Q(s_j, a_j, \theta))^2$
  end for
end for

3.3.2 Continuous deep reinforcement learning

Several methods have been suggested to apply typical continuous control algorithms (e.g. policy-gradient) to DRL. Duan et al. have recently benchmarked a number of continuous DRL algorithms to establish performance standards in this nascent field [33]. It should be noted that this sub-domain is evolving quickly right now: Duan's survey appeared in April 2016 and some of the algorithms in there (e.g. deep deterministic policy gradients [48]) have already been surpassed by newer methods, despite just being introduced in late 2015. This is not surprising in an emerging research avenue but makes it nonetheless difficult to establish the relevance of particular methods going forward. I will hence just describe a recent method I adopted for my own experiments, which was introduced at *ICML 2016* in April by Gu et al.: normalised advantage functions (NAFs) [37].

As noted above, the key question in continuous control algorithms is efficient action selection in high-dimensional spaces. This is often achieved through an actor-critic model: Actions of the agent are chosen by a separate actor policy π. The critic learns a value function to determine whether the actions have increased or decreased expected value and uses this error to adjust the actor's parameters. Hence, no explicit maximisation step is necessary to select an action as it can be directly sampled from the policy

[41, 38]. This however requires to train two separate neural networks. Normalised advantage functions (NAF) are utilise a single neural network to output both a value function and a policy, thus allowing more efficient training.

The concept of advantage functions is a technique for variance reduction. By subtracting a baseline function from the policy gradient and thus rescaling it, variance is reduced without changing the expected value. This is often achieved by subtracting the state-value function $V(s)$ from the Q-function:

$$\begin{aligned} Q(s,a|\theta^Q) &= A(s,a|\theta^A) + V(s|\theta^V) \\ A(s,a|\theta^A) &= -\frac{1}{2}(a-\mu(s|\theta^\mu))\mathbf{P}(s|\theta^P)(a-\mu(s|\theta^\mu)) \end{aligned} \tag{3.1}$$

The intuition behind this formulation is to define an analytical term which allows to directly determine the action maximising the Q-function. In this case, the advantage term is quadratic and defined so that setting $a = \mu(s|\theta^\mu)$ always yields the maximising action, which has an advantage of zero and negative advantage for all other actions. P is a state dependent matrix given by

$$P(s|\theta^P) = L(s|\theta^P)L(s|\theta^P)^T, \tag{3.2}$$

where $L(s|\theta^P)$ is a lower-triangular matrix specifying the shape of the quadratic function in each action dimension. The key point here is that the entries of $L(s|\theta^P)$, μ and V are all outputted by a single neural network. Updates are performed similar to the original DRL algorithm but instead of computing the Q-target through maximisation, the value function V is used. V is defined as

$$V^\pi(s_t) = \mathbb{E}_{r_{i\geq t}, s_{i\geq t}\sim E, a_{i\geq t}\sim\pi}[R_t|s_t, a_t], \tag{3.3}$$

or the expected reward from following the current policy from the current state. The update is hence given as

$$r_i + \gamma V'(s_{i+1}|\theta^{Q'}). \tag{3.4}$$

The loss can then be expressed as the average squared temporal difference over N observations:

$$L = \frac{1}{N}\sum_i(\gamma_i - Q(s_i, a_i|\theta^Q))^2. \tag{3.5}$$

Note that the different superscripts on θ all denote variables parametrised by the same neural network. The different outputs of the network are updated by deriving the gradient of the loss function with regard to each output and then back-propagating the loss through the network.

In summary, DRL can be described as a framework for utilising powerful neural network training mechanisms in a number of discrete and continuous RL algorithms. Note that the formulations above practically often require extensive (both with regard to time and resource requirements) training periods in an episodic setting where the task can easily be parallelised, e.g. by starting multiple instances of the Atari simulator. Following this introduction, I will now describe initial work on the thesis proposal.

3.4 Cortado: Monte Carlo simulation for data management

The reinforcement learning community has developed a common set of tasks and simulators to evaluate and compare algorithms. Classic examples include grid world problems, where an agent needs to navigate on a grid around obstacles towards an exit, or the mountain car problem, which requires an agent to accelerate a car from a valley to reach a hilltop. These early RL tasks were typically characterised by discrete, low-dimensional state spaces and few continuous or discrete actions. Recently, the research community has moved towards more complex problems, such as Atari video games [56], which can showcase the utility of deep neural networks as value function approximators in noisy environments with delayed reward sequences.

