Chapter 1

AI-driven performance management in data-intensive applications

Ahmad Alnafessah,[1] Gabriele Russo Russo,[2] Valeria Cardellini,[2] Giuliano Casale,[1*] and Francesco Lo Presti[2]

[1] *Department of Computing, Imperial College London, London, UK*
[2] *Department of Civil Engineering and Computer Science Engineering, University of Rome Tor Vergata, Rome, Italy*

*Corresponding Author: Giuliano Casale; g.casale@imperial.ac.uk

Data-intensive applications have attracted considerable attention in recent years. Business organizations are increasingly becoming data-driven and therefore look for novel ways to collect, analyze, and leverage the data at their disposal. The goal of this chapter is to overview some recurring performance management activities for data-intensive applications, examining the role that AI and machine learning are playing in enhancing practices related, among others, to configuration optimization, performance anomaly detection, load forecasting, and auto-scaling for this class of software systems.

Keywords: Performance management, AI, Data-intensive applications, batch analytics, streaming

1.1. Introduction

In recent years, the prominence of Big data has led to a growth in interest for developing intelligent data-intensive software systems in several application domains. Data-driven systems that can extract knowledge, plan, and adapt to events through processing, transformation, and analysis of datasets are thus increasingly widespread in both industry and society.

From a technical standpoint, data-driven software systems are often built by leveraging features such as batch analytics or streaming, now easily programmable through in-memory platforms such as Apache Spark, Hadoop/MapReduce, Storm, Flink, among others. We outline popular data processing platforms in Section 1.2. Although the combination of batch and streaming workloads enables richer functionalities, workload heterogeneity also means that achieving service levels objectives presents additional complexity in pinpointing causes of performance degradation and identifying tunings to address them. For example, performance metrics in data-driven software are difficult to predict as they often depend on data properties, such as volume or velocity, and frequently even on data type and content, making it difficult to reason about and tune system performance at design time. Furthermore, the combination of batch and streaming features in a software means that different system components will strive to achieve different performance goals, i.e., high-throughput and high-utilization for analytics features and low-latency for stream processing operators, making the process of runtime performance tuning a fairly heterogeneous and complex exercise.

To support these challenges, the goal of this chapter is to overview AI management techniques that are available in the literature to manage and tune the performance of data-intensive applications. AI methods offer considerable

simplicity and flexibility in choosing the features that drive the management process, in spite of some opaqueness in presenting the way the models reach decisions.

Compared to traditional management methods, which either leverage low-level system characteristics or use mathematical modeling abstractions, AI management methods leverage learning on experimental datasets that reduce dependence on assumptions and shift the attention from conceptual modeling to data-collection and model training. This offers considerable potential to increase the effectiveness of management methods in situations where the system behaves according to complex and unpredictable logic, as it is often the case for systems driven by external data.

Summarizing, in this chapter we examine the applicability of AI methods in the context of data-intensive applications. We survey in particular studies that illustrate the versatility of AI models when applied to popular data streaming and batch analytics platforms. Our aim in particular is to cover a broad spectrum of AI methods, in order to inform the reader on the range of learning techniques that may be applicable to recurring management problems involved in data-driven systems. We look at common management tasks such as platform configuration, workload forecasting, resource scaling, monitoring and detection of performance anomalies. We also give selected examples to build an intuition on their behavior, benefits and limitations.

1.2. Data-processing frameworks

In this section, we overview the essential features of common execution platforms in use to define data-intensive applications. A summary of the key char-

Table 1.1: Summary of the key characteristics of data-processing platform.

	Workloads		Processing style	
Platform	**Batch**	**Streaming**	**In-Memory**	**Disk-heavy**
Storm		✓	✓	
Hadoop/MR	✓			✓
Spark	✓	✓	✓	
Flink	✓	✓	✓	

acteristics of each platform is shown in Table 1.1. The section also highlights core performance management challenges associated to each of these platforms.

1.2.1. Apache Storm

Apache Storm[1] is a popular open source platform used for distributed real-time stream processing. The platform offers very low latency for dataflow processing, making it an ideal option for real time processing Toshniwal et al. [2014]. Storm dataflow topologies involves two main node types: spouts and bolts. A spout is the source of the data stream at the input queue and may generate data by itself Shahrivari [2014]. A bolt instead consumes the stream, operates transformations or computations, and ultimately produces an output stream as a result. Every task corresponds to one operating system thread.

Performance management of Storm applications frequently involves difficult decisions concerning optimal configuration options for spouts and bolts, ranging from decisions concerning buffer, message, and batch sizes, number of bolts, and selection of optimal waiting strategies. There is a limited understanding of the interplay between these parameters, posing intrinsic challenges

[1] http://storm.apache.org/

for optimal system configuration. Moreover, the Storm system does not automatically manage load balancing and resource scaling, thus requiring ad-hoc performance management techniques.

1.2.2. Hadoop MapReduce

Hadoop implements the MapReduce paradigm and it is a well-known example of batch processing platform. It is used for intensive Big Data applications starting from a single server and can scale up to thousands of machines [2].Usually, MapReduce uses an existing dataset that is stored in Hadoop Distributed File System (HDFS) before beginning to process batch data. Processing with native Hadoop can be paused or interrupted, but the dataset cannot be modified. This means that if current data is changed for any reason, the job needs to be run again.

Despite distinctive challenges arise in the area of optimal configuration of Hadoop platforms, over the years performance management has insisted in particular on the problem of detecting and handling straggler tasks, which falls into the general problem area of performance anomaly detection and mitigation. This is a result of the synchronizations between dataflow tasks that can block progress until straggler tasks complete their activities.

[2]https://hadoop.apache.org/

1.2.3. Apache Spark

Apache Spark[3] is a large-scale in-memory processing framework that can support both batch and stream data processing, which can make it easy with a low cost to support different types of workloads on the same engine in a production environment, such as those arising from graph analysis and machine learning applications. Compared to older solutions such as Hadoop, the main goal of Apache Spark is to speed up batch processing by utilizing in-memory computation. Thanks to a reduced use of intermediate storage of processing results, Spark is orders of magnitude faster than Hadoop for in-memory analytics.

Spark can be deployed over Hadoop as an alternative to MapReduce, as well as on Amazon EC2, Apache Mesos, or as a standalone cluster. In addition, it can access many data sources, including HDFS, Cassandra, HBase, Hive, Tachyon, and any Hadoop data source. Spark provides a general purpose engine for different kinds of computation, including iterative algorithms, job batches, streaming, and interactive queries. These different types of computation were previously difficult to find in the same distributed system Karau et al. [2015]. Beyond the ability to perform batch and stream processing, Spark also provides a rich library that is built on top of its core engine Meng et al. [2016].

