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, general-ized heuristics and on extensive manual tuning. This can be costly: they may run computations inefficiently or unnecessarily leave re-sources 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.

Clotho is a first step towards using neural networks to learn cluster scheduling policies from scratch, focusing particularly on batch-oriented par-allel 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 re-presenting 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 re-source 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 admini-trator (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 di-rected 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. 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.
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

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

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 children’s vectors; if $f = \log(\cdot/n)$, $g = \exp(n \times \cdot)$, 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 $g(\cdot)$.