Monte Carlo simulations allow for a cheap and repeatable analysis of systems dynamics as well as algorithmic properties. It is hence apparent that in order to evolve the use of sophisticated learning techniques, powerful learning task simulation tools for learning in systems are necessary. The cloud computing and database community has traditionally relied on a common set of benchmark workloads to compare system performance. In particular, the database community often relies on the TPC benchmark suite, which comprises a number of typical database use cases such as online transaction processing (TPC-C) [45, 27]. In the cloud computing community, the Yahoo! Cloud Serving Benchmark (YCSB) (as described in section 1.3) has become the de-facto standard for performance comparisons between No-SQL (or New-SQL) data services [22]. YCSB not only provides workloads, but also a software framework containing an interface to add new databases, and a workload generating mechanism.

In a first order approximation, YCSB could be extended to analyse performance of RL tasks in data management. However, I believe the YCSB framework is unfit to be the basis for a simulation tool similar to the Atari simulator. The workload generating mechanism of YCSB is a drastic simplification of cloud-service request dynamics that is viable for throughput and latency comparisons at best. This is because in YCSB, a client cor-

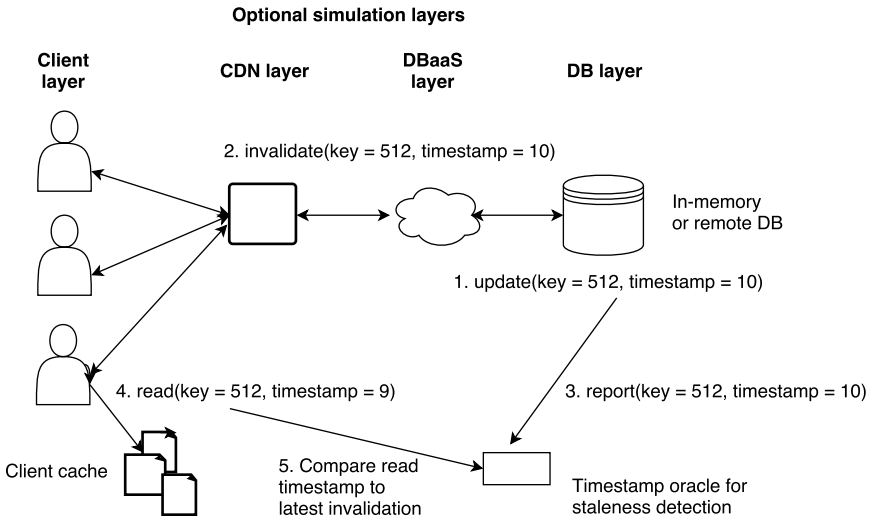

Figure 3.1: Example Cortado architecture with optional caching and DBaaS layers. The DBaaS layer serves as a collection of any middle ware logic that is executed besides simple store and load functionality, e.g. caching logic, server side machine learning.

responds to a single thread sequentially executing synchronous requests. This is a particularly unrealistic assumption for browser access, where content for a single client is retrieved via multiple asynchronous connections. I hence abandoned the YCSB design to create a new benchmarking and simulation tool based on the work done in the *Quaestor* project.

My simulation tool, which I preliminarily call *Cortado*, is based on a pipeline of completable futures, which is the Java 8 construct for asynchronous task completion. Figure 3.1 illustrates *Cortado's* layer architecture. Clients are not threads but objects with their own connection pool and client cache similar to a browser cache. Clients can perform CRUD (create, read, update, delete) requests, queries and transactions against the next processing layer, which can be a CDN, a DBaaS which does additional processing or any simulated or available database that implements the backend interface. Every layer can insert probability distributions or constants to simulate round-trip latencies. All requests are executed from the same thread pool. When scaling out clients and connection counts, processing power on a particular machine might be insufficient to see the expected throughput scale up if the thread pool executor cannot keep up with the request rate. The simulation hence supports *simulation time*, which relatively slows down simulated latencies and delays until the thread pool executor can keep up again.