In terms of performance management, Apache Spark has around 200 complex parameter configurations (e.g., executors, CPU cores, memory, shuffle behavior, compression), which may significantly impact the overall Spark system performance Herodotou et al. [2020]. The microarchitectural behaviors of Spark are different from those of other Big Data technologies. The Spark core data abstraction is the Resilient Distributed Dataset (RDD), which cannot be

[3] https://spark.apache.org/

modified and RDD can be executed in parallel on different nodes. In addition, Spark needs more advanced auto tuning solutions to boost its performance within production environment. Therefore, precisely performance management to optimally manage and auto-tune Spark is needed to increase the performance efficiency of such a complex system and immediately gain advantages of cost and time saving.

1.2.4. Apache Flink

Apache Flink Carbone et al. [2015] is another open source distributed processing engine designed for low-latency streaming computation. Analogously to Spark, Flink relies on in-memory computation and provides a unified API for processing both bounded and unbounded datasets. However, differently from Spark, where batching has a primary role, Flink has been designed with streaming in mind. Indeed, Flink applications are built upon the concepts of *streams* and *transformations*. Streams represent (possibly unbounded) data flows, while transformations are operations that, given one or more streams as input, output one or more streams as the result (e.g., filtering). A few higher-level libraries are built on top of these abstraction, easing the definition of common processing use cases (e.g., complex event processing, graph analytics).

At runtime, Flink applications are mapped to *streaming dataflows*, DAGs composed of processing nodes (often called *operators*), which implement transformations, connected by streams. For execution, Flink leverages a distributed architecture, designed according to the *master-worker* pattern. The master component is the *JobManager*, which coordinates distributed execution and

is responsible for application scheduling, checkpointing, and recovery in case of failure. The TaskManagers (i.e., the workers) execute the application *tasks* (i.e., instances of operators) and manage the data transfers between them.

Performance management of Flink applications, which are often long-running, mainly involves runtime deployment and resource adaptation. First of all, varying infrastructure conditions may require migrating operator tasks between computing nodes during execution. Flink supports migrating both stateless and stateful tasks through the *savepoint* mechanism, which ensures no loss of information. Moreover, workload variability requires dynamically scaling the parallelism of Flink applications and balancing the load across the cluster to keep consistent performance levels over time. To this end, load prediction techniques can be helpful to proactively adapt application configuration.

1.3. State of the art

We review in this section the existing techniques for dealing with the most relevant performance management issues in the context of data-intensive applications, with particular emphasis on AI-based approaches.

1.3.1. Optimal configuration

As a consequence of the availability of numerous configuration parameters, their optimization is a critical task in the domain of data-intensive systems. The goal of configuration optimization is to find the ideal configuration with respect to the system performance. Various automated parameter tuning methods have been proposed in the literature, which are discussed in the following sections .

1.3.1.1. Traditional approaches

Gunawan and Lau [2011] propose a parameter tuning framework based on Design of Experiments (DOE) approach. Their goal is to find an initial range of parameter values for automated tuning using a factorial experiment design to screen and rank all the parameters, so as to focus the search on the most influencing parameters. In addition, Gunawan and Lau [2011] examine Response Surface methodology, which is a model-based approach within DOE that can be used to quantify the effect of each parameter to find the most promising initial range for the vital parameter values. Their approach can be integrated with existing automated parameter tuning configuration, called ParamILS and Randomized Convex Search (RCS). Their method seams promising for both discrete and continuous parameter configuration settings.

Shi et al. [2014] introduce MRTuner from IBM, which is a tool to enable holistic optimization for MapReduce jobs. Their design uses an efficient search algorithm (Grid-based Search) to find the optimal execution plan. Around twenty configuration parameters are investigated to understand the relationships that have a noticeable impact on MapReduce performance. The tool is evaluated using HiBench on two Hadoop clusters. Their results show MRTuner has low latency and can find accurate execution plans.

Bilal and Canini [2017] examine an automatic parameter tuning framework for stream processing platforms. Gray-Box, Black-box analysis, and a rule-based optimization method are combined, and configuration parameters are initialized using Latin Hypercube Sampling. Hill Climbing Algorithm is used to explore the configuration space. Bilal and Canini [2017] evaluate using three benchmark applications within the Apache Storm streaming system. They find that rule-based can converge up to five times faster than other approaches,

making it suitable for parameter tuning within stream processing platforms.

1.3.1.2. AI approaches

A machine learning approach is used by Chen et al. [2015] to appropriately tune configuration parameters of Hadoop. Their approach has two stages, which are the prediction stage to estimate the performance of a MapReduce job and the optimization stage to repeatedly search for the optimal configuration parameters. The authors claim that their method can improve Hadoop performance up to eight times compared with traditional methods.

Wang et al. [2016] introduce a parameter tuning method based on binary classification and multi-classification for Apache Spark systems. Decision trees are used for auto-tuning of configurations with four different types of workloads, which are Sort, Wordcount, Grep and NavieBayes workloads from *BigDataBench* benchmark. Their experimental results show that the proposed method can improve Spark performance on average by 36% compared to default Spark configuration.

Hernández et al. [2018] optimize parallelism for data-intensive platforms using machine learning. They use Boosted regression trees are used as the authors claim that they have the lowest variance compared with other algorithms. In addition, they argue that decision trees are interpretable, which means that it is possible to quantify the impact of collected features on the overall performance. They evaluate proposed solution using a benchmark of 15 different Spark applications running on YARN. The results show that their task parallelization method is capable of improving the performance of Spark by 51%.

Bayesian Optimization (BO) is an effective and efficient method for auto-

tuning systems and machine learning algorithms. Joy et al. [2016] propose a framework that uses BO to tune hyperparameters of data-intensive applications. Their idea is dividing the data into small chunks with the same size to boost the search by applying BO tuning in parallel. To validate the performance of their framework, they use the proposed method to tune two machine learning algorithms, Deep Neural Networks (DNN) and Support Vector Machines (SVM). BO offers effective hyperparameters tuning with less computational overhead.

Jamshidi and Casale [2016] tackles the challenging issue of finding optimal configurations for a data-intensive streaming system by proposing auto-tuning methods that can help systems administrators to determine the near-optimal configurations with a limited budget of experiments. Their solution revolves around BO for configuration optimization, which utilizes Gaussian Processes (GP) to continuously capture posterior distributions of configuration space for the application. Their method works in a way that the optimal configurations will eventually be discovered. The authors validate the proposed method using a Storm clsuter in the cloud.

Yigitbasi et al. [2013] examine and explore a machine learning model to tune the configuration parameters of Hadoop and MapReduce. They use Support Vector Regression (SVR) to a smart search algorithm in terms of the effectiveness of parameter space exploration. Their results show that SVR obtains higher accuracy than Starfish auto-tuner, which uses a cost-based search model.

