EMS®: A Massive Computational Experiment Management System towards Data-driven Robotics

Qinjie Lin[1] Guo Ye[1] Han Liu[1,2]

***Abstract*— We propose EMS®, a cloud-enabled massive computational experiment management system supporting high-throughput computational robotics research. Compared to existing systems, EMS® features a sky-based pipeline orchestrator which allows us to exploit heterogeneous computing environments painlessly (e.g., on-premise clusters, public clouds, edge devices) to optimally deploy large-scale computational jobs (e.g., with more than millions of computational hours) in an integrated fashion. Cornerstoned on this sky-based pipeline orchestrator, this paper introduces three abstraction layers of the EMS® software architecture: (i) Configuration management layer focusing on automatically enumerating experimental configurations; (ii) Dependency management layer focusing on managing the complex task dependencies within each experimental configuration; (iii) Computation management layer focusing on optimally executing the computational tasks using the given computing resource. Such an architectural design greatly increases the scalability and reproducibility of data-driven robotics research leading to much-improved productivity. To demonstrate this point, we compare EMS® with more traditional approaches on an offline reinforcement learning problem for training mobile robots. Our results show that EMS® outperforms more traditional approaches in two magnitudes of orders (in terms of experimental high throughput and cost) with only several lines of code change. We also exploit EMS® to develop mobile robot, robot arm, and bipedal applications, demonstrating its applicability to numerous robot applications.**

***Index Terms*— computational robotics, cloud robotics, massive computational experiments, sky computing**

I. INTRODUCTION

We are witnessing the emergence of massive computation as a fundamental strategy in modern robotics research. For example, OpenAI uses 6,144 CPU cores and 8 GPUs to train a robot hand manipulation policy [1] for about 50 hours, and Deepmind takes 340 million training steps to train AlphaGo [2], with 50 GPUs for about 500 hours. More examples include not only traditional hardware robots such as legged locomotion [3]–[11], manipulation [12]–[21], and navigation [22]–[30], but also software robots like conversational agents [31]–[35] and game AI [36]–[40]. Though these research fields are seemingly diversified, they address the same scientific question: "Which configuration is optimal for a given utility?" A principled approach that systematically answers this question is massive computational experiments [41], which complements two other important scientific methods for modern robotics research: mathematical analysis and physical experimentation. Compared to massive computational experiments, mathematical analysis can not be conducted in many realistic settings since it generally requires stylized simplification of formal models, while physical experimentation is not applicable on larger scales due to its expensiveness in time and budget. To implement the massive computation strategy, the most crucial step is to develop an experiment management system [41].

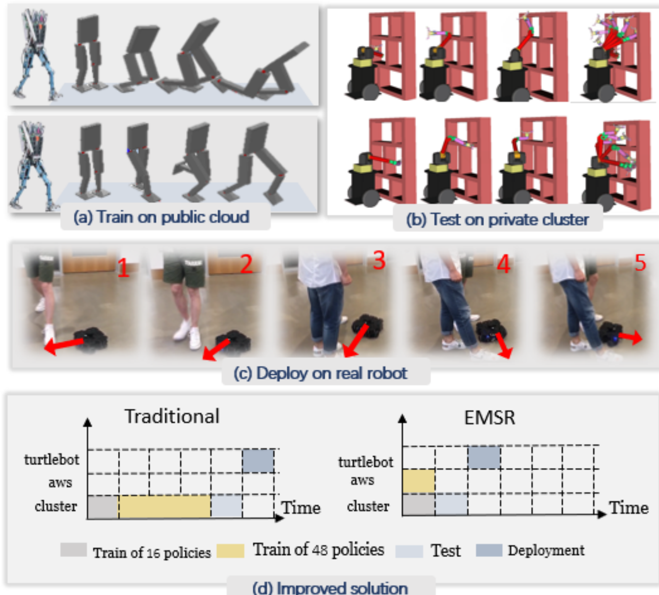

Fig. 1. Examples of EMS®on robotics experiments. (a) shows bipedal training tasks on public cloud computation, (b) illustrates motion planning tasks of a robot arm, on a private cluster, (c) demonstrates collision avoidance performance of turtlebot, using onboard computation. These show sky-features of robot pipelines. (d) shows traditional pipeline and improved pipeline in EMS®. It consists of training 64 policies, testing the trained policies, and deploying the best policy on turtlebot. Specifically, the cluster can afford parallel training for a maximum of 16 policies within 20 minutes and the other 48 are scheduled on aws or the same cluster. The testing tasks are both scheduled on the cluster and deployments are on turtlebot's onboard computation in 20 minutes.

Though some general-purpose experiment management systems have been developed [41]–[43], they are not directly applicable to modern robotics research due to the challenge that a sophisticated intelligent robot pipeline generally requires the usage of heterogeneous computational resources ranging from edge devices, on-premise deployment, public cloud, etc. In contrast, most existing systems are cluster or cloud-specific. To illustrate the multi-cloud nature of data-

[1]Department of Computer Science, Northwestern University, Evanston, IL, USA. hanliu@northwestern.edu, qinjielin2018, guoye2018@u.northwestern.edu

[2]Han Liu's research is partly supported by NIH R01LM1372201, NSF CAREER1841569 and a NSF TRIPODS1740735.

driven robotics applications, we consider an example of developing a reinforcement learning-based retail navigation robot. In this application, data collection should be run in an edge robot network, the decision model could be deployed on some public cloud (e.g., AWS, GCP, Azure), while the very intensive data storage has better to be on some on-premise cluster due to the budget concern. In this scenario, developers need to repeatedly transfer data between different components in the pipeline. More sky-based pipelines are illustrated in Figure 1.