In summary, *Cortado* provides YCSB functionality with request dynamics that support a more detailed analysis of the impact of various server

parameters. *Cortado* provides statistics on latency for individual request types, throughput, client cache hit rates, CDN cache hit rates, staleness at the client and CDN, and request invalidations. Through the work on the *Quaestor* project, *Cortado's* prediction accuracy compared to a distributed implementation on AWS EC2 could be confirmed for various metrics. Further, a master's student in Germany (associated with *Quaestor* through being supervised by my collaborators) has added scheduling and transaction functionality. This enabled him to utilise *Cortado* to illustrate the impact of client-side caching on optimistic transaction abort rates.

First, *Cortado* could be published as a stand-alone benchmarking open source project at some point, e.g. by implementing the database interface for a number of typical data stores and by comparing their benchmarks to YCSB, which should illustrate that *Cortado* can provide a superset of YCSB functionality with additional insights into staleness and caching behaviour of typical browser caches. It would be straightforward to extend *Cortado* to support concepts such as simulated replication on the backend, e.g. to analyse staleness in the case of geo-replicated databases. *Cortado* is implemented in about 8,500 lines of Java 8 with 80% of the code written by me and the remainder stemming from extensions by the master's student.

The main purpose of *Cortado* is to provide a test-bed for various typical control problems in data management. In the following section, I will introduce a case study implemented on top of *Cortado*.

3.5 Case study: Learning query result TTLs

A key problem in the *Quaestor* project was the estimation of dynamic time-to-lives (TTLs) for query results. To briefly summarise the relevant sections from the attached paper, estimating precise TTLs is necessary to reduce staleness and false positive rate in the Bloom filters at clients and servers. Overestimating TTLs for query results causes stale reads. Underestimating TTLs increases latencies by reducing cache hit rates.

In *Quaestor*, query result set TTLs are determined by estimating the Poisson process of incoming writes on individual keys. For a Poisson process, the inter-arrival times of events have an exponential cumulative distribution function (CDF), i.e. each of the identically and independently distributed random variables X_i has the cumulative density $F(x;\lambda) = 1 - e^{(-\lambda x)}\ for\ x \geq 0$ and mean $1/\lambda$. For each database record, *Quaestor* can estimate (through sampling) the rate of incoming writes λ_w in some time window t.

The result set Q of a query of cardinality n can then be regarded as a set of independent exponentially distributed random variables $X_i, \ldots, X_n$ with different write-rates $\lambda_{w1}, \ldots, \lambda_{wn}$. Estimating the TTL for the next

Table 3.1 High-level description of RL model for TTL estimation.

State	Write frequency estimation for each key in the query result
Network input	One input neuron per database key
Action	Single continuous TTL in seconds
Reward	Inverse cache miss and invalidation rate estimates

change of the result set requires a distribution that models the minimum time to the next write, i.e. $min\{X_1, \ldots, X_n\}$, which is again exponentially distributed with $\lambda_{min} = \lambda_{w1} + \ldots + \lambda_{wn}$. The quantile function then provides estimates that have a probability p of seeing a write before expiration:

$$F^{-1}(p, \lambda_{min}) = \frac{-ln(1-p)}{\lambda_{min}}. \tag{3.6}$$

Alternatively, the expected value of the next incoming write can be used. The calculation above is a typical example of a hand-crafted controller based on a statistical assumptions. I was hence interested in exploring a deep RL controller and compare its performance against the statistical estimator.

Most existing DRL research projects are implemented in Python or Lua to interface popular deep learning libraries, e.g. Torch or more recently TensorFlow [21, 2]. I found the available code often to be too specific towards problems like Atari games while also not built with concurrency in mind. I have hence implemented a prototype of a generic deep reinforcement learning framework in Java. This ensures compatibility with both *Cortado* as well as with a large number of data processing frameworks for future case studies, e.g. various Hadoop projects. I have relied on *deeplearning4j* (DL4J), an open source suite for deep learning in Java, which provides common neural network implementations and fast matrix computations (which is achieved by executing natively compiled C++ code) [31].