Liao et al. [2013] illustrate that the Hadoop platform has hundreds of configuration parameters that have very complicated interactions. This wide configuration space makes it time-consuming for system administrators to optimally

tune the Hadoop parameters. They provide an evaluation to automate the tuning process of Hadoop based on a cost-based and machine learning approach (neural network, SVR, multiple linear regression, and decision trees).

Di Sanzo et al. [2012] provide a study about auto-tuning cloud-based in-memory transactional data grids configuration by using a machine learning black-box approach. They use artificial neural networks (ANN) to optimize the dynamic selection of the amount of cache servers and the replication level of data objects to reduce the cost of cloud system operations. They conduct preliminary experiments based on a synthetic benchmark and a real data grid system that is run on Amazon EC2 virtual servers. The authors conclude that the ANN-based approach is effective for tuning transactional data grids. There are some additional works related to optimal configuration using CART tree Nguyen et al. [2018], long short-term memory (LSTM) Fang et al. [2019], regression trees, nearest neighbor Berral et al. [2015], and reinforcement learning Peng et al. [2017].

1.3.1.3. Example: AI-based optimal configuration

In this section, we illustrate the effectiveness of BO for finding optimal configurations by searching through the configuration space. The goal of BO is to utilize the prior knowledge and evidence to optimize the posterior at each evaluation step, to reduce the gap between the actual global optimization and expected optimization for the model Brochu et al. [2010]. Compared with traditional search algorithms (grid search, random search and manual tuning), Alnafessah and Casale [2020] show that how BO can facilitate parameter tuning with more parameters and fewer number of experiments to find optimal configurations.

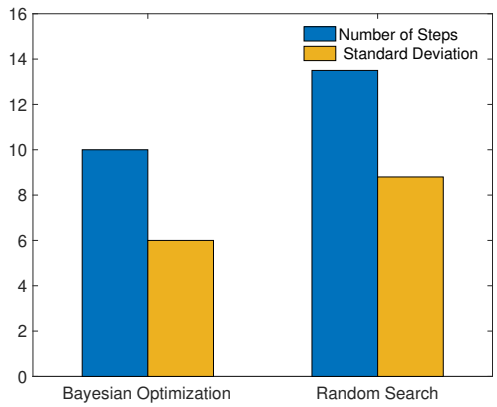

Figure 1.1: Comparison between Bayesian Optimization and Random Search to reach the highest F-score

BO is an ideal choice to find the optimal training dataset size and configuration parameters to efficiently and effectively train the anomaly detection model to achieve high F-score in a short period of time. Before applying BO, there are two main choices that need to be carefully make, which is the prior over functions and type of acquisition function Snoek et al. [2012]. We use Gaussian Processes (GP), which are stochastic processes defined by the property that any finite set of N points induces a multivariate Gaussian distribution Snoek et al. [2012], Shahriari et al. [2015]. They are are efficient for uncertainty estimation. We use the *Expected Improvement* acquisition function that Snoek et al. [2012] provide for configuration space with high uncertainty and high estimated value to evaluate a point x to sample based on the posterior distribution function to guide exploration. It can trade-off between exploration of the configuration search space and exploitation of current promising subspace.

Figure 1.1 shows a comparison between BO and random search in achieving the high performance training of the machine learning algorithm (ANNs

in our case) to efficiently detect anomalies (CPU, Cache Thrashing, and Context Switching) within Apache Spark Streaming datasets that Alnafessah and Casale [2020] provides with predefined F-score. For CPU anomaly detection, the BO can optimally train the anomaly detection model using 10 combinations of configurations, whereas the random search needs 14 combinations of configurations on average. In addition, BO outperforms the random search in training the AI model for detecting CPU, cache and context switching anomalies.

1.3.2. Performance anomaly detection

System performance is often described in terms of the time taken to process a set of tasks with a given amount of computing resources that are consumed within a given observation period Ibidunmoye et al. [2015]. The growing complexity and dynamicity of cloud systems and data-intensive technologies requires significantly higher levels of automation and significant attention Fu [2011]. Performance anomalies have become a major concern for developers and academic researchers particularly for Big Data and Artificial Intelligence technologies over cloud computing systems. Anomalous performance can occur as a result of service operator faults Oppenheimer et al. [2003], system failures, user errors Pertet and Narasimhan [2005], environmental issues, and security violations Ibidunmoye et al. [2015], among others.

Many anomaly detection studies are generic, while others are specifically conducted for certain application domains (e.g., data-intensive applications, networking, web based application, etc). There are studies that provide an overview about techniques that have been developed in traditional statistical

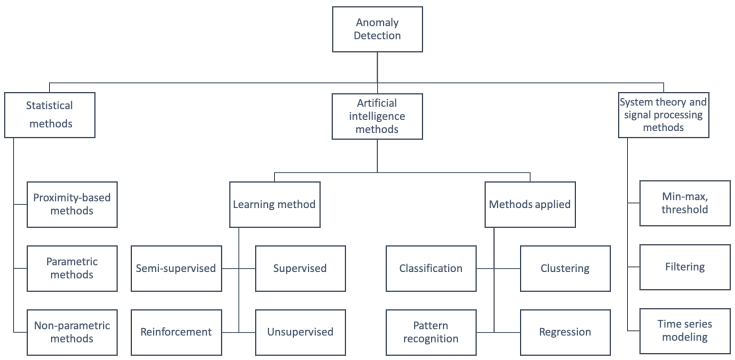

Figure 1.2: A taxonomy of anomaly detection techniques generalizing the ones in Sebestyen et al. [2018], Hodge and Austin [2004], Chandola et al. [2009].

approaches and machine learning techniques for anomaly identification Hodge and Austin [2004], Chandola et al. [2009], Ibidunmoye et al. [2015]. Figure 1.2 shows a taxonomy of existing anomaly detection techniques that is build on common taxonomy based on Sebestyen et al. [2018], Hodge and Austin [2004], Chandola et al. [2009].

1.3.2.1. Traditional approaches

Statistical techniques are the earliest approaches used for performance anomaly detection. Lu et al. [2017] use a statistical offline approach to detect abnormal Apache Spark tasks and analyze the root causes based on statistical spatial-temporal analysis. They use some features related to execution time, memory usage, garbage collection, and data locality of each Spark task to determine the degree of abnormal tasks. They use mean and standard deviation of all tasks in each stage to decide the threshold and to get information about macro-awareness on the task execution time. They analyze performance issues using

factor combination criteria for every performance anomaly based on weighted factors. They validate their method on a private Spark cluster and SparkBench Li et al..