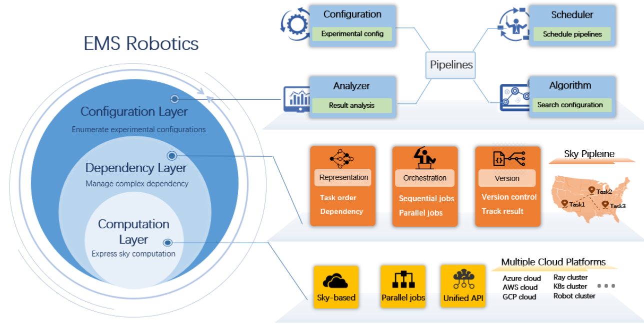

Fig. 2. An overview of EMS®. EMS® enables sky-based pipeline orchestration. It consists of three layers: (i) Computation management layer abstracting clouds computing and selecting cloud on which various jobs are supposed to run on; (ii) Dependency management layer managing complex dependency in the pipeline; (iii) Configuration management layer automatically enumerating experimental configurations.

To handle the above challenge, we propose EMS® (Experiment Management System for Robots), a cloud-enable massive computational experiment management system for robotics research (See Figure 2 for more details). Unlike most existing experiment management systems [44]–[51] which are specific to a particular cloud or cluster environment, EMS® supports a full-fledged orchestration of sky-based experiment pipelines, enabling different clouds or clusters to run various stages of a robotics research pipeline. In particular, the system provides a highly sophisticated abstraction of the underlying computation platforms such that researchers only need to change one line of code to run the experiment across different computation environments, without worrying about the data transfer and cloud service interface. In addition, EMS® automatically optimizes resource allocation according to given time and budget constraints.

The EMS® architecture consists of three abstraction layers, shown in Figure 2. (i) A sky-based computation layer providing a unified and easy-to-use API for submitting, monitoring, and canceling massive-scale computational jobs. This layer abstracts out the implementation details of all computing resources to free higher-level layers from the perceived complexity of using different cloud services. (ii) A dependency management layer using a DAG (directed acyclic graph) representation to orchestrate a task pipeline within an experiment. This layer features a sophisticated version controller which allows us to best reuse results from previous experiments. (iii) A configuration management layer providing a unified configuration programming interface to manage and optimize the schedule of massive amount of computational experiments. Such a layered design makes computational robotics research more efficient and scalable.

The rest of the paper is organized as follows. Section II introduces work related to cloud robotics and experiment management system. Section III illustrates EMS® architecture and discusses some implementation detail. Section IV demonstrates experimental results on the improved productivity, reproducibility, and scalability of EMS®.

II. Related Work

This section summarizes related work on experiment management system and cloud robotics system.

Experiment management system. An experiment management system [52] is a software stack that automates the process of experimental job management, output harvesting, data analysis, reporting, and publication of code and data. Some general-purpose experiment management systems include ClsterJob [41], codalab [42], pywren [43], Kubernetes batch [53], and AWS batch [54]. They facilitate researchers to conduct million-CPU-hour experiments in a painless and reproducible way. The most relevant works to this paper are the machine-learning pipeline management systems [55]. Such systems fall into two categories: (i) machine learning pipeline management service from a cloud provider, like AWS sagemaker [44], Microsoft Azure ML [45], and GCP MLops [46]. (ii) automatic installation packages on general bare-metal servers, including Ray Job [48], Kubeflow [47], TensorFlow Extended (TFX) [56], MLlib [49], MetaFLow [57], and ScikitLearn [58]. Though significant progress has been made, these systems are not designed for robotics research which requires heterogeneous computational resources. To bridge this gap, EMS® resorts to a sky-based pipeline orchestration framework.

Cloud robotics system. A cloud robotics system uses cloud resources to enable greater memory, computational power, and interconnectivity for robotics applications. Early works focus on facilitating the seamless integration of robot and edge devices into both on-premises clusters and public cloud services [59]–[63]. Specifically, DAvinCi [64] implements a software framework on the Hadoop cluster to scale robotic development and RoboFlow [50] is a cloud-based workflow management system orchestrating the pipelines on Kubernetes cluster. FetchCore [65] and Formant [66] robotics build web-based robot management systems, automating monitoring and controlling robotics. To leverage the power of public clouds, RoboEarth [67] and Rapyuta [68] propose a system architecture, enabling robots to delegate their intense computational tasks to the Amazon cloud service. FogRos [69], [70] automates robotics deployment on public cloud and SmartCloud [71] facilitates robot interactions with public cloud services. Honda [72] proposes a serverless architecture for service robot management systems on AWS and Robomaker [73] provides a cloud-based simulation service for scaling robotic applications. Most of these systems focus on robot deployment. In contrast, EMS® focuses on massive experiment management.

III. System Architecture and Implementation

The EMS® architecture consists of thee abstraction layers: (i) a computation management layer expressing sky-based computation, (ii) a dependency management layer providing sky-based pipeline orchestration, and (iii) a configuration management layer enumerating experimental configurations. Though EMS® is fully generic and applicable to any robotics application, we describe the system using a turtlebot pipeline, with the hope of making the system more accessible to the audience. The pipeline consists of four tasks (training collision avoidance policy using PPO, two testing, and deployment on a real robot).

EMS® uses sky computing for pipeline orchestration. In particular, it exploits an intercloud broker to optimally mediate multi-cloud computing by abstracting away the deployment details of the underlying clouds. This is different from the two related types of multi-cloud solutions. One is partitioned multi-cloud enabling different corporate teams to run their workloads on different clouds or on-premise clusters. The other is portable multi-cloud enabling the same application to run on multiple clouds. Examples include many third-party cloud applications (e.g. Confluent [74], Databricks [75], Snowflake [76], Trifacta [77]), uniform low interface across multiple clouds (e.g. Kubernetes [78], Google Anthos [79], Azure ARC [80], AWS Outposts [81]), and previous sky computing providing uniform infrastructure-as-a-service for applications [82], [83]. In contrast to these works, sky computing provides a set of uniform high-level APIs for different cloud services so that users can split the experiment pipelines across different clouds. In addition, the intercloud broker enables selecting different clouds for job execution while optimizing customized metrics (such as price or performance). To conduct sky-based pipeline orchestration, EMS® implements an intercloud broker in the computation abstraction layer as described in Section III-A.