Currently, my RL framework supports the original deep Q-learning for discrete actions as well as normalised advantage functions (NAF) for discrete or continuous actions. By instantiating a controller, users have to pass the state and action dimensions and typical parameters such as learning rate and network depth. Table 3.1 describes the learning model for TTLs.

The input dimensionality n_{in} of the neural network depends on the average number of results returned by the server. If result set cardinality is not equal to value function input dimensionality, inputs can be left 0 or only the highest write rates are fed into the network (by sorting the vector of approximated write rates). Rewards for TTLs are not directly observable. Servers cannot measure stale reads or cache hits at the client and streaming them would incur too much traffic. On the other hand,

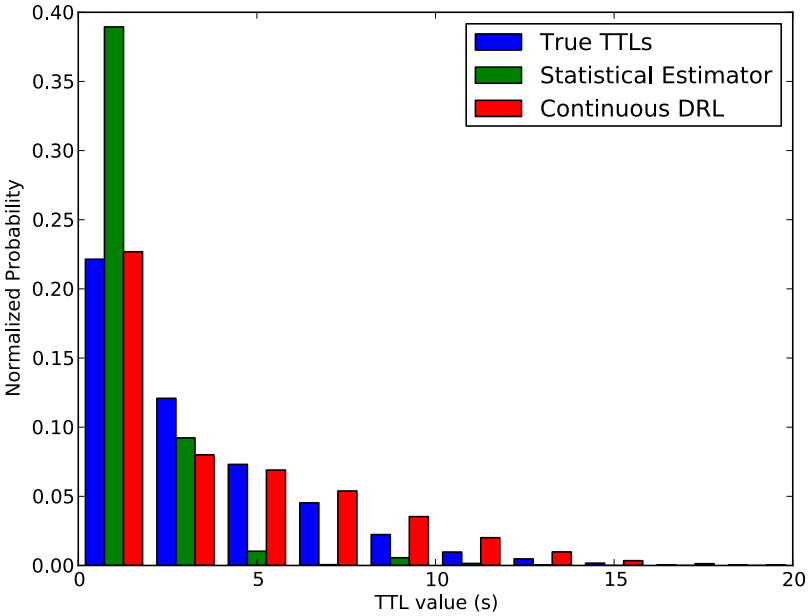

Figure 3.2: TTL estimation comparison. True TTLs can be measured in the simulation by considering the interval between serving the query and invalidation.

assuming a temporarily stable request rate, cache misses and invalidations at the server can be used as a proxy for client latency and staleness (since invalidations create potential for stale reads). A linear sum of weighted rewards can hence be used to direct learning towards different service level objectives, i.e. by favouring latency or tighter bounds on staleness.

Figure 3.2 shows results from running *Cortado* for 5 minutes with 10 clients (6 connections each). The "true" TTL of a query is the time between serving the query and invalidating the query after an update. Notably, no off-line training took place. This is possible because model dimensions are magnitudes smaller than when training from visual input. Batch-updates of NAFs were executed asynchronously.

The initial results show that DRL can smoothly approximate the TTL distribution while the assumptions in the statistical estimator seem to introduce a systematic underestimation. Note that the DRL seems to slightly overestimate the occurrence higher TTLs, which is likely due to exploration. It should also be noted that given enough manual tuning, the statistical estimator could likely be improved to perform better on this specific experiment. However, the dynamic nature of cloud infrastructures would likely require adjustments for every new deployment. In contrast, the DRL estimator should be able to adapt to new workloads, which I am planning to

evaluate in detail in the coming months.

My initial performance benchmarks show that decision queries (forward passes through the value function) can easily support the request throughput of a single database node with around 40,000 TTL estimates per second on a commodity laptop. However, batch updates on the same machine slow down the framework to around 4,000 operations per second. An obvious performance improvement comes from adjusting the update frequency depending on whether the model still significantly changes. I am hence currently designing experiments to be able to temporally trace and visualise the learning process in this case study. As a result, I aim to devise a fast, asynchronous NAF controller that adjusts learning frequency and batch size based on load patterns.