Kelly [2005] examines how to get insights about performance issues of globally distributed systems using simple queueing theoretic observations together with standard optimization methods. The author obtain extensive empirical results from three distributed commercial production systems that serve real customers.

Yang et al. [2007] propose an anomaly detection and diagnosis solution within grid environments using statistical and signal processing approaches. Their work extends the traditional window-based strategy by using signal processing to filter out recurring background variations and determine which resource is the probable cause of an anomalous performance in a system. They use window averaging, which is a widely used statistical anomaly identification technique because it is simple and efficient. The anomalies are injected into three grid systems (Cactus, GridFTP, and Sweep3d) at random time intervals.

1.3.2.2. AI approaches

Research on automated anomaly detection is essential in practice because any late detection or slow manual resolution of performance anomalies in a real production environment may cause prolonged service-level agreement violations and significant financial penalties Dean et al. [2012], Tan et al. [2012]. This leads to a demand for performance anomaly detection solutions in cloud computing and data intensive systems that are both dynamic and proactive in nature Ibidunmoye et al. [2015]. The need to develop these methods for production environment with very different characteristics means that AI is

ideally positioned for system diagnosis to automatically identify performance anomalies. These techniques provide the capability to quickly learn baseline performance characteristics through a large monitoring metrics space in order to distinguish normal and anomalous patterns Rogers and Girolami [2016].

Classification techniques aim to determine whether the instances in a given feature space belong to a particular class or multiple classes Ibidunmoye et al. [2015]. There are popular classification techniques for anomaly identification, such as ANNs, SVM, and nearest neighbor. The classification technique is significantly affected by the accuracy of the labeled data and algorithms that have been used. For example, the training and testing processes for decision trees algorithms are usually faster than SVM, which involve quadratic optimization.

Alnafessah and Casale [2018] propose an ANN-driven methodology for anomaly identification, particularly for Apache Spark. The authors use a machine learning approach to quickly sift through Spark logs and system monitoring metrics to precisely detect and classify anomalous behaviors. The authors evaluate the proposed method against three popular machine learning algorithms, decision trees, nearest neighbor, and SVM, as well as against four different monitoring datasets. Their results show that the recommended method has ability to classify overlapped anomalies and outperforms other methods by obtaining 98%-99% F-scores, and offering much higher performance than alternative techniques to detect both the period in which anomalies occurred and their type.

Lu et al. [2018] utilize convolutional neural networks (CNN) for performance anomaly diagnosis for Big Data system logs, and specifically for the Hadoop Distributed File System (HDFS) logs. They implement the proposed model with different filters to automatically train model on the relationships

among events. The CNN is configured to have *logkey2vec* embeddings, three 1D convolutional layers, dropout layer, fully connected softmax layer, and max-pooling. The authors provide a comparison between CNNs and other well-known networks such as Long Short Term Memory networks (LSTM) and Multilayer Perceptron (MLP). The experimental results show that the CNN model is more accurate and faster in detecting anomalies than LSTM and MLP for HDFS logs.

Fulp et al. [2008] predicting system failures using Support Vector Machines algorithm (SVM) for binary classification based on system log files. The proposed approach utilizes advantages of the sequential nature of logs and uses use a sliding window of messages to predict the likelihood of system failure within that has 1024 computing nodes. The SVM associates the messages to a class of normal or abnormal event. Their results show that the proposed solution can predict hard disk failure with 73% accuracy. Fu et al. [2012] propose a hybrid anomaly identification Framework using one-class and two-class SVM algorithms. They claim that their approach does require prior knowledge about system failure history and offers self adapt learning from observing system failure within cloud environment.

There are several other studies in the literature that deal with stragglers. Yadwadkar and Choi [2012] introduce a proactive straggler avoidance regression decision tree model that periodically learns correlations between node level status and task execution time for MapReduce logs. The authors justify the choice of regression trees by showing the fast prediction of stragglers. They apply their method on a trace from Facebook Hadoop system and Berkeley EECS department's local Hadoop cluster (icluster). Qi et al. [2017] use a white-box model that utilizes classification and regression trees for root causes

analyses for Spark logs and hardware sampling tools to train their model. A special type of tree called a CART tree (*classification and regression tree*) is used to mitigate overfitting issues. They use a customized prune method for several iterations to improve analysis accuracy and the classification performance metrics are checked for each node and its leaves. The authors applied their method on Spark with HiBench benchmark.

Based on the local neighborhoods of event, the neighbor-based technique uses unsupervised learning to analyze data instances. This technique can distinguish an anomalous instance among normal instances because normal instances usually occur in dense neighborhoods, whereas anomalous instances occur far from their closest neighbors (Chandola et al. [2009]). Huang et al. [2013] propose a special type of neighbor-based technique, called local outlier factor (LOF), for an anomaly identification that can learn system behaviors during training and detecting time within cloud computing environment. They argue that their method is adaptive to changes, detects contextual anomalies, and requires less effort for collecting performance metrics for training process.

1.3.2.3. Example: ANNs-based anomaly detection

Classification techniques aim to determine whether the instances in a given feature space belong to a particular class or multiple classes Ibidunmoye et al. [2015]. ANNs algorithms are the most popular classification technique for anomaly identification. This is because ANNs represent a data-driven, non-linear and self-adaptive method that can adjust itself to the given datasets without requiring prior knowledge about the distribution or function of the used model, and generalize the models even to input data that has never been seen before Zhang [2000]. These advantages have caused ANNs to be considered

a universal functional approximation.

One of the well-known critical issues for ANNs and other classifiers is feature selection. The objective of feature selection is to discover the smallest set of appropriate input features and at the same time achieve the desirable predictive performance. It is crucial to reduce the number of input features for the classifier to achieve satisfactory accuracy with reduced computation in the model Zhang [2000].

We use Backpropagation and conjugate gradients to train ANNs, that is to update values of weights and biases in the network. We use scaled conjugate because it is often fast Møller [1993], especially for time-dependent applications. *Sigmoid* transfer function is often used as an activation function in the hidden layer because it exists between (0 to 1), where zero means absence of the feature and one means its presence. In addtion, we use *Softmax* transfer function in the output layer to handle classification problems with multiple classes (e.g., normal, CPU anomaly, cache thrashing anomaly, context switching anomaly). For a cost function, we use *cross-entropy* to evaluate the performance and compare the actual output error results with the desired output values (labeled data). We use *Cross-entropy* because it has significant practical advantages over squared-error cost functions Kline and Berardi [2005].

The input layer contains a number of neurons equal to the number of input features. The size of the hidden layer is determined by using a "trial and error" method, by trying all the possible numbers between the sizes of input neurons and output neurons Sheela and Deepa [2013]. The output layer contains a number of neurons equal to the number of target classes (normal + types of anomalies).