A. Computation Management Layer

The computation layer features an intercloud broker that hides all the implementation details of the multi-cloud computation. As shown in Figure 3, the input of the intercloud broker is a set of computational jobs. It parses the job list and submits the jobs to cloud services (e.g. AWS batch) or cluster services (e.g. Kubernetes, Ray Cluster) for execution. After job submission, it monitors the job output and provides execution feedback to the parent process. We describe each job as a Python `dataclass` object whose attributes include job name, job resource demand like CPU number, GPU number, memory size, and job executed commands, target cloud name. In the turtlebot pipeline, the computation layer can schedule jobs of training collision avoidance policy on aws cloud or a private kubernetes cluster. This provides massive computation to the pipeline.

The intercloud broker consists of the following components (1) **Job APIs**: Job APIs provide a set of unified interfaces for developers to submit jobs to clouds, monitor and log running jobs on clouds, and cancel submitted jobs on clouds. Available APIs are shown in Table I. (2) **Optimizer**: Given job descriptions, the optimizer generates optimal execution plans including moving data to a different region and selecting service providers. For example, if developers specify a job on AWS batch, the optimizer determines AWS batch instance types, numbers, and region (3) **Cloud Service Catalog**: This catalog is a list of open service interfaces, provided by cloud or cluster vendors. The service in this catalog can be a third-party library like boto3 [84], or a command line like Azure CLI. (4) **Executor**: The executor creates required resources and executes commands on the resources allocated by the optimizer. (5) **Data Orchestratio**n: Data orchestration manages the data related to jobs, like code folder, raw data, and generated model data. If computation jobs are launched on the same region as the data, then it mounts the data-related storage to the instance, executing jobs. If they are in different regions, it copies the related data to the job region before launching jobs. (6) **Logger**: Loggers keep pulling jobs output from clouds. It provides real-time job execution status for the parent process.

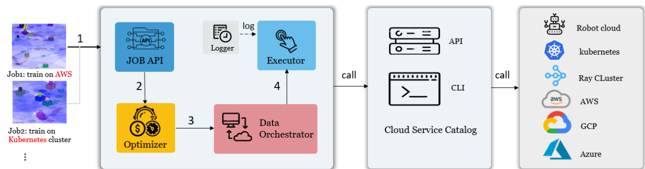

Fig. 3. The computation layer architecture. The intercloud broker consists of the middle two panels. It creates an interface for jobs and clouds. It takes job descriptions as input and executes jobs on target clouds.

TABLE I. Description of job APIs.

Job APIs	Description
`submit_job()`	Submit jobs to clouds. The resource requirement, execution command lines, conda environment, and related data storage are specified in the job.json.
`status_job()`	List jobs status on the clouds. Status can be submitted, running, canceled, failed, or succeeded.
`cancel_job()`	Cancel jobs on the clouds.
`watch_job()`	Pull job output from clouds and print out the content.

B. Dependency Management Layer

Built upon the computation management layer, the dependency management layer orchestrates task pipelines within one experiment. The architecture is illustrated in Figure 4. Specifically, this layer takes turtlebot pipeline description as input, generates a version number and job description for each task in the pipeline, and then submits the job description in both sequential and parallel order.

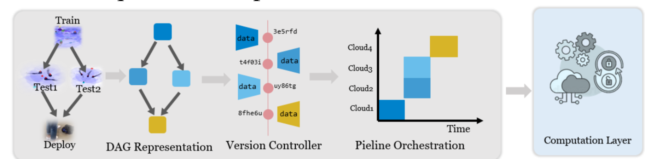

Fig. 4. The dependency management layer, managing a turtlebot pipeline. Rectangles represent tasks and trapezoids represent version seed generated from the version controller.

To perform these functions, the dependency management layer consists of four major components: (1) **DAG Representation**: We use a directed acyclic graph (DAG) to represent the execution dependency between tasks. Each task in the

pipelines describes the computation requirement, source file location, and dependency tasks. (2) **Pipeline Orchestrator**: Given a DAG representation, the pipeline orchestrator generates job descriptions for each task and determines the execution order. It also detects the last modification time of the related data files in each task. If the detected time is later than the last task execution time, it submits the job for execution, otherwise, it keeps the previous results. (3) **Version Controller**: Version Controller tracks every pipeline experiment metadata like code, data, timing information, and pipeline results. It generates a version number for each task in the pipeline, which changes if the pipeline name or task name or task dependency changes. The version number is used as a directory name, which is created for saving all relevant information about the task.

C. Configuration Management Layer

Built upon the dependency management layer, the configuration management layer features a flexible programming interface for users to specify the whole configuration space for massive computational experiments. It specifies the set of pipelines along with configurations, serving as the input to the dependency management layer. It is also responsible for harvesting the results from all the executed pipelines for downstream analytics. To manage massive computational experiments, we need to frequently modify and automatically generate the configurations. We implement a configuration manager to handle this task.

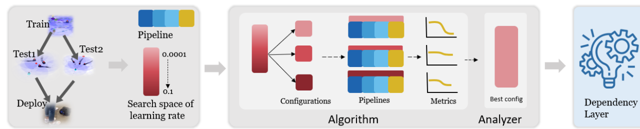

Fig. 5. The configuration management layer, managing a turtlebot pipeline. Given pipeline representation and search space of configurations, the algorithm module suggests configurations to pipelines, and the analyzer module finds the configuration with the best metric.

As shown in Figure 5, the configuration layer consists of three components. (i) The configuration component provides a flexible programming interface for users to describe the search space. Like hyperparameters of models (batch size or learning rate). Given configuration space and pipeline, algorithm, and analyzer component together automates the process of running pipelines. (ii) The algorithm component generates and schedules configurations for pipelines according to the given search space. (iii) The analyzer component aims at finding the best configurations for the pipelines.

D. System Implementation

The implementation of the computation layer, the dependency management layer, and the configuration management layer comprises 92% in Python, 5% in Rust, and 3% in Shell scripts. Specifically, the computation layer abstracts three clouds, including AWS batch service, Ray Cluster service, and Kubernetes Cluster Service. We set up Ray Cluster and Kubernetes Cluster on two on-premise clusters and connect them with real robots. To access cloud service and cluster service, we use AWS Boto3 library, Kubernetes command lines, and Ray Job/Core APIs. For transferring data between different clouds, we use network filesystems like AWS EFS and SSHFS to mount data from different clouds. Then, for the dependency management layer, we exploit Python Hashlib to manage tasks in pipelines. To perform scalable configuration searching, we exploit Ray library to run, monitor, and analyze pipelines.

IV. Experimental Results

We exploit EMS® to develop an offline reinforcement learning(RL) pipeline for training mobile robot navigation policy and demonstrate its productivity, scalability, and reproducibility. Also, we develop three realistic robotics applications to demonstrate that EMS® is applicable to numerous intelligent robots. In particular, we aim to answer the following questions: (1) How well does EMS® improve productivity and scalability upon an evaluation baseline? (2) What advantages do the computation layer, dependency management layer, and configuration management layer provide for EMS®? (3) Is it difficult to connect EMS® 's applicability to robotics?

A. Experiment Setting

We train a decision transformer [85] based robot agent in the Four Rooms environment of Gym Minigrid [86], where the agent navigates in a maze composed of four rooms interconnected by four gaps in the walls. We implement the problem using a four-task pipeline. Firstly, a data acquisition task that applies PPO [87] to interact with the Four Rooms environment to collect data. Secondly, a data preprocessing task which transforms collected data (represented by state, action, reward) into a trajectory representation. Thirdly, a model training task that trains a decision transformer model using one hyperparameter set (experiment seed, head number, and embedding size) of the transformer model. Fourthly, an evaluation task that evaluates the trained model in the Four Rooms environments over 100 goals and reports accumulated rewards of the model in each goal. More details are referred to in the original paper [85]. We implement EMS® on four cloud computing environments: the AWS Batch computing, a Kubernetes cluster, a Ray cluster, and a robot cluster. To make consistent benchmarks, we deploy all these environments on the AWS EC2 p3.8xlarge (Ubuntu 20.04) instances[1]. To simulate a multi-cloud setting, we deploy the four computing clusters in four different AWS VPCs and connect them through AWS site-to-site VPN.

B. Overall Performance

We run a robotics daily routine pipeline on three systems (Naive, Single-cloud EMS, EMSR) and compare their productivity, scalability, and reproducibility. Specifically, Naive runs decision transformer's original code from [85] on one AWS EC2 p3.8xlarge instance. Single cloud-EMS runs the robotic pipeline in the subsectionIV-A using EMS® but only enabling AWS batch computing. EMSR is the full-fledged implementation of EMS®. We use these systems to run

[1] We also use the same instance for AWS Batch job submission

16 pipelines at the beginning of every hour from 1:00 pm to 10:00 pm. Each pipeline conducts model training with different hyperparameters (experiment seed, head number, and embedding size) using 1 GPU, and reports the averaged time duration, money cost, and evaluation reward of pipelines every hour. In addition, we use three systems to repeatedly run the pipeline at 2:00 pm multiple times (8, 16, 24, 32) within one hour and report the averaged time duration and money cost, evaluation reward of pipelines.

We collect 3 evaluation metrics: (i) We measure **productivity** by total_output and total_input where total_input is the averaged money cost and duration time of running pipelines, and total_output is averaged evaluation reward of running pipelines. Given the same total_output, less total_input means more productivity. (ii) We measure **scalability** by money cost and duration time of different pipeline frequency, which is the total number of pipeline runs within one hour. (iii) We measure **reproducibility** by the evaluation reward of running pipelines. Moreover, the money cost on simulated on-premise devices is considered free and not counted towards instance hours[2].

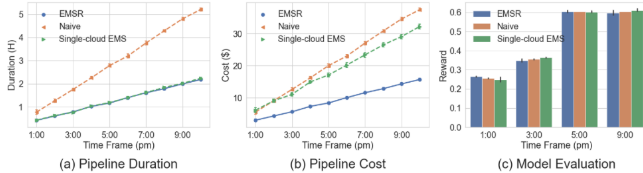

Fig. 6. Productivity of EMS® in a robotics daily routine pipeline. We run the pipeline every 1 hour from 1:00 pm to 10:00 pm and report the pipeline duration in (a), the cost in (b), and evaluation rewards in (c). The blue line and bar represent EMS®, implemented on an on-premise cluster and AWS batch service. The green line and bar represent single-cloud EMS, implemented by EMS® only enabling AWS batch service. The orange line and bar represent the naive pipeline, implemented on one EC2 instance. As expected, EMS® requires less time and cost than the other methods.

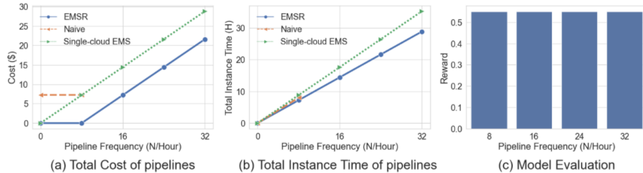

Fig. 7. Scalibility and reproducibility of EMS® in a robotics daily routine pipeline. We run different number of pipelines every hour and report averaged duration in (a), averaged cost in (b), averaged evaluation rewards in (c). The blue line and bar represents EMS®, implemented on an on-premise cluster and AWS batch service. The green line and bar represents single-cloud EMS, implemented on EMS® only enabling AWS batch service. As expected, EMS® requires less time and cost for running the same number of pipelines than the others. Also, EMS® reproduces the model performance by running the pipeline multiple times.

