Learning Graph-based Cluster Scheduling Algorithms

Hongzi Mao Malte Schwarzkopf Shaileshh Bojja Venkatakrishnan Mohammad Alizadeh
MIT CSAIL

Abstract. Cluster schedulers today rely on coarse-grained, generalized heuristics and on extensive manual tuning. This can be costly: they may run computations inefficiently or unnecessarily leave resources idle. Modern reinforcement learning techniques, however, allow future cluster schedulers to *learn* their own scheduling policy instead of using an off-the-shelf policy, with significant benefits.

We introduce Clotho, a system that uses reinforcement learning to automatically train scheduling policies for high-level objectives without encoding human-engineered heuristics or making assumptions about the workload. Clotho learns to schedule jobs that consist of graphs of tasks, setting both parallelism level and execution order based purely on experience. Our Clotho prototype scheduler for Spark outperforms the best heuristics — which already run jobs 2× faster than Spark's default scheduler — by another 20–33%. The policies it learns generalize across cluster sizes and workload variation.

1 Introduction and Motivation

Good cluster scheduling policies are crucial to efficiently utilize data center resources: even small improvements in utilization can save thousands of dollars at scale. Current cluster schedulers use general-purpose heuristics that prioritize easy understanding and implementation over achieving the ideal performance on a specific workload. For example, giving extra resources to a particular job (*e.g.*, by relaxing fair sharing [2]) may help it speed through a bottleneck; understanding the workload can help the scheduler plan ahead (*e.g.*, anticipate future job arrivals); and setting workload-specific threshold values can reduce idle resources (*e.g.*, for delay scheduling [6]). However, such workload-specific policies are rarely used in practice because they require expert knowledge and significant effort to devise, implement, and validate. We show that modern machine learning techniques can help avoid this burden by *automatically learning* a scheduling policy for a workload and high-level objective (*e.g.*, minimal average job runtime) without explicit policy input.

Cluster scheduling is a good fit for machine learning: plentiful training data already exists in the form of monitoring information, small inaccuracies in decisions are often tolerable, and repetitive workloads allow comparing different decisions' goodness. Clotho is a first step towards using neural networks to learn cluster scheduling policies from scratch, focusing particularly on batch-oriented parallel data analytics jobs. To achieve this, we had to devise (*i*) new representations of the cluster state appropriate for learning, (*ii*) new training techniques to deal with complex, graph-shaped inputs representing data-parallel jobs, and (*iii*) a model that combines global and job-local scheduling decisions, and accurately represents the behavior of a real-world data-parallel processing framework. Unlike other research projects — *e.g.*, for learning multi-dimensional resource packing [3], or for learning device placement for TensorFlow workloads [4] — Clotho learns complete, complex policies purely from experience and observations of the cluster state.

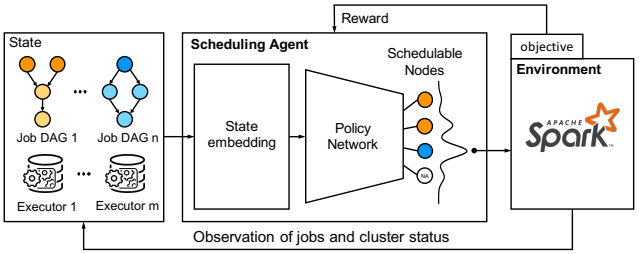


Figure 1: Clotho uses reinforcement learning (RL) to train an end-to-end system for a high-level objective. Its graph-based state embedding combines DAGs and feeds a policy network that makes the next scheduling decision.

2 Design

Figure 1 shows the overall architecture Clotho uses to learn scheduling policies via **reinforcement learning**: a *scheduling agent* observes the *state* (*viz.*, available jobs and resources) of an *environment* (*e.g.*, a Spark cluster). When the cluster state changes, the agent uses several neural networks to choose an *action*: either a scheduling decision or doing nothing. The action affects the environment, and Clotho uses a high-level *objective* specified by the cluster administrator (*e.g.*, minimize average job completion time) to compute a *reward* that the agent receives for the chosen action. Actions better aligned with the objective yield higher rewards, and the neural networks consequently learn to reinforce them.