While I believe these initial results are a promising proof of concept, they have also raised a variety of new issues, which I believe have not been explored sufficiently by the (deep) RL community. Typically, reinforcement learning algorithms are evaluated on sample complexity and solution quality. In the seminal work on learning to play Atari games or Go, up to hundreds of GPUs have been used for training over multiple weeks. Data management problems are characterised by much smaller problem dimensions but higher decision query volumes with a large degree of concurrency as well as resource and latency constraints. The contribution of my doctoral work will be an understanding of this systems dimension of deep reinforcement learning. The goal is to deliver an algorithmic method and a software framework for model-free learning of runtime parameters. In the following, I will outline a research plan and a schedule towards this goal.

3.6 Research plan and schedule

The following list covers a research schedule from July 2016 to September 2018:

1. **July 2016 - October 2016** First, I plan to spend more time with the TTL case study exploring various issues such as understanding in detail the memory and runtime overhead of the system and extending *Cortado* to capture these metrics. I am particularly interested in understanding how to alleviate runtime bottlenecks in the update step. Further, I am interested in exploring other value functions besides neural networks and compare their performance.

   DRL as an online request configuration mechanism can only work if learning and decision queries can be performed timely, e.g. adding minimal constant latency at the backend. This low latency requirement is not present for more established DRL problems, as typical

simulators only generate modest frame rates (below 100 fps) compared to thousands of concurrent requests per second on web services.

Currently, the learning agent and *Cortado* are hosted on the same system and can be linked into the same virtual machine. This is not always practical and providing a server-implementation for a standalone DRL service will likely be necessary to isolate and understand performance on a separate machine. I plan to submit the results of the TTL case study to the *NIPS Deep Reinforcement Learning Workshop* in form of a position paper motivating the topic of DRL in cloud data management (Deadline likely in October, Workshop in December).

2. **November 2016 - January 2017** A particularly interesting modelling problem in the context of request volume scalability is the analysis of state approximation accuracy.The TTL case study relied on the availability of a write rate approximation for keys. In typical reinforcement learning research projects, the state itself is often presumed to be straightforward to capture. The systems applications of RL discussed earlier usually rely on high-level systems properties which are readily available from standard application monitoring. More theoretical papers use toy problems to illustrate specific algorithmic aspects and rarely concern themselves with scalability of the state representation itself, which is distinct from state and action space scalability (achieved through parametric models and non-linear function approximation).

   Consider per-key write rate approximation for query result TTLs. Even if every write time stamp is naively captured and stored forever, it's not clear how to weigh more recent writes compared to older ones, e.g. through an exponential moving average. In my proof-of-concept experiment, I have used a time-windowed queue. However, maintaining such data structures on a key-level is an unreasonable assumption with regard to memory overhead. However, there are several low-overhead alternatives available, e.g. counting Bloom filters and stream summary data structures such as count-min sketches [14, 15, 24, 25, 26]. The key question is to what extent reducing the state approximation precision affects prediction quality and whether one can hence come up with a closed-form model for trading off runtime overhead in maintaining state metrics against prediction quality. A related point of interest here is the utilisation of recurrent neural networks to deal with partial observability [66, 46].

3. **February 2017 - June 2017** Following these analyses, I plan to conduct another case study at scale on a typical data management problem. This could either be in the domain of latency-consistency trade-offs, e.g. managing client-side consistency (similar to Terry et al. with Pileus [72]) quorum-based staleness in replicated data stores (e.g. by implementing controllers for managing probabilistically bounded staleness as described by Bailis et al. [8, 9, 10]. This is particularly interesting in the context of driving the learning agent through SLAs, e.g. consistency SLAs.

   Closely related is the topic of transactional correctness, e.g. leveraging runtime information to select optimistic or pessimistic protocols per partition/collection/table. Instead of letting developers perform client-side invariant analysis to facilitate faster, coordination-avoiding transaction execution [5, 6], developers should be able to assume correct execution and execution optimisations should be omitted from them [7]. This paradigm has also been adopted by organisations such as Google in their Spanner database to accelerate development [23].

   The choice of case study will depend on the insights gained on performance and scalability in the TTL study. Further application domains can be found in distributed data processing frameworks such as flow control in stream processing systems like Storm [4, 78] and resource-constrained task placement on execution engines like Spark or TensorFlow. The results of this work could be submitted to a systems venue such as *EuroSys* or *SIGMOD*.