We reports producitivity in Figure 6, and reports scalibility and reproducibility in Figure 7. We highlight three key findings: (1) EMS® improves the productivity of naive approach and single cloud-based system due to its sky-based architecture. Single-cloud EMS's cost is lower than Naive since its version controller in the dependency manage layer enables reducing duration by recycling the data preprocessing (2) EMS® has the highest scalability due to its sky-based computation layer optimizes the cost by scheduling pipeline on an on-premise device (3) EMS® reproduces the pipeline results with different number of pipeline runs.

C. Performance Gain from Each of the Abstraction Layers

Computation layer. A key benefit of the sky-based computation layer is to allow EMS® to scale jobs while maintaining low time duration and money cost. In Figure 8a and Figure 8b, we evaluate this ability on parallel workloads of empty jobs, where each job requests 1 CPU hour. Compared with 3 existing job management systems (AWS-Batch, Kubernetes-Batch, Ray-Job), we observe that the total time duration of EMS® near-perfectly stays the same when the total number of jobs increases. This is due to the sky-based computation leveraging public computing for parallel-jobs scheduling. Since Kubernetes-batch and Ray-job are deployed on an EC2 instance with only 96 CPU cores to schedule the job, the time increases linearly with respect to the number of jobs. In Figure 8b, we observe that the instance hours of the computation layer are lower than the others. This is due to the sophisticated data orchestrator reducing the time of resource transfer.

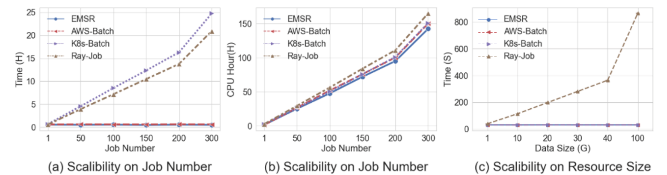

Fig. 8. Scalability of computation layer in EMS®. We report total time duration in (a) and total instance hours of jobs in (b) with increasing job numbers. We also report single job time duration with increasing data size of jobs in (c). The blue line represents the computation layer, the red line represents the AWS Batch computing, the purple line represents the Kubernetes Batch job and the brown line represents Ray Job. The computing environment of Kubernetes Batch job and Ray Job is set up with mounted data volumes.

Another benefit of the computation layer is the ability to scale the dataset size of a job while maintaining a low time duration. In Figure 8c, we see that EMS®'s job duration almost stays the same even dataset size of the job increases. Compared to Ray-job, it outperforms in two magnitudes of order. The data orchestrator utilizes a networked file system to mount the source file to the jobs and this reduces the total data transfer time.

Dependency management layer. To evaluate the performance of the dependency layer, we track the data preprocessing time of the pipeline experiment from Section IV-B and compare it with Naive. In Figure 10a, we observe that the time of EMS® stays almost the same with the dataset size increases while Naive increases linearly. This is because the version controller of EMS® tracks preprocessed results from the previous pipeline runs and the new pipeline only spends time on the newly arrived data from the data collection task.

Also, the dependency management layer brings EMS® the ability to schedule sky-based pipeline as demonstrated in

[2]In reality, there is a tiny cost on maintaining the on-premise cluster, but this detail does not change the main conclusion drawn from these experiments.

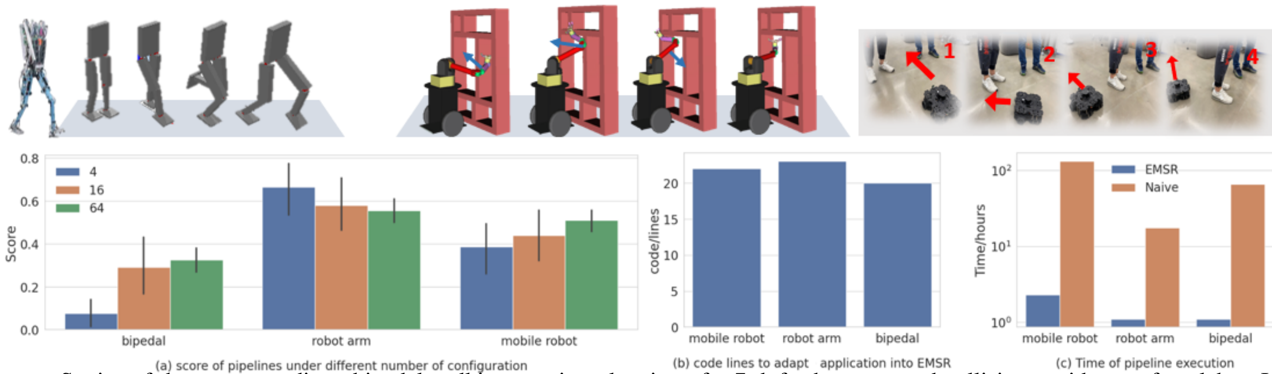

Fig. 9. Statics of three case studies - bipedal walking, motion planning of a 7-dof robot arm, and collision avoidance of turtlebot. In (a), we report the average score of pipelines under different configurations (4,16,64). The score of bipedal and turtlebot is normalized reward and the score of the robot arm is normalized path cost. We also report the number of code lines for adapting naive implementation into EMS® and the time duration in hours of 64 pipeline execution.