Figure1.3 shows a sensitivity analysis for the size of collected datasets to

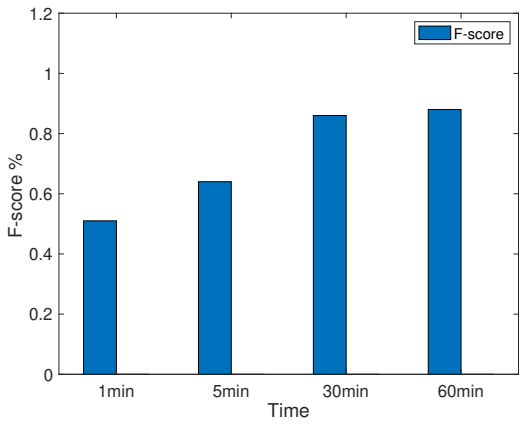

Figure 1.3: The performance of ANNs models that is trained with training dataset that have been collected for 1, 5, 30, and 60 min.

train the neural network algorithms to learn complex nonlinear relationships among performance metrics and detect the anomalous performance within Spark systems. It is clear that the size of collected training data significantly impact the ANN model, while it is challenging to find the optimal size of the training data. The small size of training dataset causes unacceptable F-score, whereas a large dataset may lead to a waste of computing resources.

1.3.3. Load prediction

Data streaming applications usually deal with unbounded data flows, meaning that they are kept in execution indefinitely. As a consequence, these applications likely face different working conditions over time (e.g., varying workloads), hence requiring dynamic resource management solutions to keep acceptable performance. Indeed, researchers have spent a lot of effort aiming to enhance streaming systems with online adaptation capabilities and, in particu-

lar, *elasticity*, that is the ability to dynamically acquire and release computing resources Hummer et al. [2013] as needed. To this end, there are two main research directions so far, revolving around (i) application load prediction and (ii) auto-scaling policies. In this section, we focus initially on load prediction.

1.3.3.1. Traditional approaches

Given the variability that often characterizes streaming workloads, predicting the application processing load in the future (e.g., application input data rate) is a difficult yet important task for driving resource management with foresight. To this end, traditional time series forecasting methods can be useful in the context of streaming applications. For instance, Imai et al. [2018] use the well-known ARIMA model for workload forecasting. They additionally consider an online regression approach for predicting the maximum sustainable throughput of streaming applications, and scale the number of virtual machines (VMs) allocated to the system accordingly. Kombi et al. [2019] instead exploit regression techniques to predict the input rate of Storm operators, and drive the application auto-scaling. They consider three prediction models, respectively based on linear, logarithmic, and exponential regression, and select the best model to use at runtime based on fitting accuracy observed in the previous iteration.

1.3.3.2. AI approaches

AI techniques often allow to outperform traditional forecasting approaches for workload and resource utilization prediction. For instance, Zacheilas et al. [2015] use GPs for predicting the future input rate and processing latency of operators, and hence drive horizontal elasticity of *complex event processing*

applications running on top of Storm. Their elasticity algorithm exploits the uncertainty estimation provided by GPs to avoid making auto-scaling decisions whenever the uncertainty level is considered too high. Hu et al. [2019] instead use SVR for predicting resource usage of Spark Streaming applications, and allocate virtual machines for the cluster so as to meet SLA requirements. Runsewe and Samaan [2017] also target Spark Streaming and leverage Layered Hidden Markov Models to predict the resource usage of multiple applications running on a Spark cluster. Based on the obtained predictions, they scale the Spark cluster as needed.

A few works have investigated the use of NNs to predict the future load of data streaming applications. Lombardi et al. [2018] propose ELYSIUM, a multi-level elasticity solution for Storm, which controls both the operator parallelism and the number of worker nodes in the Storm cluster. ELYSIUM relies on NNs for predicting both (i) the application input rate in the near future, and (ii) the CPU utilization of each application operator based on the input rate.

Mu et al. use DNNs for multi-step operator performance prediction. They define two prediction strategies based on DNNs, to be used, respectively, on offline and online collected metrics. They use ensemble learning techniques to merge the offline and online predictions, and obtain the final prediction, which can be used to drive auto-scaling. Xu et al. [2019] rely on DNNs as well, to predict operator performance online. Specifically, they exploit Recurrent DNNs to make accurate performance predictions, which also account for interference due to co-located operators (i.e., operators deployed in the same worker node). They integrate this solution in Storm, and their experiments show that it outperforms ARIMA- and SVR-based approaches for prediction. Khoshkbarforoushha et al. [2017] instead exploit Mixture Density Networks to estimate

resource usage of streaming applications as probability density functions. They show how the resulting distribution based workload prediction can be applied to drive both auto-scaling and application admission control in presence of SLAs.

1.3.4. Scaling techniques

Data-intensive systems largely exploit parallelism to efficiently process high-volume datasets. For streaming applications, whose datasets are collected in real-time and hence are not known at deployment time, scaling the amount of computing resources at runtime is fundamental to avoid the risk of under- or over-provisioning resources.

1.3.4.1. Traditional approaches

As extensively surveyed in Röger and Mayer [2019], researchers so far have investigated a large number of approaches to devise auto-scaling policies for streaming systems, including, e.g., queueing theory, control theory, state-space based methods, and, recently, AI. Existing solutions can be classified as either reactive and proactive. *Reactive* solutions make auto-scaling decisions in response to observed changes (e.g., increase in the application input data rate), whilst *proactive* approaches try to adapt the application deployment before observing changes, based on predictions.

Among the reactive approaches, threshold-based policies are widely adopted. According to these policies, auto-scaling actions are triggered whenever one or more observed metrics (e.g., resource utilization, throughput) violate predefined threshold values Castro Fernandez et al. [2013], Gedik et al. [2014].

Other works instead rely on models to periodically evaluate the expected application performance or resource utilization, and trigger scaling actions accordingly. For instance, Lohrmann et al. [2015] rely on queueing theory to model application performance, and make horizontal scaling decisions so as to meet response time requirements.

Among the proactive auto-scaling solutions, several works present policies that exploit load prediction to make scaling decisions. Indeed, most the prediction solutions mentioned in the previous section, are complemented with auto-scaling mechanisms. A different approach is considered in De Matteis and Mencagli [2016], where *model predictive control* is used to proactively scale streaming operators, also combining horizontal and vertical elasticity.

