Learning Scheduling Algorithms for Data Processing Clusters

Tejas Ravindra Dhawale

May 25, 2020

Efficient cluster scheduling is a complex task requiring complex algorithms and Mao et al. [1] demonstrate how machine learning techniques can be used for it. In their work, Mao et al. have proposed Decima, a reinforcement learning (RL) and neural network based cluster scheduling technique with a specified high level objective (e.g. minimize average job completion time (JCT)).

1 Introduction

Computing on cluster resources is expensive and even slight improvement in the performance can lead to massive cost savings. The existing cluster schedulers do not take into consideration the readily available job structure information. Incorporating workload specific information in scheduling policies requires expert knowledge and efforts. Decima, on the other hand, uses workload specific information to automatically learn efficient scheduling policy. Decima uses a graph neural network to scale to input jobs of arbitrary size and an RL agent, which takes the scheduling decision with the help of a neural network.

2 Motivation

Data processing and query compilers create a directed acyclic graph (DAG) structured job, which consists of interdependent processing stages. The main challenges of using the information embedded in the job DAGs are illustrated below.

2.1 Dependency-aware task Scheduling

A job DAG consists of numerous stages that are interdependent on each other. Taking this dependency into account and ensuring that no stages have to wait for their completion even if sufficient resources are available is algorithmically difficult.

2.2 Setting right level of parallelism

Assigning more resources to complete the processing of the job stages decreases the job runtime. However, for each job, there is a sweet spot after which assigning more resources results in diminishing gains.

3 Design Challenges

Decima is a framework for automatically scheduling DAG structured jobs. For better understanding, the design of Decima is specified in the context of the Spark processing system. A Spark job consists of stages given by connected nodes in the job DAG and each stage represents an operation that the system runs in parallel over multiple shards of the stage's input data, with each shard being processed by a single task [1]. Executors are assigned by the Spark master to process the tasks. Decima represents the scheduler as an agent that takes as input the current state of the DAG and determines which stage the executor should work on using a neural network (policy network).

The key design challenges for Decima are:

1. **Scalable state information processing.**
   The cluster needs to handle job DAGs with arbi-

trary shape and size. The scheduler must thus consider a large amount of dynamic information for making scheduling decisions.

2. **Huge space of scheduling decisions.**
   Owing to multiple options for selecting an action the RL agent must explore the large search space to learn a good policy.

3. **Training for continuous stochastic job arrivals.**
   Continuous job arrivals cause variance in the received rewards and make it difficult to train RL algorithms, which requires fixed length episodes.

4 Design

Figure 1 describes the design of Decima and how it addresses the challenges mentioned in (§3).

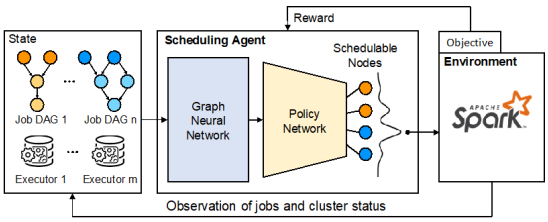


Figure 1: Decima's RL framework

4.1 Scalable state information processing

Decima addresses the scalability issue (§3) using graph neural network, which takes as input the job DAGs and reduces the input state space by converting the state information into three different types of embedding:

1. **Per-node embedding** which gives the details of the node and aggregates the information from its children.

2. **Per-job embedding** which aggregates the information of all the nodes inside the job DAG.

3. **Global embedding** which aggregates the information of all the per-job embeddings.

Each level of embedding uses two non-linear functions because with a single non-linear function the graph neural network cannot compute the critical path of a DAG. This embedded output feature vector is then fed as input to the policy network.

4.2 Encoding scheduling decisions as actions

Decima handles the challenge of large action space (§3) for selecting an action using the policy network, which is based on the input feature vector from graph neural network using the following approaches:

1. **Stage selection**
   Whenever a scheduling event (e.g., a stage completion, or a job arrival) occurs, Decima uses a softmax operation to calculate the probability of selecting a stage for execution.

