Abstract

Stragglers—computations that exhibit extreme tail latencies—present a major challenge to delivering predictable performance in datacenters. Accurately predicting stragglers would enable efficient, proactive intervention. While a number of approaches use machine learning to predict computer system performance, they routinely rely on carefully curated training sets. Assembling the right training set requires prior knowledge, i.e., sufficient examples of all possible behaviors to be identified, which is challenging when new workloads could be unique and unlike any workload from training. To make accurate predictions with neither prior knowledge nor carefully curated datasets this paper presents Agatha, a straggler prediction framework that augments existing machine learning approaches with causal analysis. Agatha weights predictions using a statistical method—called propensity scoring—to account for a lack of stragglers in training data. Additionally, Agatha exploits model-agnostic interpretation to gain insights into the reasons for straggling behavior. We evaluate Agatha on datacenter traces from Google and Alibaba and find that, compared to prior work that is heavily reliant on the training set, Agatha (1) produces much more accurate predictions, (2) does so much earlier in job execution, (3) provides interpretable results, and (4) reduces completion time. Agatha achieves these improved predictions even with no examples of stragglers in its training set, making it better suited for practical deployments.

1 Introduction

Reducing completion time for latency-critical applications is a fundamental problem in datacenter-scale computing [10, 1, 59, 78, 44, 47, 48, 96, 16, 70, 9]. A major impediment is the presence of stragglers, exceptionally slow subcomputations within some larger computation. Several studies show that, while rare, stragglers degrade overall performance by as much as 30 to 50% [6, 79, 102].

Most recent straggler mitigation proposals use predictive modeling, whose main idea is to monitor executing computations and predict straggling before it reveals itself with long run times [7, 80, 95, 46]. Such approaches critically depend on prediction accuracy, while promising greater resource utilization: if stragglers is accurately predicted, extra resources will only be used for the rare stragglers.

As a straightforward application of machine learning (ML) for systems, predictive modeling has been extensively studied for predicting computer systems performance (see recent surveys for a broad overview [75, 100]). The vast majority of these are based on finding correlations between some set of input features (like CPU utilization) and computation latency. Unfortunately, these learning techniques have limitations that are surprisingly subtle, and yet particularly problematic in the context of straggler mitigation:

- **Prior knowledge required.** Most learning methods make predictions based on prior knowledge; i.e., observing samples from all classes at training before prediction. However, when predicting stragglers on lively monitored data, stragglers are not revealed early because they finish last. That said, insufficient prior knowledge exists in the training set because it includes only one class of non-stragglers, which renders most learning methods invalid for straggler prediction on live data.

- **Training set imbalance.** Achieving accurate predictions requires the training set to be balanced; i.e., reflect all classes with roughly equal representation. However, curating an appropriately balanced training set is hard when predicting rare events like stragglers. The state-of-the-art to address imbalance is oversampling by counting the rare examples more than once, which has been used in both ML in general [18] and in ML for systems when working with rare events like stragglers [95]. Unfortunately, this technique will fail if a new workload seen during deployment is unlike any workload from the training set. Also, oversampling makes it likely that the trained model will overfit to the oversampled data [55].

- **Poor interpretability.** Linear models are interpretable and have been used to understand straggling behavior [95]. However, most learning models applied to systems are black-box and thus are difficult to interpret despite high prediction accuracy [101]. Lack of interpretability is a problem, because without an indication of why the task is straggling it is difficult to intervene effectively.
To address these limitations, we present Agatha (Figure 1), a straggler prediction framework with causal analysis [51]. The key insight is that rather than requiring prior knowledge or a carefully curated balanced training set, Agatha predicts stragglers early and accurately by augmenting the learner with a causal tool. Specifically, Agatha starts with a correlation-based learner. Then Agatha weights this learner’s predictions by their propensity score (PS), a statistical method from causal inference for dealing with imbalanced data [83]. Agatha uses PS to quantify how different a particular sub-computation’s features are from those that are known not to straggle. This weighting scheme accounts for training sensitivity by preserving latency predictions for sub-computations that are similar to known non-stragglers and increasing predicted latency for those that are very different. When a straggler is predicted, Agatha applies permutation feature importance (PFI) [33] to return a list of features ranked by how much each contributes to latency. Agatha’s results from causal predictions and model interpretations can be used within a variety of schedulers including those that re-launch the predicted stragglers on other machines to reduce job completion time.