1.3.4.2. AI approaches

The behavior of traditional auto-scaling solutions often depends on manually configured parameters, and AI-based approaches aim at overcoming this limitation. For instance, Reinforcement Learning (RL) is a class of methods allowing agents (e.g., resource managers) to learn policies by direct interaction with their environment (e.g., managed applications). RL has been adopted by several works to derive auto-scaling policies for streaming applications at runtime. For instance, Heinze et al. [2014] use the SARSA algorithm, relying on a reward function that captures the difference between current operator CPU utilization and target utilization values. Similarly, Cheng et al. [2018] rely on the well-known Q-learning algorithm to adapt the amount of resources allocated to jobs running in Spark Streaming. They consider a performance-oriented reward function that accounts for throughput and latency. Lombardi et al. [2018] also use Q-learning to automatically tune the parameters for a

threshold-based auto-scaling algorithm.

Russo Russo et al. [2019] consider the auto-scaling problem in presence of heterogeneous computing resources to host parallel operator instances. They use linear function approximation to deal with the large model state space, and investigate model-based initialization to speedup the learning process. Their reward function accounts for the amount of allocated resources, the adaptation cost, and a SLO violation penalty.

1.3.5. Example: RL-based auto-scaling policies

RL agents learn by experience the *actions* to perform in order to maximize a cumulative *reward* over time (Sutton and Barto [2018]). We define the task faced by RL agents as an infinite-horizon, discrete-time Markov Decision Process (MDP), where agents perform an action at every time step, selected according to their *policy* and the observed current *state*. Following action execution, agents get a reward and possibly enter a new state. Their goal is maximizing the (discounted) cumulative reward over the infinite time horizon.

The auto-scaling problem for a streaming operator could be modeled as follows. Considering a slotted time model, we define the state at time step i, s_i, as the pair (k_i, λ_i), where $1 \leq k_i \leq K_{max}$ denotes the operator parallelism, and λ_i the monitored input rate (discretized using a suitable quantum). Actions in this model represent scaling operations that alter the parallelism level, hence they are selected from the set $\mathcal{A} = \{-1, 0, +1\}$, except for the states where no further scale-out (or, scale-in) is allowed.

In the context of resource management, it is often convenient to reason in terms of *cost* instead of reward. Therefore, we define the cost $c(s, a, s')$

paid for operating the system in state s' after taking action a in s, and let the reward $r(s, a, s') = -c(s, a, s')$. Depending on the specific scenario, the cost function may capture different aspects. We define it as a weighted sum of three cost components: resources cost (proportional to the operator parallelism level), adaptation cost (capturing the overhead due to scaling), and performance violation cost (paid whenever the chosen configuration does not satisfy performance requirements).

Most RL algorithms rely on the so-called *Q-function* $Q(s, a)$, which estimates the cumulative discounted reward obtained in the long-term when choosing action a in s. The most popular RL algorithm is Q-learning, which performs a single Q update at every time step:

$$Q^{new}(s_i, a_i) \longleftarrow (1 - \alpha)Q^{old}(s_i, a_i) + \alpha \left(r_i + \gamma \max_{a'} Q^{old}(s_{i+1}, a') \right) \quad (1.1)$$

where r_i is the reward obtained at time i, and $\alpha \in (0, 1)$ is the learning rate. Q-learning is easy to implement, and guarantees convergence to the optimal policy as infinite exploration is provided. However, to achieve faster convergence, in practice it is worth including any available knowledge about the model in the learning algorithm (*model-based* RL). For instance, the concept of *post-decision state* (PDS) (Mastronarde and van der Schaar [2011]) can be used to separate the known system dynamics (e.g., impact of a scaling action on the parallelism) from the unknown ones (e.g., input rate variations), and let the agent only learn the latter. To demonstrate the benefits provided by PDS, we simulated the execution of a data streaming operator under varying input data rate, using the RL-based auto-scaling policy. Figure 1.4 shows a sample of the workload we used, and the average reward accumulated over time by the RL agent, in the case of plain Q-learning and Q-learning with PDS. We can note

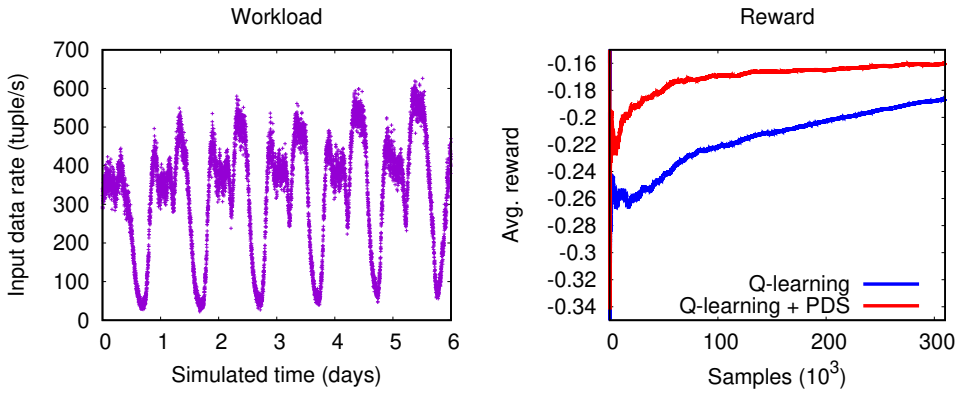

Figure 1.4: Streaming workload in a simulated experiment (left), and average reward obtained by RL-based auto-scaling agents (right).

that the PDS-based agent clearly outperforms plain Q-learning, as it exploits the available knowledge about the system and needs to learn fewer parameters.

This model can be extended to account for other adaptation mechanisms (e.g., operator migration), infrastructure characteristics, or to include more general workload characterizations. As more complex models are used, however, the state space often grows significantly, requiring, e.g., function approximation techniques for manipulating the Q-function. To this end, Deep RL is receiving growing interest, where DNNs are used to approximate Q.

1.4. Summary and conclusion

Table 1.2 summarizes the mapping of AI models to management and resource allocation activities we have referenced through this chapter. Our analysis reveals that AI-driven optimal configuration and anomaly detection in data-intensive applications have received considerable attention already, covered a

Table 1.2: Summary of the AI techniques considered for performance management of data-intensive systems: ✓: in this chapter, ∗: in extended bibliography.

AI method	Opt. Config.	Anom. Det.	Load Pred.	Scaling
Neural networks	✓	✓	✓	
Boosted regr. trees	✓			
CART	✓	✓		
Decision trees	✓	✓		
Gaussian processes	✓		✓	
LSTM	✓	✓		
Nearest neighbor	✓	✓		
RL	✓			✓
SVMs	✓	✓	✓	

broad range of methods. In spite of this, certain techniques, such as reinforcement learning, still present open room for further investigation.

As mentioned, performance management for data streaming systems benefits from runtime load prediction, which is often coupled with auto-scaling or admission control mechanisms. Neural networks, and in particular DNNs, have received the largest share of attention so far, as they have been shown to outperform other techniques, like SVMs and GPs. The adoption of other approaches, already used for forecasting in other domains, including, e.g., long-short term memory (LSTM) models and Regression Trees, could be the subject of future investigation.