4. **July 2017 - September 2017** I plan to spend the summer of my second year at an industrial placement. Ideally, an internship would facilitate insights into machine learning at scale.

5. **October 2017 - December 2017** At this point, I plan to condense the algorithmic and practical insights gained from prior case studies into a general DRL architecture for data management. The focus will be on providing an actionable understanding of algorithmic limitations and the necessity of offline training or simulation for tasks where online exploration is undesirable. Another point of interest is to investigate offloading of the asynchronous training step to GPUs, which has not been necessary for the small models used in the TTL case study but is likely beneficial for processing much larger batches. Recent work has also investigated the acceleration of multi-core CPU training as compared to utilising GPU clusters [54].

6. **January 2018 - April 2018** Towards the end of my doctoral work, I plan to conduct another case study incorporating all aspects mentioned above as well as exploring other concepts such as lightweight

model representations through compression (e.g. binarised or integer weights) [39]. The case study will likely be selected from one of the topics mentioned above or a related one.

7. **May 2018 - September 2018** Dissertation write-up and submission.

Bibliography

[1] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. *Software available from tensorflow. org*.

[2] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[3] Eiman Alotaibi and Biswanath Mukherjee. A survey on routing algorithms for wireless ad-hoc and mesh networks. *Computer networks*, 56(2):940–965, 2012.

[4] Frank McSherry Andrea Lattuada and Zaheer Chothia. Faucet: a user-level, modular technique for flow control in dataflow engines. BeyondMR 2016.

[5] P.a Bailis, A.a Davidson, A.b Fekete, A.a Ghodsi, J.M.a Hellerstein, and I.a Stoica. Highly available transactions: Virtues and limitations. *Proceedings of the VLDB Endowment*, 7(3):181–192, 2013.

[6] Peter Bailis, Alan Fekete, Michael J Franklin, Ali Ghodsi, Joseph M Hellerstein, and Ion Stoica. Coordination Avoidance in Database Systems (Extended Version). In *Proceeedings of the VLDB Endowment*, volume 8, 2015.

[7] Peter Bailis, Alan Fekete, Michael J. Franklin, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Feral Concurrency Control. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1327–1342, 2015.

[8] Peter Bailis and Ali Ghodsi. Eventual consistency today: limitations, extensions, and beyond. *Communications of the ACM*, 56(5):55–63, 2013.

[9] Peter Bailis and Shivaram Venkataraman. Probabilistically bounded staleness for practical partial quorums. *Proceedings of the VLDB Endowment 5.8*, pages 776–787, 2012.

[10] Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. Quantifying eventual consistency with pbs. *Commun. ACM*, 57(8):93–102, August 2014.

[11] Salman a. Baset. Cloud SLAs. *ACM SIGOPS Operating Systems Review*, 46(2):57, 2012.

[12] Yoshua Bengio, Aaron Courville, and Pierre Vincent. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1798–1828, 2013.

[13] Giuseppe Bianchi, N Blefari Melazzi, Alberto Caponi, and Andrea Detti. A general, tractable and accurate model for a cascade of caches. *arXiv preprint arXiv:1309.0718*, 2013.

[14] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[15] Flavio Bonomi, Michael Mitzenmacher, Rina Panigrahy, Sushil Singh, and George Varghese. An improved construction for counting bloom filters. In *Algorithms–ESA 2006*, pages 684–695. Springer, 2006.

[16] Justin A Boyan and Michael L Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in neural information processing systems*, pages 671–671, 1994.

[17] Wouter Caarls, Eduardo Hargreaves, and Daniel S Menasché. Q-caching: an integrated reinforcement-learning approach for caching and routing in information-centric networks. *arXiv preprint arXiv:1512.08469*, 2015.

[18] Raffaele Chiocchetti, Diego Perino, Giovanna Carofiglio, Dario Rossi, and Giuseppe Rossini. Inform: a dynamic interest forwarding mechanism for information centric networking. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, pages 9–14. ACM, 2013.

[19] Raffaele Chiocchetti, Dario Rossi, Giuseppe Rossini, Giovanna Carofiglio, and Diego Perino. Exploit the known or explore the unknown?: hamlet-like doubts in icn. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, pages 7–12. ACM, 2012.