We implement Agatha and evaluate it on two production traces from Google [79] and Alibaba [4]. We parse these traces into time series data and then construct a simulator that sends data from each job to Agatha as if it was live. Thus, for all evaluations, Agatha works on live data and predicts which sub-computations will straggle without first seeing any stragglers during training. This setup differentiates Agatha from all prior work that requires appropriate representation of stragglers in training data [95]. Compared to existing work that requires prior knowledge and uses oversampling to balance training sets:

- Agatha’s causal model improves the true positive rate by 59% and reduces the false positive rate by 7%.
- Agatha identifies stragglers 1.46× earlier. In fact, unlike prior work, Agatha produces accurate predictions before it has any positive examples of stragglers.
- Agatha can be integrated into a simple scheduler to reduce average job completion time by 8–11%.
- Agatha provides reliable interpretations of stragglers that are consistent with prior human expert analysis [102, 42].

In summary, this work makes the following contributions:

- Proposing permutation feature importance to interpret and gain insights into the straggling behavior on live data.
- Implementing Agatha in a real system and releasing its testing infrastructure.

### 2 Related Work and Motivation

We discuss prior work on mitigating stragglers and predicting system performance. We then illustrate how these approaches struggle if their training sets are not carefully constructed.

#### 2.1 Mitigating Stragglers

There is a large body of prior work on understanding and mitigating stragglers [23, 3]. This is effectively a scheduling problem: allocating resources to computations to reduce the longest running tasks’ latency [84]. Mitigation techniques can thus be categorized by whether the scheduler is performance-oblivious or performance-aware.

Performance-oblivious schedulers do not predict which computations will straggle but simply schedule such that stragglers will naturally be reduced. For example, speculative execution launches redundant computations and uses the result of the first to complete [24, 98, 5]. Another example removes slow machines as schedulable resources without attempting to determine which computations might be suited for them [22, 6]. Performance-oblivious approaches are effective; e.g., if every computation is replicated, the worst case latency is very likely to be reduced. The downside is that they waste resources on redundant computation.

Performance-aware schedulers predict stragglers and only allocate additional resources to them. For example, MittOS allows IO requests to be submitted with a target latency [45] and the scheduler predicts if the latency can be satisfied before serving it or sending it to a less-loaded replica. Similar performance-aware schedulers predict stragglers before allocating resources [7, 80, 95]. These approaches promise better resource utilization: with perfect predictions, extra resources only go to slow computations. The downside is that producing accurate predictions is challenging.

#### 2.2 ML for Performance Prediction

ML has influenced much recent research including systems support for ML; e.g., [2, 53, 89, 97, 60, 58, 74, 87, 61, 34, 97, 36, 37, 43, 63, 86, 64, 81, 27]. We, however, are interested in ML for systems, especially scheduling and resource management. Many examples exist, including

---

1 Link to repository removed for double blind review.
learning to predict performance from microarchitectural features [12, 31, 52, 54, 57, 66, 76, 90, 11, 38, 69], application features [15, 25, 26, 67, 65, 29, 103, 50], and combinations of the two [71, 56, 99].

Two issues, however, make it difficult to apply these ML techniques to straggler prediction: (1) training set sensitivity and (2) model interpretability. First, the accuracy of the learning models cited above is highly sensitive to the data set used to train the model. Ideally, the training set data should come from controlled, randomized trials [39] and have roughly equal representation of both stragglers and non-stragglers [94]. Furthermore, all possible causes of straggling should be represented. Meeting these requirements is difficult because stragglers exhibit tail behavior and are thus rare by definition [23, 3]. Furthermore, training set curation is a challenging process to debug as it is often unclear when poor prediction is a result of a poor learning algorithm, a poor training set, or both.