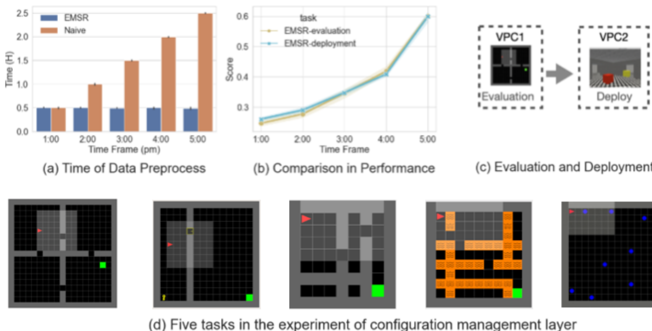

Fig. 10. Performance of the pipeline management layer in EMS®. In (a), we report the data preprocessing duration of pipelines running on different time frames. In (b), we also report the evaluation score and deployment score of the trained models in pipelines. (c) illustrates the experiment setting of (b), where the evaluation cloud and deployment cloud is different. From left to right in (d), they are Four Rooms, Door Keys, Simple Crossing and Lava Crossing, Dynamics Obstacle Avoidance.

Figure 10b, allowing us to evaluate models on one cloud (VPC1) and deploy the models on another cloud (VPC2).

Configuration management layer. The configuration management layer is important for hyperparameter search in many data-driven workloads.We run the same pipeline in Section IV-B for five different tasks, as shown in Figure 10d. Specifically, for each task, we search configuration among 1, 8, 16, 32, and 64 sets of hyperparameters, which are head number, layer number, embedding size, activation functions, batch size, and experiment seeds for the decision transformer model. For each set, we report the best model evaluation reward among the hyperparameters. We report the average evaluation reward and variance of the performance on 100 goals in Table II and observe that a larger amount of hyperparameters delivers better-performed models.

TABLE II. Best evaluation score of decision transformer models in different hyperparameter number.

Number	Four Rooms	Door Keys	Simple Crossing	Lava Crossing	Dynamic Obstacles
1	0.460 ± 0.03	0.892 ± 0.08	0.789 ± 0.28	0.798 ± 0.01	0.654 ± 0.11
8	0.481 ± 0.05	0.934 ± 0.05	0.820 ± 0.12	0.855 ± 0.02	0.685 ± 0.09
16	0.520 ± 0.03	0.954 ± 0.07	0.824 ± 0.11	0.909 ± 0.03	0.720 ± 0.12
32	0.523 ± 0.02	0.953 ± 0.08	0.850 ± 0.10	0.920 ± 0.03	0.719 ± 0.08
64	0.524 ± 0.03	0.960 ± 0.02	0.888 ± 0.12	0.920 ± 0.02	0.722 ± 0.10

D. Case Studies

We exploited EMS® to develop three data-driven robotics applications. **(i)** Developing collision avoidance policy of turtlebot robot. This pipeline consists of training collision avoidance policy in Stage simulation [88] with PPO, testing in stage, and deployment on a real robot [89]. **(ii)** Developing motion planning policy of a 7-dof robot arm in OpenRave simulation. This pipeline consists of collecting datasets in OpenRave [90] with BIT* [91], training NEXT [92] motion plan policy, and testing in simulation. **(iii)** Developing a bipedal walking policy. We model Flame robot [93], [94] in pybullet [95], train walking policy using PPO, and testing policy in pybullet. The details of these implementation are referred to in paper [26], [91], [96], [97]. Specifically, the naive implementation of these applications can only leverage computation on a AWS EC2 instance of p3.8xlarge and EMS® integration can leverage AWS batch computation to run pipelines. Each running pipeline of them has one configuration and requires resource of one p3.8xlarge instance.

In the experiment, we run 4,16,64 pipelines with different configurations(seed, batch size. and learning rate) and report average score, total execution time and code line number in Figure 9. As shown in subfigure 9a, the average score is higher with more configuration of pipelines, which justifies the importance of massive computation in EMS®. Also, subfigure 9b shows that adapting existing robotic applications EMS® only requires a few of code lines, which implies that EMS® is applicable to numerous intelligent robotics applications. Then we compare the time cost of EMS® with naive implementation and demonstrate the less time cost of EMS® than the naive with the same pipeline workloads.

V. Conclusion

We propose EMS®, a cloud-enabled massive computational experiment management system supporting massive data-driven robotic routine experiments. By exploiting it to develop an offline reinforcement learning pipeline for mobile robot navigation, we demonstrate the scalability, reproducibility, and productivity of EMS®. Then, we show that EMS® is applicable to numerous data-driven robotics applications, through three realistic robot examples. Future works will consider developing more sophisticated methods to optimize the cost of computation scheduling and integrate EMS®with a robotic deployment system in a continuous integration and continuous delivery (CI/CD) fashion.

References

[1] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, p. eaau5872, 2019.

[4] A. Bouman, M. F. Ginting, N. Alatur, M. Palieri, D. D. Fan, T. Touma, T. Pailevanian, S.-K. Kim, K. Otsu, J. Burdick *et al.*, "Autonomous spot: Long-range autonomous exploration of extreme environments with legged locomotion," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 2518–2525.

[5] S. Gangapurwala, M. Geisert, R. Orsolino, M. Fallon, and I. Havoutis, "Real-time trajectory adaptation for quadrupedal locomotion using deep reinforcement learning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 5973–5979.

[6] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, "Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3699–3706, 2020.

[7] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.

[8] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[9] Z. Xie, X. Da, M. van de Panne, B. Babich, and A. Garg, "Dynamics randomization revisited: A case study for quadrupedal locomotion," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 4955–4961.

[10] O. Nachum, M. Ahn, H. Ponte, S. Gu, and V. Kumar, "Multi-agent manipulation via locomotion using hierarchical sim2real," *arXiv preprint arXiv:1908.05224*, 2019.

[11] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," *arXiv preprint arXiv:2004.00784*, 2020.

[12] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[13] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 3406–3413.