[20] Cisco. Cisco routing basics. http://docwiki.cisco.com/wiki/Routing_Basics, May 2016.

[21] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. Torch: a modular machine learning software library. Technical report, IDIAP, 2002.

[22] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with YCSB. *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, pages 143–154, 2010.

[23] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's globally distributed database. *ACM Trans. Comput. Syst.*, 31(3):8:1–8:22, August 2013.

[24] Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Found. Trends databases*, 4(1–3):1–294, January 2012.

[25] Graham Cormode and S Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. In *LATIN 2004: Theoretical Informatics*, pages 29–38. Springer, 2004.

[26] Graham Cormode, Vladislav Shkapenyuk, Divesh Srivastava, and Bojian Xu. Forward decay: A practical time decay model for streaming systems. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 138–149. IEEE, 2009.

[27] Transaction Processing Performance Council. Tpc benchmark c, standard specification version 5, 2001.

[28] Henggang Cui, Hao Zhang, Gregory R Ganger, Phillip B Gibbons, and Eric P Xing. Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In *Proceedings of the Eleventh European Conference on Computer Systems*, page 4. ACM, 2016.

[29] V. Dalibard and E. Yoneki. Optimizing complex computer systems with structured bayesian optimization (poster). https://www.cl.cam.ac.uk/ vd241/pubs/SBO_abstract.pdf, 2015.

[30] Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuška. The importance of experience replay database composition in deep reinforcement learning.

[31] Deeplearning4j Development Team. Deeplearning4j: Open-source distributed deep learning for the jvm, apache software foundation license 2.0. http://deeplearning4j.org.

[32] Li Deng and Dong Yu. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, 7(3–4):197–387, 2014.

[33] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. April 2016.

[34] The Apache Software Foundation. Apache Cassandra. http://cassandra.apache.org.

[35] Johannes Fürnkranz. Recent advances in machine learning and game playing. *ÖGAI Journal*, 26(2):19–28, 2007.

[36] AndreyV. Gavrilov and Artem Lenskiy. Mobile robot navigation using reinforcement learning based on neural network with short term memory. In De-Shuang Huang, Yong Gan, Vitoantonio Bevilacqua, and JuanCarlos Figueroa, editors, *Advanced Intelligent Computing*, volume 6838 of *Lecture Notes in Computer Science*, pages 210–217. Springer Berlin Heidelberg, 2012.

[37] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. March 2016.

[38] Roland Hafner and Martin Riedmiller. Reinforcement learning in feedback control. *Machine learning*, 84(1-2):137–169, 2011.

[39] Itay Hubara, Daniel Soudry, and Ran El Yaniv. Binarized neural networks. *arXiv preprint arXiv:1602.02505*, 2016.

[40] Jin Kyu Kim, Qirong Ho, Seunghak Lee, Xun Zheng, Wei Dai, Garth A Gibson, and Eric P Xing. Strads: a distributed framework for scheduled model parallel machine learning. In *Proceedings of the Eleventh European Conference on Computer Systems*, page 5. ACM, 2016.

[41] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *NIPS*, volume 13, pages 1008–1014, 1999.

[42] Gautam Kumar, Ganesh Ananthanarayanan, Sylvia Ratnasamy, and Ion Stoica. Hold them or fold them? aggregation queries under performance variations. Technical report, UC Berkeley TR UCB/EECS-2015-267, 2015.

[43] Shailesh Kumar and Risto Miikkulainen. Dual reinforcement q-routing: An on-line adaptive routing algorithm. In *Artificial neural networks in engineering*, 1997.

[44] Shailesh Kumar and Risto Miikkulainen. Confidence based dual reinforcement q-routing: An adaptive online network routing algorithm. In *IJCAI*, volume 99, pages 758–763. Citeseer, 1999.

[45] Scott T. Leutenegger and Daniel Dias. A modeling study of the tpc-c benchmark. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 22–31, New York, NY, USA, 1993. ACM.

[46] Xiujun Li, Lihong Li, Jianfeng Gao, Xiaodong He, Jianshu Chen, Li Deng, and Ji He. Recurrent reinforcement learning: A hybrid approach. *arXiv preprint arXiv:1509.03044*, 2015.

