**Project Title: Advanced methods for massive real-time data stream processing**

As we enter the era of Big Data, a large number of data-driven systems and applications have become an integral part of our daily lives. It is estimated that 1.7MB of data is created every second for every person on Earth, for a total of over 2.5 quintillion bytes of new data every day, reaching 163 zettabytes by 2025. In addition to the growing volume, velocity, and variety of continuously generated data, novel technological trends such as edge processing, IoT, 5G, and federated AI bring new requirements for faster processing and deeper, more computationally heavy data analysis. Meeting these requirements in modern applications by relying on the “old school” data processing mechanisms is nearly impossible, and a new solution is needed.

Stream processing comprises a variety of methods for scalable and efficient data processing that do not rely on traditional databases for storing and processing the data. Instead, the main focus of these methods is on performing highly complex computations on high-rate data streams while only using minimal resources. This makes stream processing a perfect choice for implementing intensive data processing operations in real-time applications and on edge devices.

In this project, you will contribute to the development of [OpenCEP](https://eur01.safelinks.protection.outlook.com/?url=https%3A%2F%2Fgithub.com%2Filya-kolchinsky%2FOpenCEP&data=04%7C01%7Cassaf%40technion.ac.il%7C26371d19555d4ee23c4d08d94c466b3e%7Cf1502c4cee2e411c9715c855f6753b84%7C1%7C0%7C637624687593437354%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=X8kY6aeooanlZGjmW2DYdAasxe7E6Zns9Y9DLmatzRg%3D&reserved=0" \t "_blank) - an open source next-generation data stream processing engine. The engine utilizes state-of-the-art optimization methods and algorithms stemming from the latest research advancements to provide a high-performance framework for real-time monitoring and processing of Big Data streams. This functionality is crucial for applications like credit card fraud detection, network security monitoring, stock market analysis, and IoT. Designed with many streaming operations in mind, its main focus is on answering advanced streaming queries and on-the-fly detection of complex patterns in streaming data, also known as complex event processing (CEP).

**A Reinforcement Learning-based Scheduler for Kubernetes**

[Kubernetes](https://kubernetes.io/) is a leading open source framework for *container orchestration* - the process of automating deployment and managing applications in a containerized and clustered environment.

**At its basic level, Kubernetes** runs and coordinates container-based applications across a cluster of machines. It is designed to completely manage the life cycle of containerized applications and services using methods that provide predictability, scalability, and high availability.

A Kubernetes user provides the applications to be deployed in the form of *pods* - basic units of execution containing one or more tightly-coupled containers. These pods are then assigned to *nodes*, that represent actual computing resources such as physical or virtual servers. A collection of nodes forms a *Kubernetes cluster*, managed by the *control plane*. All nodes are heterogeneous by definition, that is, they may encapsulate resources of different capacities. Each node can contain and concurrently execute multiple pods. The following figure provides a simplified view of the architecture of Kubernetes.



Scheduling pods to nodes - that is, assigning each pod to a node on which it will be deployed - is a highly crucial, yet extremely difficult task. A suboptimal pod placement strategy could result in severe underutilization of cluster resources and unsatisfactory application performance. However, finding an optimal placement is a well-known NP-hard combinatorial problem (similar to bin packing) that cannot be solved under practical time and resource constraints. Therefore, the currently available implementation of Kubernetes *scheduler*, a component responsible for this task, resorts to simple heuristics that often provide results of poor quality.

The goal of this project is to design and implement an **alternative scheduler for Kubernetes**, utilizing recent advances in *deep reinforcement learning* to combine low running time with high quality of the produced placements. This *RL-based scheduler* will repeatedly attempt different pod placements in the cluster and learn the most potent scheduling decisions based on the resulting application performance and resource utilization. The scheduler will be fully compatible with any application currently relying on the default Kubernetes scheduler in terms of API and the supported parameters and constraints.