[14] J. Mahler, F. T. Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Kröger, J. Kuffner, and K. Goldberg, "Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 1957–1964.

[15] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar, "Deep dynamics models for learning dexterous manipulation," in *Conference on Robot Learning*. PMLR, 2020, pp. 1101–1112.

[16] E. Stengel-Eskin, A. Hundt, Z. He, A. Murali, N. Gopalan, M. Gombolay, and G. Hager, "Guiding multi-step rearrangement tasks with natural language instructions," in *Conference on Robot Learning*. PMLR, 2022, pp. 1486–1501.

[17] R. Strudel, A. Pashevich, I. Kalevatykh, I. Laptev, J. Sivic, and C. Schmid, "Learning to combine primitive skills: A step towards versatile robotic manipulation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4637–4643.

[18] A. X. Lee, S. Levine, and P. Abbeel, "Learning visual servoing with deep features and fitted q-iteration," *arXiv preprint arXiv:1703.11000*, 2017.

[19] Q. Bateux, E. Marchand, J. Leitner, F. Chaumette, and P. Corke, "Training deep neural networks for visual servoing," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3307–3314.

[20] P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson, "Implicit behavioral cloning," in *Conference on Robot Learning*. PMLR, 2022, pp. 158–168.

[21] O. Mees and W. Burgard, "Language-conditioned policy learning for long-horizon robot manipulation tasks."

[22] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 5129–5136.

[23] B. Shacklett, E. Wijmans, A. Petrenko, M. Savva, D. Batra, V. Koltun, and K. Fatahalian, "Large batch simulation for deep reinforcement learning," *arXiv preprint arXiv:2103.07013*, 2021.

[24] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, L. Chen, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska, "One thousand and one hours: Self-driving motion prediction dataset," *arXiv preprint arXiv:2006.14480*, 2020.

[25] H. Surmann, C. Jestel, R. Marchel, F. Musberg, H. Elhadj, and M. Ardani, "Deep reinforcement learning for real autonomous mobile robot navigation in indoor environments," *arXiv preprint arXiv:2005.13857*, 2020.

[26] G. Ye, Q. Lin, T.-H. Juang, and H. Liu, "Collision-free navigation of human-centered robots via markov games," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 11 338–11 344.

[27] K. Fang, A. Toshev, L. Fei-Fei, and S. Savarese, "Scene memory transformer for embodied agents in long-horizon tasks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 538–547.

[28] N. Savinov, A. Dosovitskiy, and V. Koltun, "Semi-parametric topological memory for navigation," *arXiv preprint arXiv:1803.00653*, 2018.

[29] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2616–2625.

[30] Y. Lyu, Y. Shi, and X. Zhang, "Improving target-driven visual navigation with attention on 3d spatial relationships," *Neural Processing Letters*, pp. 1–20, 2022.

[31] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, no. 4, pp. 681–694, 2020.

[32] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[33] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[34] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[35] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[36] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[37] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[38] A. Team, "Alphastar: mastering the real-time strategy game starcraft ii," *DeepMind blog*, vol. 24, 2019.

[39] M. Mathieu, S. Ozair, S. Srinivasan, C. Gulcehre, S. Zhang, R. Jiang, T. Le Paine, K. Zolna, R. Powell, J. Schrittwieser *et al.*, "Starcraft ii unplugged: Large scale offline reinforcement learning," in *Deep RL Workshop NeurIPS 2021*, 2021.

[40] D. Jiang, E. Ekwedike, and H. Liu, "Feedback-based tree search for reinforcement learning," in *International conference on machine learning*. PMLR, 2018, pp. 2284–2293.

[41] H. Monajemi, D. L. Donoho, and V. Stodden, "Making massive computational experiments painless," in *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 2016, pp. 2368–2373.

[42] "Codalab worksheets," worksheets.codalab.org.

[43] "Pywren," pywren.io.

[44] A. V. Joshi, "Amazon's machine learning toolkit: Sagemaker," in *Machine Learning and Artificial Intelligence*. Springer, 2020, pp. 233–243.

[45] ——, "Azure machine learning," in *Machine Learning and Artificial Intelligence*. Springer, 2020, pp. 207–220.

[46] E. Bisong, "An overview of google cloud platform services," *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pp. 7–10, 2019.

[47] ——, "Kubeflow and kubeflow pipelines," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Springer, 2019, pp. 671–685.

[48] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan *et al.*, "Ray: A distributed framework for emerging {AI} applications," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 561–577.

[49] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[50] Q. Lin, G. Ye, J. Wang, and H. Liu, "Roboflow: a data-centric workflow management system for developing ai-enhanced robots," in *Conference on Robot Learning*. PMLR, 2022, pp. 1789–1794.

[51] A. Richie-Halford and A. Rokem, "Cloudknot: A python library to run your existing code on aws batch," in *Proceedings of the 17th python in science conference*, 2018, pp. 8–14.

[52] H. Monajemi, R. Murri, E. Jonas, P. Liang, V. Stodden, and D. L. Donoho, "Ambitious data science can be painless," *arXiv preprint arXiv:1901.08705*, 2019.

[53] "K8s batch," kubernetes.io/docs/concepts/workloads/controllers/job.

[54] "Aws batch," https://aws.amazon.com/batch.

[55] D. Xin, H. Miao, A. Parameswaran, and N. Polyzotis, "Production machine learning pipelines: Empirical analysis and optimization opportunities," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2639–2652.

[56] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc *et al.*, "Tfx: A tensorflow-based production-scale machine learning platform," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1387–1395.

[57] C. Arisdakessian, S. B. Cleveland, and M. Belcaid, "Metaflow—mics: Scalable and reproducible nextflow pipelines for the analysis of microbiome marker data," in *Practice and Experience in Advanced Research Computing*, 2020, pp. 120–124.