[47] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2719–2727, 2015.

[48] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. September 2015.

[49] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, DTIC Document, 1993.

[50] Glenn F Matthews and Khaled Rasheed. Temporal difference learning for nondeterministic board games. In *IC-AI*, pages 800–806, 2008.

[51] David Meisner, Brian T Gold, and Thomas F Wenisch. Powernap: eliminating server idle power. In *ACM Sigplan Notices*, volume 44, pages 205–216. ACM, 2009.

[52] David Meisner, Christopher M Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F Wenisch. Power management of online data-intensive services. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 319–330. IEEE, 2011.

[53] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *arXiv preprint arXiv:1505.06807*, 2015.

[54] Volodymyr Mnih, Adri Puigdomnech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. February 2016.

[55] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[56] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[57] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

[58] Paritosh Puri and Mrigendra Pratap Singh. A survey paper on routing in delay-tolerant networks. In *Information Systems and Computer Networks (ISCON), 2013 International Conference on*, pages 215–220. IEEE, 2013.

[59] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, and Kun Wang. A distributed self-learning approach for elastic provisioning of virtualized cloud resources. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, pages 45–54. IEEE, 2011.

[60] Carl Edward Rasmussen and Malte Kuss. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems 16*, pages 751–759. MIT Press, 2004.

[61] Michael Schaarschmidt. Towards latency: An online learning. mechanism for caching dynamic. query content. Master's thesis, University of Cambridge, 2015.

[62] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. November 2015.

[63] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[64] Prateek Sharma, Tian Guo, Xin He, David Irwin, and Prashant Shenoy. Flint: batch-interactive data-intensive processing on transient servers. In *Proceedings of the Eleventh European Conference on Computer Systems*, page 6. ACM, 2016.

[65] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[66] Ivan Sorokin, Alexey Seleznev, Mikhail Pavlov, Aleksandr Fedorov, and Anastasiia Ignateva. Deep attention recurrent q-network. December 2015.

[67] R S Sutton and a G Barto. Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998.

[68] Richard S Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 993–1000. ACM, 2009.

[69] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063, 1999.

[70] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[71] Doug Terry. Replicated data consistency explained through baseball. *Communications of the ACM*, 56(12):82–89, 2013.

[72] Douglas B. Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K. Aguilera, and Hussam Abu-Libdeh. Consistency-based service level agreements for cloud storage. *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles - SOSP '13*, pages 309–324, 2013.

[73] 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.

[74] 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.

[75] Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, March 1995.

[76] Gerald Tesauro, Rajarshi Das, Hoi Chan, Jeffrey Kephart, David Levine, Freeman Rawson, and Charles Lefurgy. Managing power consumption and performance of computing systems using reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1497–1504, 2007.

[77] Gerald Tesauro et al. Online resource allocation using decompositional reinforcement learning. In *AAAI*, volume 5, pages 886–891, 2005.

[78] The Apache Software Foundation. Apache Storm. https://storm.apache.org/.

[79] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993.

[80] John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *Automatic Control, IEEE Transactions on*, 42(5):674–690, 1997.

[81] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. Ernest: Efficient performance prediction for large-scale advanced analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 363–378, 2016.

[82] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[83] Philipp Wieder, Joe M Butler, Wolfgang Theilmann, and Ramin Yahyapour. *Service level agreements for cloud computing*. Springer Science & Business Media, 2011.

[84] M. Wiering and M. van Otterlo. *Reinforcement Learning: State-of-the-Art*. Adaptation, Learning, and Optimization. Springer Berlin Heidelberg, 2012.

[85] Cheng-Zhong Xu, Jia Rao, and Xiangping Bu. Url: A unified reinforcement learning approach for autonomic cloud management. *Journal of Parallel and Distributed Computing*, 72(2):95–105, 2012.

[86] G. Yen and T. Hickey. Reinforcement learning algorithms for robotic navigation in dynamic environments. In *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, volume 2, pages 1444–1449, 2002.

[87] Tingting Zhao, Hirotaka Hachiya, Gang Niu, and Masashi Sugiyama. Analysis and improvement of policy gradient estimation. In *Advances in Neural Information Processing Systems*, pages 262–270, 2011.