**Ceph Drive Failure Prediction**

More than [2500 petabytes](https://www.domo.com/learn/data-never-sleeps-5?aid=ogsm072517_1&sf100871281=1) of data is generated every day by sources such as social media, IoT, commercial services, etc. Of this, a sizeable chunk is persisted in storage systems (HDDs and SSDs). To ensure that data is not lost or corrupted, large scale storage solutions often used erasure-coding or mirroring. However, these techniques become more difficult and/or expensive to deal with at scale.

This project aims to enhance [Ceph](https://ceph.io), a high-performance distributed storage system (see architecture illustration below), by giving it the capability to **predict the failure of storage devices** well in advance. These predictions can then be used to determine when to add/remove replicas. In this way, the fault tolerance may be improved by up to an order of magnitude, since the probability of data loss is generally related to the probability of multiple, concurrent device failures.



The Backblaze Hard Drive dataset will be used for this project. This dataset consists of daily snapshots of basic information, SMART metrics, and status (failure label) for the hard drives in the Backblaze data center. Details about this dataset can be found [here](https://www.backblaze.com/b2/hard-drive-test-data.html). To learn more about the SMART system and SMART metrics, see [this](https://en.wikipedia.org/wiki/S.M.A.R.T.) Wikipedia article.

The goal is to create predictive models using the Backblaze dataset to determine when a hard drive will fail. Ideally, the model should be able to predict the health of a hard drive in terms of "good" (>6 weeks till failure), "warning" (2-6 weeks till failure), and "bad" (<2 weeks till failure), as well as provide a probability that a hard drive will fail within some pre-defined time period. The solutions will incorporate a variety of well-known *time series forecasting (TSF)* algorithms, based on statistical methods, deep neural networks, or their combination.

At inference time, 6 days of SMART data (6 rows from the Backblaze dataset) will be available to feed to this multiclass classification model. How the model makes use of this is a design choice. It may predict on all 6 individually, or generate features using multiple days data, or use only the last day data, etc. For details on how this model would be integrated into Ceph (API, preprocessing at inference time, etc) see [this](https://github.com/ceph/ceph/tree/master/src/pybind/mgr/diskprediction_local).

**Learning Resource Consumption of Kubernetes applications**

[Kubernetes](https://kubernetes.io/) is a leading open source framework for *container orchestration* - the process of automating deployment and managing applications in a containerized and clustered environment.

**At its basic level, Kubernetes** runs and coordinates container-based applications across a cluster of machines. It is designed to completely manage the life cycle of containerized applications and services using methods that provide predictability, scalability, and high availability.

A Kubernetes user provides the applications to be deployed in the form of *pods* - basic units of execution containing one or more tightly-coupled containers. These pods are then assigned to *nodes*, that represent actual computing resources such as physical or virtual servers. A collection of nodes forms a *Kubernetes cluster*, managed by the *control plane*. All nodes are heterogeneous by definition, that is, they may encapsulate resources of different capacities. Each node can contain and concurrently execute multiple pods. A simplified view of the architecture of Kubernetes is presented in the accompanying figure of the Kubernetes scheduler project.

Scheduling pods to nodes - that is, assigning each pod to a node on which it will be deployed - is a highly crucial, yet extremely difficult task. A suboptimal pod placement strategy could result in severe underutilization of cluster resources and unsatisfactory application performance. Pod placement decisions in Kubernetes are handled by a dedicated component named *scheduler*.

To select a node for an unscheduled pod, the scheduler must take a variety of user-provided constraints and policy directives into account. In particular, knowing the expected *resource (CPU and memory) consumption* of a pod is essential for producing good placements. However, many applications fail to provide good estimates for their expected resource usage. As a result, the scheduling decisions are based on biased and inaccurate information, ultimately leading to poor performance.

The goal of this project is to overcome the above issue by devising and implementing a **smart application profiler** - a ML-based component that observes the behavior of an application in a cluster over time and learns to predict the expected resource consumption of a new pod awaiting to be scheduled. Armed with this learned/predicted information, the Kubernetes scheduler will be able to dramatically improve the quality of its decisions.