[58] R. Garreta, G. Moncecchi, T. Hauck, and G. Hackeling, *Scikit-learn: machine learning simplified: implement scikit-learn into every step of the data science pipeline*. Packt Publishing Ltd, 2017.

[59] O. Saha and P. Dasgupta, "A comprehensive survey of recent trends in cloud robotics architectures and applications," *Robotics*, vol. 7, no. 3, p. 47, 2018.

[60] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on automation science and engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[61] K. Goldberg and B. Kehoe, "Cloud robotics and automation: A survey of related work," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-5*, 2013.

[62] S. Shakya *et al.*, "Survey on cloud based robotics architecture challenges and applications," *Journal of Ubiquitous Computing and Communication Technologies (UCCT)*, vol. 2, no. 01, pp. 10–18, 2020.

[63] Y. Liu and Y. Xu, "Summary of cloud robot research," in *2019 25th International Conference on Automation and Computing (ICAC)*. IEEE, 2019, pp. 1–5.

[64] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, and G. W. Kit, "Davinci: A cloud computing framework for service robots," in *2010 IEEE international conference on robotics and automation*. IEEE, 2010, pp. 3084–3089.

[65] "Fetch robotics," fetchrobotics.com.

[66] "Formant robotics," formant.io.

[67] M. Waibel, M. Beetz, J. Civera, R. d'Andrea, J. Elfring, D. Galvez-Lopez, K. Häussermann, R. Janssen, J. Montiel, A. Perzylo *et al.*, "Roboearth," *IEEE Robotics & Automation Magazine*, vol. 18, no. 2, pp. 69–82, 2011.

[68] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea, "Rapyuta: The roboearth cloud engine," in *2013 IEEE international conference on robotics and automation*. IEEE, 2013, pp. 438–444.

[69] Y. Liang, N. Jha, J. Ichnowski, M. Danielczuk, J. Gonzalez, J. Kubiatowicz, K. Goldberg *et al.*, "Fogros: An adaptive framework for automating fog robotics deployment," *arXiv preprint arXiv:2108.11355*, 2021.

[70] J. Ichnowski, K. Chen, K. Dharmarajan, S. Adebola, M. Danielczuk, V. Mayoral-Vilches, H. Zhan, D. Xu, R. Ghassemi, J. Kubiatowicz *et al.*, "Fogros 2: An adaptive and extensible platform for cloud and fog robotics using ros 2," *arXiv preprint arXiv:2205.09778*, 2022.

[71] J. M. Stauffer, "A smart and interactive edge-cloud big data system," Ph.D. dissertation, Purdue University Graduate School, 2022.

[72] K. Nishimiya and Y. Imai, "Serverless architecture for service robot management system," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 11 379–11 385.

[73] "Aws robomaker," aws.amazon.com/robomaker.

[74] "Confluent," www.confluent.io.

[75] R. Ilijason, "Getting started with databricks," in *Beginning Apache Spark Using Azure Databricks*. Springer, 2020, pp. 27–38.

[76] B. Dageville, T. Cruanes, M. Zukowski, V. Antonov, A. Avanes, J. Bock, J. Claybaugh, D. Engovatov, M. Hentschel, J. Huang *et al.*, "The snowflake elastic data warehouse," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 215–226.

[77] "Trifacta," www.trifacta.com.

[78] G. Sayfan, *Mastering kubernetes*. Packt Publishing Ltd, 2017.

[79] "Google anthos," cloud.google.com/anthos.

[80] "Azure arc," azure.microsoft.com/services/azure-arc.

[81] "Aws outposts," aws.amazon.com/outposts.

[82] K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes, "Sky computing," *IEEE Internet Computing*, vol. 13, no. 5, pp. 43–51, 2009.

[83] A. Monteiro, C. Teixeira, and J. S. Pinto, "Sky computing: Exploring the aggregated cloud resources—part ii," in *2014 9th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 2014, pp. 1–6.

[84] M. Garnaat, "boto documentation," 2018.

[85] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *Advances in neural information processing systems*, vol. 34, 2021.

[86] M. Chevalier-Boisvert, L. Willems, and S. Pal, "Minimalistic gridworld environment for openai gym," https://github.com/maximecb/gym-minigrid, 2018.

[87] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[88] R. Vaughan, "Massively multi-robot simulation in stage," *Swarm intelligence*, vol. 2, no. 2, pp. 189–208, 2008.

[89] R. Amsters and P. Slaets, "Turtlebot 3 as a robotics education platform," in *International Conference on Robotics in Education (RiE)*. Springer, 2019, pp. 170–181.

[90] R. Diankov, "Automated construction of robotic manipulation programs," 2010.

[91] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Batch informed trees (bit*): Informed asymptotically optimal anytime search," *The International Journal of Robotics Research*, vol. 39, no. 5, pp. 543–567, 2020.

[92] B. Chen, B. Dai, Q. Lin, G. Ye, H. Liu, and L. Song, "Learning to plan in high dimensions via neural exploration-exploitation trees," *arXiv preprint arXiv:1903.00070*, 2019.

[93] J. H. Solomon, M. A. Locascio, and M. J. Hartmann, "Linear reactive control for efficient 2d and 3d bipedal walking over rough terrain," *Adaptive Behavior*, vol. 21, no. 1, pp. 29–46, 2013.

[94] D. Hobbelen, T. De Boer, and M. Wisse, "System overview of bipedal robots flame and tulip: Tailor-made for limit cycle walking," in *2008 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2008, pp. 2486–2491.

[95] C. Erwin and B. Yunfei, "Pybullet a python module for physics simulation for games," *PyBullet*, 2016.

[96] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6252–6259.

[97] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, J. Gonzalez, K. Goldberg, and I. Stoica, "Ray rllib: A composable and scalable reinforcement learning library," *arXiv preprint arXiv:1712.09381*, vol. 85, 2017.