The second challenge is that most learners used in the above projects are difficult to interpret as they rely on finding nonlinear relationships between input features and the predicted outcomes. This means that even if stragglers are correctly predicted, the underlying causes are not clear (the learned model is essentially a black-box [41]) and so it is not obvious how to intervene. For example, a scheduler should intervene differently if straggling is due to the underlying hardware rather than some property of the workload [30]. Wrangler [95] is a performance-aware scheduler that tries to overcome the problems of training set sensitivity and interpretability. To ensure proper representation of stragglers in the training set, Wrangler oversamples them—counting each more than once. While oversampling is commonly used when dealing with imbalanced training sets [32], it tends to result in overfitting the training data, which hurts generalization to new tasks that were not seen at training time [18]. To address interpretability, Wrangler uses only linear models so that the linear weights associated with any feature can be interpreted as that feature’s importance. Linear models, however, usually come with low predictive power and cannot capture interactions between features [49]; e.g., a task that is memory-limited in some circumstances and compute-limited in others [25, 26, 65].

2.3 Learning From Imbalanced Data

Agatha addresses the same challenges as Wrangler—training set sensitivity and black-box model interpretation—but it takes a fundamentally different approach. Rather than carefully curate the training set through oversampling or any other method, Agatha is designed from the beginning to produce accurate predictions from imperfect training data. This approach makes Agatha robust to potential errors in training set construction while enabling it to generalize to workloads for which it has no prior knowledge.

To do so, we start from existing techniques and then reweight their predictions to account for training set imbalance. This design assumes that initial predictions will be biased due to imbalanced training data. We construct a weighting function that organically corrects this bias using features of the input data. Specifically, we use propensity scores [83] from causal analysis, which represent the probability that a subject is in a particular group given its features.

Causal analysis has been used to understand computing systems behavior by carefully randomizing experiments [21, 93] and to find causes of performance anomalies after execution [102]. Causal analysis has also been used to understand ML models and guard them against pollution with malicious data [14]. However, we are not aware of any other work that uses causal analysis to predict future stragglers on live computations. In fact, we believe our use of propensity scores for prediction is novel in any field, not just computing.

2.4 Motivational Example of Training Bias

We demonstrate how existing learners’ straggler predictions are influenced by training set sensitivity. We study a job from a production trace and construct two models: (1) trained with a representative number of stragglers and (2) trained by oversampling stragglers to balance the straggling and non-straggling data (based on prior work [95]). All models predict latency as a function of features (like CPU utilization). However, they either (1) falsely predict all stragglers to be non-stragglers (due to too few stragglers at training) or (2) have high false positive rate, predicting many non-stragglers to straggle (due to overfitting to oversampled stragglers).

We consider the public Google trace data [79], which has millions of jobs, each with multiple tasks. We conduct analysis on a specific job with ID 6343946350 consisting of 268 tasks and use these tasks to construct a data set whose features are task metrics regarding machine capacity, scheduling, resource request and usage, and microarchitecture (Section 4 details the full experimental setup). We define tasks with completion time beyond 90th percentile as stragglers.

Figure 2: CDF of normalized latency on a real computation: (a) true data; (b) predictions from learning model with representative stragglers (5% of training data); (c) predictions from learning model with oversampled stragglers.
Our goal is to use the features to predict whether a task will straggle. We use gradient boosting [35] as our learner.

Figure 2 shows the results. The blue curve shows the true CDF for tasks in this job. A red vertical line shows the p90 cutoff. The orange curve shows the predicted CDF from the model trained with representative stragglers, which predicts all tasks not to straggle (the CDF hits 1 to the left of the true p90 cutoff). The green curve shows the predictions from oversampling stragglers, which predicts a large number (26.7%) of non-stragglers to straggle (the CDF is shifted to the right of the p90 cutoff). This example shows that due to training sensitivity, learning models are either too conservative—due to lack of straggling examples—or too aggressive due to overfitting to stragglers. In summary, carefully curating the training set is difficult because having too few stragglers causes the model to fail to recognize new stragglers (false negatives), while having too many stragglers in the training set causes many non-stragglers to be labeled as stragglers (false positives).