More formally, the proposed application profiler will receive as input (or collect over time) a series of prior observations recording the resource consumption of the pods previously executed in the cluster. Based on this dataset, the profiler will learn the features of the pod that are most indicative of its expected CPU and memory consumption. In addition, it will consider external features, such as the time of the day or the day of the week, for making a prediction.

**Cloud Price Analysis and Prediction**

With the amount of data rapidly growing, many applications need higher-performance hardware to support their execution. However, for most individuals and organizations, acquiring this hardware is prohibitively expensive. Consequently, a highly popular solution of migrating user applications to the *cloud* and utilizing the *pay-as-you-go* pricing scheme has emerged as a suitable avenue for reducing the maintenance costs and achieving true scalability. There are three major service models for cloud computing: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).

[Amazon Elastic Compute Cloud (EC2)](https://aws.amazon.com/ec2) [[2](https://hcis-journal.springeropen.com/articles/10.1186/s13673-020-00239-5#ref-CR2)] is one of the representatives of IaaS, which can provide users with basic hardware resources, such as CPU, network, memory and storage. Amazon EC2 provides users with many purchase options, including reserved instances, on-demand instances and spot instances, that differ in their prices and guarantees. Specifically, Amazon’s spot instance is the most popular, widely used and representative in the cloud market.

Cloud providers keep changing the prices of the various instances (VMs) from time to time. These highly fluctuating changes are caused by known phenomena and follow complex (and mostly hidden) patterns. A lack of knowledge of these patterns and the information about how and when these prices change results in a great degree of uncertainty for customers. Being able to understand price changes would help customers take appropriate measures and make correct decisions to best manage their infrastructures and the associated costs.



In this project, the aim is to develop a framework for forecasting the upcoming costs of cloud instances based on historical price data (available on [Amazon EC2 dashboard](https://console.aws.amazon.com/ec2)). Such a capability would potentially allow cloud users to save a considerable amount of money as well as improve the reliability of their applications. The framework will utilize a combination of novel methods for time series forecasting based on statistical and deep learning approaches.

**Predicting the Future Prometheus Metrics of a Kubernetes Cluster**

[Kubernetes](https://kubernetes.io/) is a leading open source framework for *container orchestration* - the process of automating deployment and managing applications in a containerized and clustered environment. **At its basic level, Kubernetes** runs and coordinates container-based applications across a cluster of machines. It is designed to completely manage the life cycle of containerized applications and services using methods that provide predictability, scalability, and high availability.

A Kubernetes user provides the applications to be deployed in the form of *pods* - basic units of execution containing one or more tightly-coupled containers. These pods are then assigned to *nodes*, that represent actual computing resources such as physical or virtual servers. A collection of nodes forms a *Kubernetes cluster*, managed by the *control plane*.

One of the most important tasks of a Kubernetes cluster admin is to constantly monitor the status of the cluster and react accordingly when situations of interest occur or anomalies in the workflow are detected (such as an application failure or a malfunctioning node). The de facto standard monitoring software framework utilized to that end is called [Prometheus](https://prometheus.io/). As illustrated in the figure below, Prometheus collects real-time metrics from application services and nodes (up to millions of metrics per second), stores them in a time-series database, and provides a convenient visualization tool for the user to examine, query and analyze the data.



While Prometheus is a very powerful monitoring system capable of high-quality real-time data gathering, it entirely lacks one important capability: *predicting the expected values of future metrics*. By detecting and observing repeatedly occurring patterns and trends in the monitored data and learning from them, it might be possible to not only detect abnormal conditions and situations, but also to forecast their occurrence in advance, thus providing an admin with sufficient time to prevent the possibly harmful or otherwise undesired event.

The goal of this project is to fill this gap by implementing a predictive component for Prometheus metrics. This component will incorporate a variety of well-known *time series forecasting (TSF)* algorithms, based on statistical methods, deep neural networks, or their combination. It will receive as input a stream of Prometheus updates (i.e., files containing the last recorded values for the monitored metrics) and generate a stream of predicted future values for all involved metrics.