As regards the definition of auto-scaling control policies for data-intensive applications, only RL techniques have been exploited so far. Nevertheless, as the class of RL methods is quite large and complex, this research direction is far from being completely explored. Among the main issues to be tackled when adopting RL algorithms, state space explosion is critical in the context of performance management, as it limits the granularity and completeness of

the application and performance models used by the agents. To overcome this issue, *Deep RL* (DRL) algorithms exploit DNNs for approximate learning in otherwise intractable tasks. For instance, Li et al. [2018] have recently applied DRL for controlling the placement of streaming operators over a cluster of machines. Further investigations are needed in the literature to understand the potential of Deep RL in the management of data-intensive systems.

In conclusion, the chapter has shown that AI methods are already subject to intense research work across various management areas, with the areas of optimal configuration and anomaly detection being the most mature. The richness and the rapid evolution of the AI research landscape offer considerable opportunities to further raise the maturity of AI methods. Our analysis reveals that, although significant work already exists in this area, load prediction and scaling techniques would particularly benefit from a broader research investigation.

Bibliography

A. S. Alnafessah and G. Casale. A neural-network driven methodology for anomaly detection in Apache Spark. In *2018 11th QUATIC*. IEEE, 2018.

A. S. Alnafessah and G. Casale. TRACK: Optimizing artificial neural networks for anomaly detection in Spark Streaming systems. In *13th VALUETOOLS*, 2020.

J. L. Berral, N. Poggi, D. Carrera, A. Call, R. Reinauer, and D. Green. ALOJA-ML: A framework for automating characterization and knowledge discovery in Hadoop deployments. In *21th ACM SIGKDD*, 2015.

M. Bilal and M. Canini. Towards automatic parameter tuning of stream processing systems. In *2017 SoCC*. ACM, 2017.

E. Brochu, V. M. Cora, and N. De Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache flink^{TM}: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015.

R. Castro Fernandez, M. Migliavacca, E. Kalyvianaki, and P. Pietzuch. Integrating scale out and fault tolerance in stream processing using operator state management. In *2013 ACM SIGMOD*. ACM, 2013.

V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15, 2009.

C.-O. Chen, Y.-Q. Zhuo, C.-C. Yeh, C.-M. Lin, and S.-W. Liao. Machine learning-based configuration parameter tuning on Hadoop system. In *2015 IEEE Int'l Congress on Big Data*. IEEE, 2015.

D. Cheng, X. Zhou, Y. Wang, and C. Jiang. Adaptive scheduling parallel jobs with dynamic batching in Spark Streaming. *IEEE TPDS*, 29(12):2672–2685, 2018.

T. De Matteis and G. Mencagli. Keep calm and react with foresight: Strategies for low-latency and energy-efficient elastic data stream processing. *SIGPLAN Not.*, 51(8), 2016.

D. J. Dean, H. Nguyen, and X. Gu. UBL: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems. In *Proc. of ICAC '12*. ACM, 2012.

P. Di Sanzo, D. Rughetti, B. Ciciani, and F. Quaglia. Auto-tuning of cloud-based in-memory transactional data grids via machine learning. In *2012 2nd NCCA*. IEEE, 2012.

H. Fang, W. Lu, Q. Li, J. Kong, L. Liang, B. Kong, and Z. Zhu. Predictive analytics based knowledge-defined orchestration in a hybrid optical/electrical datacenter network testbed. *J. of Lightwave Technology*, 37(19):4921–4934, 2019.

S. Fu. Performance metric selection for autonomic anomaly detection on cloud computing systems. In *2011 IEEE GLOBECOM*. IEEE, 2011.

S. Fu, J. Liu, and H. Pannu. A hybrid anomaly detection framework in cloud computing using one-class and two-class support vector machines. In *ADMA 2012: Advanced Data Mining and Applications*. Springer, 2012.

E. W. Fulp, G. A. Fink, and J. N. Haack. Predicting computer system failures using support vector machines. In *WASL '08*. USENIX Association, 2008.

B. Gedik, S. Schneider, M. Hirzel, and K. Wu. Elastic scaling for data stream processing. *IEEE TPDS*, 25(6):1447–1463, 2014.

A. Gunawan and Lindawati Lau, H. C. Fine-tuning algorithm parameters using the design of experiments approach. In *LION 2011: Learning and Intelligent Optimization*. Springer, 2011.

T. Heinze, V. Pappalardo, Z. Jerzak, and C. Fetzer. Auto-scaling techniques for elastic data stream processing. In *2014 ICDEW*, 2014.

Á. B. Hernández, M. S. Perez, S. Gupta, and V. Muntés-Mulero. Using machine learning to optimize parallelism in big data applications. *Future Generation Computer Systems*, 86: 1076–1092, 2018.

H. Herodotou, Y. Chen, and J. Lu. A survey on automatic parameter tuning for big data processing systems. *ACM Comput. Surv.*, 53(2):1–37, 2020.

V. J. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.

Z. Hu, H. Kang, and M. Zheng. Stream data load prediction for resource scaling using online support vector regression. *Algorithms*, 12(2):37, 2019.

T. Huang, Y. Zhu, Q. Zhang, Y. Zhu, D. Wang, M. Qiu, and L. Liu. An LOF-based adaptive anomaly detection scheme for cloud computing. In *2013 IEEE COMPSACW*, pages 206–211. IEEE, 2013.

W. Hummer, B. Satzger, and S. Dustdar. Elastic stream processing in the cloud. *WIREs Data Mining and Knowledge Discovery*, 3(5):333–345, 2013.

O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth. Performance anomaly detection and bottleneck identification. *ACM Comput. Surv.*, 48(1):1–35, 2015.

S. Imai, S. Patterson, and C. A. Varela. Uncertainty-aware elastic virtual machine scheduling for stream processing systems. In *2018 18th IEEE/ACM CCGRID*, pages 62–71, 2018.

P. Jamshidi and G. Casale. An uncertainty-aware approach to optimal configuration of stream processing systems. In *2016 24th IEEE MASCOTS*. IEEE, 2016.

T. T. Joy, S. Rana, S. Gupta, and S. Venkatesh. Hyperparameter tuning for big data using Bayesian optimisation. In *2016 23rd ICPR*. IEEE, 2016.

H. Karau, A. Konwinski, P. Wendell, and M. Zaharia. *Learning Spark: Lightning-fast Big Data Analysis*. O'Reilly Media, Inc., 2015. ISBN 144935906X.

Terence Kelly. Detecting performance anomalies in global applications. In *WORLDS*, volume 5, pages 42–47, 2005.

A. Khoshkbarforoushha, R. Ranjan, R. Gaire, E. Abbasnejad, L. Wang, and A. Y. Zomaya. Distribution based workload modelling of continuous queries in clouds. *IEEE Transactions on Emerging Topics in Computing*, 5(1):120–133, 2017.

D. M. Kline and V. L. Berardi. Revisiting squared-error and cross-entropy functions for training NN classifiers. *Neural Computing Applications*, 14(4):310–318, 2005.

R. K. Kombi, N. Lumineau, P. Lamarre, N. Rivetti, and Y. Busnel. DABS-Storm: A data-aware approach for elastic stream processing. In *Transactions on Large-Scale Data and Knowledge-Centered Systems XL*, pages 58–93. Springer, 2019.

M. Li, J. Tan, Y. Wang, L. Zhang, and V. Salapura. In *2015 12th ACM CF*, number 53, pages 1–8. ACM.

T. Li, Z. Xu, J. Tang, and Y. Wang. Model-free control for distributed stream data processing using deep reinforcement learning. *Proceedings of the VLDB Endowment*, 11(6):705–718, 2018.

G. Liao, K. Datta, and T. L. Willke. Gunther: Search-based auto-tuning of mapreduce. In *Euro-Par 2013*. Springer, 2013.

B. Lohrmann, P. Janacik, and O. Kao. Elastic stream processing with latency guarantees. In *2015 35th IEEE ICDCS*, 2015.

F. Lombardi, L. Aniello, S. Bonomi, and L. Querzoni. Elastic symbiotic scaling of operators and resources in stream processing systems. *IEEE TPDS*, 29(3):572–585, 2018.

S. Lu, B. Rao, X. Wei, B. Tak, L. Wang, and L. Wang. Log-based abnormal task detection and root cause analysis for Spark. In *2017 IEEE ICWS*. IEEE, 2017.

S. Lu, X. Wei, Y. Li, and L. Wang. Detecting anomaly in big data system logs using convolutional neural network. In *2018 IEEE 16th DASC/PiCom/DataCom/CyberSciTech*. IEEE, 2018.

N. Mastronarde and M. van der Schaar. Fast reinforcement learning for energy-efficient wireless communication. *IEEE Trans Signal Process.*, 59(12):6262–6266, 2011.

X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, and S. Owen. MLlib: Machine learning in Apache Spark. *JMLR*, 17(1): 1235–1241, 2016.

M. F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525–533, 1993.

W. Mu, Z. Jin, F. Liu, W. Zhu, and W. Wang. OMOPredictor: An online multi-step operator performance prediction framework in distributed streaming processing. In *2019 IEEE ISPA/BDCloud/SocialCom/SustainCom*.

N. Nguyen, M. M. H. Khan, and K. Wang. Towards automatic tuning of Apache Spark configuration. In *2018 IEEE 11th CLOUD*. IEEE, 2018.

D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do internet services fail, and what can be done about it? In *2003 4th USITS*. USENIX Association, 2003.

C. Peng, C. Zhang, C. Peng, and J. Man. A reinforcement learning approach to map reduce auto-configuration under networked environment. *Int'l J. of Security and Networks*, 12 (3):135–140, 2017.

S. Pertet and P. Narasimhan. Causes of failure in web applications. Technical Report CMU-PDL-05-109, Carnegie Mellon University, 2005.

W. Qi, Y. Li, H. Zhou, W. Li, and H. Yang. Data mining based root-cause analysis of performance bottleneck for big data workload. In *2017 IEEE HPCC/SmartCity/DSS*. IEEE, 2017.

H. Röger and R. Mayer. A comprehensive survey on parallelization and elasticity in stream processing. *ACM Comput. Surv.*, 52(2), 2019.

S. Rogers and M Girolami. *A First Course in Machine Learning*. Chapman and Hall/CRC, 2nd edition, 2016.

O. Runsewe and N. Samaan. Cloud resource scaling for big data streaming applications using a Layered Multi-dimensional Hidden Markov Model. In *2017 17th IEEE/ACM CCGRID*, 2017.

G. Russo Russo, V. Cardellini, and F. Lo Presti. Reinforcement learning based policies for elastic stream processing on heterogeneous resources. In *13th ACM DEBS*, 2019.

G. Sebestyen, A. Hangan, Z. Czako, and G. Kovacs. A taxonomy and platform for anomaly detection. In *2018 IEEE AQTR*. IEEE, 2018.

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

S. Shahrivari. Beyond batch processing: Towards real-time and streaming big data. *IEEE Computers*, (4):117–129, 2014.

K. G. Sheela and S. N. Deepa. Review on methods to fix number of hidden neurons in neural networks. *Mathematical Problems in Engineering*, 2013.

J. Shi, J. Zou, J. Lu, Z. Cao, S. Li, and C. Wang. MRTuner: A toolkit to enable holistic optimization for MapReduce jobs. *Proceedings of the VLDB Endowment*, 7(13):1319–1330, 2014.

J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *2012 NISP*, pages 2951—2959, 2012.

R. S. Sutton and A. G. Barto. *Reinforcement Learning - An Introduction*. MIT Press, 2nd edition, 2018.

Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, and D. Rajan. Prepare: Predictive performance anomaly prevention for virtualized cloud systems. In *2012 IEEE 32nd ICDCS*, 2012.

A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, and J. Donham. Storm @Twitter. In *2014 ACM SIGMOD*. ACM, 2014.

G. Wang, J. Xu, and B. He. A novel method for tuning configuration parameters of Spark based on machine learning. In *2016 IEEE 18th HPCC/SmartCity/DSS*. IEEE, 2016.

J. Xu, J. Tang, Z. Xu, C. Yin, K. Kwiat, and C. Kamhoua. A deep recurrent neural network based predictive control framework for reliable distributed stream data processing. In *2019 IEEE IPDPS*, 2019.

N. J Yadwadkar and W. Choi. Proactive straggler avoidance using machine learning. *White paper, University of Berkeley*, 2012.

L. Yang, C. Liu, J. M. Schopf, and I. Foster. Anomaly detection and diagnosis in grid environments. In *2007 ACM/IEEE Supercomputing*, 2007.

N. Yigitbasi, T. L Willke, G. Liao, and D. Epema. Towards machine learning-based auto-tuning of MapReduce. In *2013 IEEE 21st MASCOTS*. IEEE, 2013.

N. Zacheilas, V. Kalogeraki, N. Zygouras, N. Panagiotou, and D. Gunopulos. Elastic complex event processing exploiting prediction. In *2015 IEEE Big Data*, 2015.

G. P. Zhang. Neural networks for classification: A survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 30(4):451–462, 2000.