2. **Parallelism limit**
   In contrast to most of the existing scheduling algorithms, which use stage level parallelism, Decima implements job level parallelism. Decima uses a softmax operation to determine the probability of selecting the parallelism limit.

4.3 Training

Decima uses a policy gradient algorithm to learn by performing gradient descent on the neural networks using rewards observed in training [1]. Decima's policy is poor if it trains only on batch inputs. So in order to make Decima's policy more robust, it is trained on a job arriving at different intervals of time (§3). RL agent makes poor decisions in early training stages and with an unbounded stream of incoming jobs, the agent may take time in training on such data and might lead to accumulation of job backlog. To avoid this, the training episodes are terminated at the beginning so that the agent can learn on short job sequences first, and then the size of the episodes is gradually increased so that the agent learns more challenging arrival sequences. The variance and impedance in the training process are minimized by calculating the baselines for policy gradient algorithm over episodes with the same arrival sequence instead of baselines computed by averaging over episodes with different arrival sequence [2].

5 Implementation

Decima is implemented as a pluggable scheduling service with Spark. Figure 2 shows the modifications made for integrating Decima with Spark.

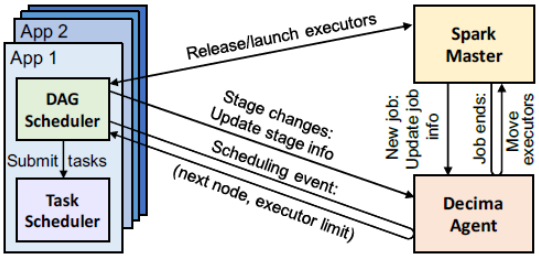


Figure 2: Spark standalone cluster architecture, with Decima additions highlighted [1]

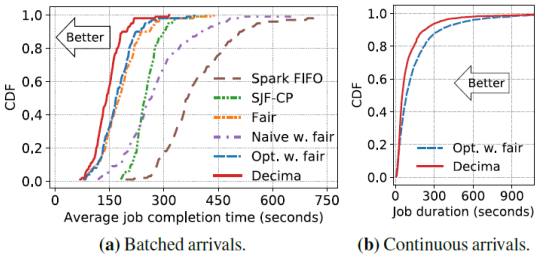
1. DAG scheduler contacts Decima agent whenever a new scheduling event occurs and Decima agent returns the next stage to execute and the parallelism limit.
2. The Spark master contacts Decima agent when a new job arrives to update the job information and Decima returns the number of executors to be allocated to the job.
3. Executors are associated with a job until completion. Once the job is completed, Decima informs the Spark master to which job, the executors should be moved to.

Neural network architecture: The graph neural network and the policy network are implemented using two hidden layers with 32 and 16 hidden units on each layer. Decima is a lightweight model (50KB) and it takes around 15ms for mapping cluster state to scheduling decision [1].

6 Evaluation

Figure 3 shows the evaluation of the performance of Decima with different baseline cluster scheduling algorithms [1].

Batched arrivals: Figure 3a shows evaluation results of over 100 experiment samples from 6 different input sizes (2.5.10.20.50. and 100 GB) and all



(a) Batched arrivals. (b) Continuous arrivals.

Figure 3: Performance evaluation of Decima vs. Baseline Algorithms [1]

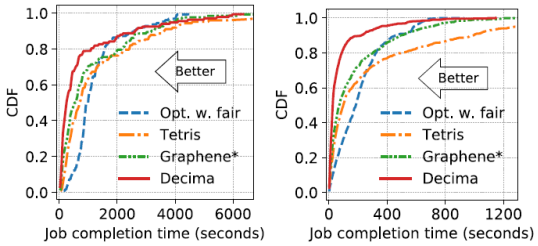
22 TPC-H [1] queries. All the baseline algorithms underperform in comparison with Decima, which shows an improvement in average JCT by 21% over its closest rival optimized weighted fair scheduling.

Continuous arrivals: Figure 3b shows the evaluation results of 1000 TPC-H job samples with six different input sizes with an average interarrival time of 45 seconds. Decima's average JCT is 29% lower than optimized weighted fair scheduling algorithms with other algorithms in 3a unable to handle continuous job arrivals. Decima performs exceptionally well as compared to optimized weighted fair scheduling during high work load where taking scheduling decisions are crucial.

Multi-dimensional resource packing: There are some advanced cluster schedulers, which unlike Spark allows to specify resource requirement (e.g., CPU, memory). Decima's environment is modified to contain different executor classes with different resource specifications. A task can run only in an executor class, which is equal to or larger than the requested resource by the task [3]. Decima now also selects an executor class along with stage selection and parallelism limit. Figure 4a shows evaluation results of 20,000 jobs from Alibaba production cluster where Decima outperforms its closest rival Graphene* with 32% lower average JCT. Figure 4b shows evaluation results with TPC-H workload in which Decima shows a performance improvement of 43% over Graphene*.

7 Decima's Key Ideas:

This section demonstrates the impacts of the idea developed for Decima.



(a) Industrial trace replay. **(b)** TPC-H workload.

Figure 4: Multi resource sharing performance evaluation [1]

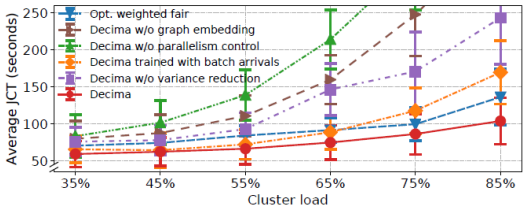


Figure 5: Breakdown of Decima's key ideas [1]

1. **Parallelism Limit:** .
   Specifying the number of executors for each job (§4.2) is vital for Decima to learn high quality policies. By excluding the parallelism limit, the RL agent will assign all the executors to a single DAG stage worsening Decima's policy.

2. **Graph Embeddings :**
   Without graph embeddings (§4.1), Decima is unaware of the time taken for the completion of child stages and jobs while making scheduling decisions thus leading to poor policy.

3. **Variance reduction:**
   Variance reduction (§4.3) improves the performance of Decima by 2x and is a key factor for Decima to learn good policies.

4. **Training on continuous job arrivals:**
   With training only on batched inputs, Decima learns to defer the execution of large jobs leading to long job backlog. Training on continuous job (§4.3) arrivals helps Decima to generalise and learn a more robust policy.

Figure 5 shows that excluding even a single parameter leads to a poor policy of Decima in comparison with optimized weighted fair scheduling. For example, optimized weighted fair scheduling assigns executors in proportion to the job size and for Decima without the knowledge of job size, it may assign less number of executors to larger jobs, consequently leading to poor performance.

8 Future Work

The evaluation results in (§6) are based on job duration related metrics (e.g., average JCT). Training Decima on different aspect like meeting job deadlines might guide it to learn a deadline aware policy.

Using multi-agent RL algorithms, preemptive scheduling can be performed with Decima.

9 Conclusion

The key ideas innovated for Decima can find their use in a wide variety of applications. Decima demonstrates the use of reinforcement learning and neural networks to automatically learn complex cluster scheduling policies.

References

[1] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizade. *Learning Scheduling Algorithms for Data Processing Clusters*. Proceedings of the 2019 ACM SIGCOMM Conference, 2019.

[2] H. Mao, S. B. Venkatakrishnan, M. Schwarzkopf, and M. Alizadeh. *Variance Reduction for Reinforcement Learning in Input-Driven Environments*. Proceedings of the 7th International Conference on Learning Representations (ICLR), 2016.

[3] R. G. G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. *Multi-resource Packing for Cluster Schedulers*. In Proceedings of the 2014 ACM SIGCOMM Conference (SIGCOMM). 455–466., 2014.