Cluster state representation. Data-parallel jobs are usually directed acyclic graphs (DAGs) in which each node represents an operation split into parallel tasks, and each edge represents data dependencies and shuffles between operations. DAGs must execute in topological order, *i.e.*, subsequent nodes cannot start until all parent nodes are finished. A cluster runs multiple DAGs of different jobs concurrently, sharing executor slots between them.[1] The scheduler decides on how to divide executor slots between DAGs, but also what topological order to choose, and how to split executors within each DAG. Figure 2 illustrates how the DAGs of Spark jobs that implement TPC-H queries vary in shape, task duration, and number of tasks per node (*i.e.*, the maximum degree of parallelism).

We developed a new graph embedding technique to transform job DAGs into a continuous vector-space, which enables learning scheduling policies for such jobs with neural networks. The idea is to embed the DAG information in per-node vectors [1], which the neural networks use to compute a probability of scheduling each node. The embedding consists of two main steps: (*i*) aggregating local information at each node from its children (Fig. 3a), and (*ii*) summarizing global information across all nodes (Fig. 3b). Additionally, Clotho appends external, non-DAG information (*e.g.*, executor status) to the vector representation that it feeds to the policy network.

DAG-level message passing. Each DAG node has associated information such as the number of unfinished tasks, number of running

[1] As per Spark terminology, "executors" are parallel workers that process tasks from DAG nodes in a particular job; Clotho currently assumes that a cluster has a fixed number of statically-sized executor slots that it multiplexes between jobs.

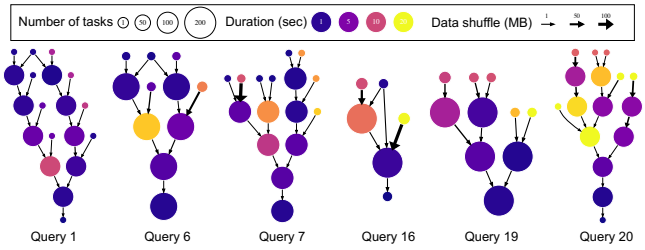
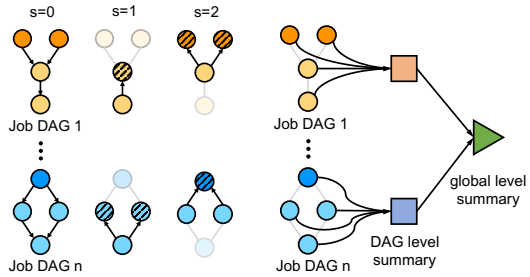


Figure 2: Data-parallel jobs have complex data-flow graphs like the ones shown (TPC-H queries in Spark), with each node having a distinct task duration distribution, number of tasks, and input/output size.



(a) DAG-level message passing. **(b)** Global summarization.

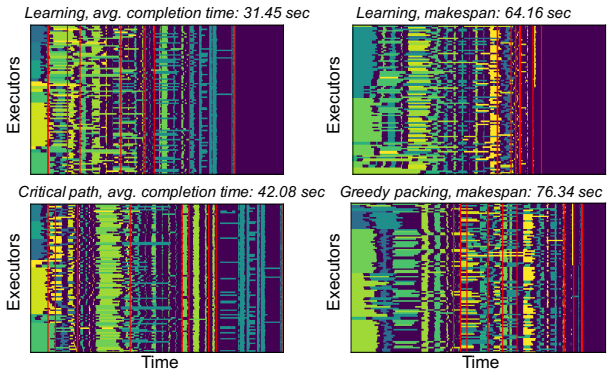
Figure 3: Clotho's graph embedding combines information from each DAG and across DAGs into a tensor with one vector per node, DAG, and globally.

tasks, average task runtime (if known), etc. Our embedding captures, for each node, the information for that node and all its descendants. We compute this embedding by propagating information through the DAG using a sequence of parameterized *message passing* steps, from the leaves to the roots. Initially, each node i has a vector $\mathbf{x}_i^0$ with that node's information. At step s of message passing, nodes aggregate information ("messages") from their children by performing

$$\mathbf{x}_i^{s+1} = g\left[\sum_{j\in\xi(i)} f(\mathbf{x}_j^s)\right] + \mathbf{x}_i^s, \qquad (1)$$

where $f(\cdot)$ and $g(\cdot)$ are non-linear transformations over vector inputs expressed using neural networks, and $\xi(\cdot)$ denotes the set of children for a node. This rule is expressive enough to capture a wide variety of aggregation functions. For example, if f and g are identity functions, they sum the childrens' vectors; if $f \sim \log(\cdot/n)$, $g \sim \exp(n\times\cdot)$, they take the maximum of the child vectors (for large n). As the domain and range of $f(\cdot)$ and $g(\cdot)$ do *not* depend on the DAGs themselves, the same transformations can be applied to DAGs of any scale.

Global summarization. Besides information obtained with local message passing, we also aggregate information across all nodes in each DAG, and across all DAGs. This information is useful for learning policies that require global information, such as comparing the total work in different DAGs. We compute these *summaries* using the same message passing mechanism via two new node types: DAG-level summary nodes (squares in Fig. 3b) and a global summary node (the triangle in Fig. 3b). DAG-level summary nodes are a parent to all the nodes in their DAG, and the global summary node is a parent to all DAG-level summary nodes. Each level of summarization has its own set of parametrized non-linear functions $f(\cdot)$ and $g(\cdot)$.



(a) average completion time **(b)** makespan

Figure 4: Clotho outperforms to two heuristics optimized for average completion time (critical path) and makespan (greedy packing). The x-axis covers 80 seconds in all figures, and earlier job completion (red lines) is better.

3 Preliminary Results

Our early results are promising. In simulations with random mixes of TPC-H queries, Clotho outperforms both Spark's default scheduling policy and purpose-built heuristics for objectives like minimizing average job completion time or makespan (the time to complete the last job). In the examples shown in Figure 4, we sampled 10 DAGs from the TPC-H queries and schedule them over 100 executor slots. Different colors represent different jobs, with vertical red lines indicating job completions. Dark purple color indicates idle periods of executors. Clotho learns entirely different policies for the two objectives: for makespan, it packs jobs tightly to keep executors busy and finishes all jobs in the final ten seconds; for average completion time, it correctly prioritizes shorter jobs for earlier completion.

Generalization. Clotho appears to learn policies that generalize across varying input sizes to jobs (and consequent changes to task durations) and varying cluster size (*i.e.*, executor slot count). In our experiments, Clotho's performance degrades only by 5–10% if one of these factors changes drastically (*e.g.*, from a 5 GB input size to a 100 GB input size, or 50 to 100 executor slots).

Integration with Spark. We have integrated Clotho as a pluggable scheduler into Spark. Whenever the set of runnable "stages" (DAG nodes) in a Spark job changes, it invokes the Clotho service, which may decide to move executors between jobs, or to change the distribution of stages' tasks to executors. Initial experiments indicate that real Spark performance matches the Clotho simulations.

Next steps. We are investigating whether training Clotho on offline logs in simulation yields models that work well on real clusters, and whether we can continue training safely in a running cluster after deployment. Clotho will also need the ability to classify new, unseen cluster behavior at runtime to make better decisions; and Clotho will need to understand mixed workloads with long-running jobs that have different objectives than batch jobs. We also hope to look into whether other systems — *e.g.*, query optimizers, compilers, or database management systems [5] — can use our embedding techniques to learn workload-specific policies.

References

[1] Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. Learning Combinatorial Optimization Algorithms over Graphs. *CoRR* abs/1704.01665

(2017). http://arxiv.org/abs/1704.01665

[2] Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. 2016. Graphene: Packing and dependency-aware scheduling for data-parallel clusters. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 81–97. https://www.usenix.org/system/files/conference/osdi16/osdi16-grandl-graphene.pdf

[3] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource Management with Deep Reinforcement Learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets)*. 50–56. https://doi.org/10.1145/3005745.3005750

[4] Azalia Mirhoseini, Hieu Pham, Quoc V Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. 2017. Device Placement Optimization with Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*. 2430–2439. https://arxiv.org/pdf/1706.04972.pdf

[5] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. 2017. Self-driving Database Management Systems. In *Proceedings of the 8th Biennial Conference on Innovative Data Systems Research (CIDR)*. http://cidrdb.org/cidr2017/papers/p42-pavlo-cidr17.pdf

[6] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. 2010. Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In *Proceedings of the 5th European Conference on Computer Systems (EuroSys)*. 265–278. https://doi.org/10.1145/1755913.1755940