The next section presents an alternative and the paper’s theme: rather than reweighting the training data, reweight the prediction based on features observed at run time.

3 Agatha Design

Figure 1 illustrates Agatha consisting of two complementary modules. The first, Causal Prediction (CP), monitors runtime behavior and predicts future stragglers. The second, Model Interpretation (MI), diagnoses the predictions. For simplicity, we assume that we have a job made of tasks and we want to predict which tasks will straggle. However, different frameworks use different terminology; Agatha handles any computation (job) consisting of sub-computations (tasks). Agatha continually monitors a job, measuring data for each task at fixed time intervals. At each time step, Agatha runs CP to predict which tasks will straggle; i.e., whether or not they will exceed a user-specified latency threshold. For each predicted straggler, Agatha runs MI to interpret the prediction. Agatha runs live so both CP and MI react immediately as they receive new data. Agatha’s prediction and interpretation results can be sent to any schedulers that intervene with the predicted stragglers on other machines to reduce the job completion time. For example, a simple approach would be relaunching tasks that are predicted to straggle. A more complicated approach might use the interpretation to dynamically reallocate resources and speed up the predicted stragglers based on their predicted resource needs.

3.1 Intuition and Challenges

Straggler prediction estimates latency as a function of task features; i.e., readily available metrics like CPU utilization, IO behavior, etc.. Then, tasks predicted to exceed a user-specified latency threshold are stragglers. The difficulty is that imbalanced training data biases predictions.

Agatha overcomes this issue by reweighting predictions based on a function of task features. The intuition is that a straggler behaves differently from a typical task and this behavior can be captured by available features. If we think of a task’s features as a point in a high-dimensional (feature) space, the non-straggler points will be relatively close together and the stragglers will be far away. Using this insight, we reweight latency predictions by an amount proportional to the “distance” between the points represented by the features. This weighting will be small for non-stragglers and large for stragglers separating the predicted latencies for the two groups. We note four challenges to applying this insight:

1) Finding a weighting function. Weighting by propensity scores (PS) has been used in other sciences to estimate causal effects once outcomes are known [8]. We propose a novel approach: applying propensity on live data to predict future stragglers, leading to the next challenge.

2) Labeling live data. PS can be estimated by a supervised learning method called Logistic Regression [17], but doing so requires labeled training data. Since Agatha acts on live data, there are no labels for tasks that have not completed. Agatha addresses this challenge by first waiting for a small number of tasks to complete. These tasks are not stragglers because they finish early and their latencies are revealed, and thus Agatha labels them all non-stragglers. From that point on, Agatha labels the rest tasks stragglers and computes their PS. While most remaining tasks will not straggle, this approach is feasible because the number of stragglers is so small that the PS will be low unless the new task behaves fundamentally differently than the non-stragglers. Even when mislabeled, the PS will be low because a mislabeled non-straggler’s features will be “close” to those of other non-stragglers, while a straggler’s features will be “far”.

3) Correcting the weighting function. As it runs, Agatha accumulates mislabeled examples and thus the PS (computed through logistic regression) will lose the ability to distinguish stragglers. Agatha addresses this challenge by updating its models online once the true latency is known; i.e., when a task finishes, Agatha relabels it with its known outcome (straggler or not) and updates both of the correlation-based learner and logistic regression models. Thus, Agatha actually improves its models as it collects more data.

4) Interpreting models. To intervene properly, we want to determine which features have the largest effect on a predicted straggler. As Agatha uses nonlinear models, it applies Permutation Feature Importance (PFI) [33] for interpretation. This technique would typically be applied after the outcomes are known and would evaluate feature importance by measuring the change in prediction error as features are permuted. Since Agatha operates on live data we make a subtle change of modifying PFI to measure importance as
the change in predicted outcome instead of error.

3.2 Causal Prediction

Causal prediction (CP) assumes no prior knowledge or pre-training. Therefore, deployment has both a training phase—where labels are collected and the models are built—and a prediction phase, where it identifies stragglers on live data.

3.2.1 Training on Live Data

Agatha treats straggler prediction as a regression problem by estimating latency first and then comparing it to a user-defined target latency. