A Reinforcement Learning Approach to Optimize Performance of Stream Processors

Arnaud Dethise
KAUST
arnaud.dethise@kaust.edu.sa

1 INTRODUCTION

Real-time data analytics are commonly used by online services providers to gain insight from the large amount of data they collect. For example, a social network wants to detect the sudden popularity of a specific type of content, or an e-commerce company wants to verify the success of a flash promotional campaign.

Companies using real-time big data analytics systems have stringent requirements on their performance, and want to achieve both high throughput and low (tail) latency. Distributed stream processing systems such as Apache Storm [1] and Twitter Heron [13] are being increasingly used in the industry for their ability to quickly process streams of real-time big data and their scalability.

Available stream processing systems such as Storm and Heron have dozens of parameters that can be configured to change the behavior of the stream processor. The value of those parameters can drastically affect the performance of the application and must be tuned carefully. This configuration task is very challenging because the parameters have complex inter-dependencies and can be arranged in thousands of different configurations. An inappropriate configuration could potentially cause a violation of the Service Level Objectives (SLOs) or waste resources.

Another source of challenges in parameter tuning comes from dynamic changes in workload [4] and difference between stream processing applications [12]. A configuration that is optimal for a specific application and a given workload might be inefficient in a different environment, and automated systems might fail to adapt to some particular environments.

Figure 1 illustrates how parameter tuning can change the behavior and performance of a stream processor. This figure shows an example of a simple stream processing application that counts words in a stream of text (such as books or tweets) and outputs the most frequent ones over a tumbling window.

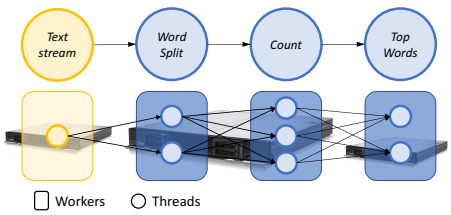

Figure 1: An example streaming application

The application is divided in four logical *stages* (Text Stream, Word Split, Count and Top Words). Within each of the stages, we can set the system parameters to change the number of workers and the degree of parallelism (represented by the colored shapes), the size of communication buffers (represented by the black arrows), and the mapping of workers to physical machines.

To perform this configuration task, one approach is to configure the system manually. However, this is a daunting and time-consuming process that requires hours of inspection and testing by the performance engineers [2, 8], as well as advanced domain-specific knowledge of the system and application from an expert performance analyst [14]. This costly task will also need to be repeated when the system and requirements change, as the best configuration depends on the infrastructure, application, workload and desired performance objectives.

Our objective is to attempt to build a tool to automatically solve this tuning task. We propose a new approach for stream processing system configuration by employing recent advances in machine learning. In particular, we follow the ideas from DeepRM [10] and use a deep reinforcement learning model to solve the configuration problem.

2 RELATED WORKS

A lot of research has been done in the area of parameter tuning for other systems such as databases, with iTuned [5] and OtterTune [14], and batch processing systems, with MROnline [9] and Ernest [15]. These categories of systems also benefit from other works in the literature that developed highly efficient query planners and fast approximate system models.

However, unlike databases and batch processing systems, automating the configuration and optimizing the performance of stream processors is an area that has seen little work, and the performance of this type of system is not well understood. Stream processors can support a wide variety of user-defined applications that have a difficult to predict behavior, and additional constraints arise from working with real-time data. Additionally, multiple previous works performing stream processor tuning only considered single metrics (such as optimizing the end-to-end latency) rather than multiple metrics (for example, optimizing the latency while maintaining a minimal throughput).

Different techniques have been proposed to automate the optimization of stream processing systems. Bilal et al. [3] developed a search-based framework for automated offline parameter tuning of stream processors, which applies iterative

optimization algorithms on measurements from actual runs of streaming applications. However, the drawback of this solution is that adapting to any change in workload needs to run the optimization process again, which takes from minutes to hours and requires a testing environment separate from the production systems. Floratou et al. [6] modified Heron to provide self-healing and self-tuning capabilities through a diagnose-and-resolve approach. However, this system requires human expertise in performance engineering to define the policies that are applied to resolve problems.

3 MACHINE LEARNING FOR SYSTEMS

Artificial intelligence offers many solutions that can be used to improve the performance of systems, and we believe it could apply to stream processors as well. Those solutions include for example search-based techniques, system modeling and performance prediction, and machine learning. We focused our research on machine learning because this approach makes it possible to reuse past observations and separate the training and testing phases, two features that are useful in always-running systems.

Several Machine Learning techniques have been used in the literature, such as Deep Reinforcement Learning [11], regression models [7] or Gaussian Processes [5]. In order to apply those techniques to stream processors, we will need to find the most appropriate ways to specialize them to our application domain, which presents multiple challenges such as encoding the measurements in a way that can represent the abstract system features, defining the space of possible decisions (i.e. the available configurations), and ensuring the convergence of our method. We expect to encounter significant challenges in the encoding of our very large action space and the effects of using non-linear utility functions.

Lastly, as explained in section 4, our current approach relies on using neural networks and deep reinforcement learning. Unlike black box optimization algorithms, using deep learning also requires designing a new efficient neural network architecture.

4 GOALS AND METHODOLOGY

Our goal is to automatically determine the optimal value of configurable parameters for a system. The definition of "optimal" depends on the use case and can be configured depending on the requirements. For example, a text processing application might want to favor high throughput; one that manages user requests would prefer low latency; and an application that has flexible constraints could favor configurations that minimize the cost.

Another goal is that our solution should be able to make online predictions that can be applied to production systems. This introduces two additional challenges: we cannot afford to run test configurations which violate the SLOs (we need to have reasonable confidence that a configuration is good, if not optimal, when it is installed), and our predictions need to be fast (in the order of seconds). To achieve this, we can pre-train a model in an offline phase (commonly called the *training phase* in machine learning), and leverage that knowledge to improve the prediction in the online phase (also called *testing phase*). Since applying a new configuration to a running system can incur an additional cost in performance, it is important to balance the frequency of new configurations with the cost of disturbing the system stability.

It is possible that the workload applied to the system quickly changes while it is running. For example, the arrival rate of new data could increase, or the data could suddenly become more relevant to the application and require more complex processing. Our solution needs to be capable of finding the new optimal configuration when the characteristics of the workload change.

Lastly, our solution should also be able to cope with new, unseen types of applications, workloads, and requirements.

To achieve those goals, we follow an approach based on deep reinforcement learning, inspired from successful work at automating resource allocation [10] and optimizing video streaming on networks with unpredictable bandwidth [11]. The context of those solutions is similar to ours as they work on systems that are complex and face dynamic workloads and different types of performance requirements.

By using machine learning, we will be able to leverage a large amount of performance metrics collected in the system. This data reflects the status and behavior of the application, but is difficult to understand and use directly. One of the strengths of neural networks is their ability to extract useful information without guidance, automatically detecting the relevance of hidden features and approximating the function of $application, workload \rightarrow performance$. This function is assumed to be arbitrary, potentially non-convex (with local maxima) and non-smooth, which is a challenge for some other machine learning techniques.

Another advantage of using reinforcement learning is that it removes the need for explicit performance measurements. Instead, we define a *reward function* that will be maximized independently of actual metrics. This approach focuses on optimizing the configuration quality depending on the objective rather than raw performance numbers.

Figure 2 illustrates the general structure of reinforcement learning systems. In this figure, the *reward* signal is a function of the performance defined arbitrarily during training, which our objective is to maximize. Our work will be to find the relevant *state* information to extract from the system and establish a structure that produces an appropriate *action* signal for system configuration.

We can use a corpus of workload traces during the training phase to infer the best configuration in a given environment without expressing the workload properties explicitly. The training corpus defines the various workloads our system can adapt to, and changing the corpus to redo the training allows us to adapt the optimizer to a new environment.

Figure 3 shows an example of the proposed architecture of the optimizer. In this architecture, we show three different categories of inputs (workload, external metrics, internal metrics) that are sent to the optimizer for performance prediction. In addition to this structure required for the online phase,

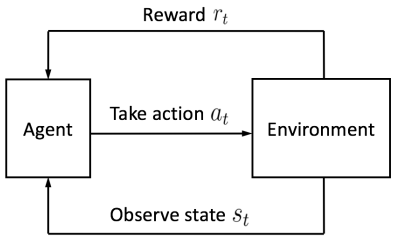

Figure 2: Overview of a Reinforcement Learning architecture

there exists a reward signal that is sent as feedback to the optimizer during the training.

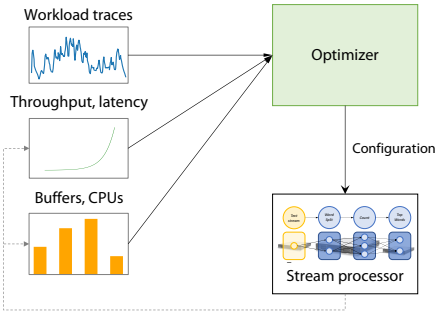

Figure 3: Proposed structure for the optimizer

The performance of our system will be evaluated by measuring the quality of generated configurations against other existing solutions using benchmarks that have been presented in the literature.

A major challenge in designing this system is that machine learning techniques, and in particular deep reinforcement learning techniques, need to collect a large amount of samples to train the model. Without a fast and accurate simulator, this data collection needs to be run on an actual topology, which is a time-consuming process. Two approaches could allow us to mitigate this drawback.

First, we could reuse measurements from past runs to speed-up the training of our model. While those measurements would lack accuracy due to different conditions, they can help bootstrap the training of the reinforcement learning model, particularly in the initial phase when the model hasn't collected enough data to avoid regions of the action space that yield very poor performance results. Optionally, this could also include measurements collected from running other similar jobs, although this requires us to prove that those jobs have a similar behavior when applying the same configuration.

The second approach is broader and based on the idea that a long training time can be justified if the model is able to adapt to a wide variety of applications. To this end, we would need to be able to accurately predict the optimal configuration even when facing unforeseen changes in workload or different applications without retraining the model.

5 RESEARCH OBJECTIVES

Our first research objective is to investigate whether machine learning techniques can be successfully applied to distributed system optimization to build efficient optimizers. Since those systems contain many components with complex interactions, it is very difficult and labor-intensive (if at all possible) to create accurate models of the system. Without a model, it is difficult to predict how the system will react to certain changes.

We will attempt to show that, provided a correct encoding of system measurements and available tuning knobs, machine learning techniques are capable of building an internal model of the system to extract the information relevant to optimizing the system and selecting a configuration that is (close to) the optimal.

We also want to characterize how the behavior of a stream processing system evolves when moving to a different application or changing the underlying system (such as system upgrades or changing network configurations). This would allow researchers to better characterize the adaptability and portability of their solutions and systems. In the field of machine learning in particular, this would also give us a way to predict when a model needs to be retrained in order to remain relevant to the application.

To this end, multiple types of difference need to be taken into account. We have already identified sources of uncertainty in applications, system configuration and updates, dynamic workloads and hardware.

Another objective is to extract knowledge from our machine learning agent to improve the understanding of stream processors. This requires us to adapt techniques of explainable artificial intelligence to our system and discover what information is most significant to improving the application performance. Those results would allow us to improve optimization techniques for systems. This approach could also be extended to other uses of machine learning for systems.

Finally, we will try to leverage the knowledge gained in the previous steps to build a new system that includes machine learning in its design. This self-tuning system could use neural networks to infer from measurements how to configure itself and tune its performance without external intervention.

REFERENCES

[1] Apache Storm. https://storm.apache.org/.
[2] Apache Storm performance tuners. https://www.ericsson.com/research-blog/apache-storm-performance-tuners/.
[3] M. Bilal and M. Canini. Towards automatic parameter tuning of stream processing systems. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, pages 189–200, 2017.
[4] N. Bruno and S. Chaudhuri. An online approach to physical design tuning. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 826–835, 2007.
[5] S. Duan, V. Thummala, and S. Babu. Tuning database configuration parameters with ituned. *Proceedings of the VLDB Endowment*, pages 1246–1257, 2009.
[6] A. Floratou, A. Agrawal, B. Graham, S. Rao, and K. Ramasamy. Dhalion : Self-Regulating Stream Processing in Heron. *Proceedings of the VLDB Endowment*, pages 1825–1836, 2017.
[7] H. Herodotou, F. Dong, and S. Babu. No One (Cluster) Size Fits All: Automatic Cluster Sizing for Data-intensive Analytics.

Proceedings of the 2nd ACM Symposium on Cloud Computing, pages 1–14, 2011.

[8] How Spotify Scales Apache Storm. https://labs.spotify.com/2015/01/05/how-spotify-scales-apache-storm/.

[9] M. Li, L. Zeng, S. Meng, J. Tan, L. Zhang, A. R. Butt, and N. Fuller. Mronline: Mapreduce online performance tuning. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, pages 165–176.

[10] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. Resource Management with Deep Reinforcement Learning. *HotNets '16*, pages 50–56, 2016.

[11] H. Mao, R. Netravali, and M. Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 197–210, 2017.

[12] M. Stonebraker, U. Çetintemel, and S. Zdonik. The 8 requirements of real-time stream processing. *SIGMOD Rec.*, pages 42–47, 2005.

[13] Twitter Heron. https://twitter.github.io/heron/.

[14] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1009–1024, 2017.

[15] S. Venkataraman, Z. Yang, M. J. Franklin, B. Recht, and I. 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.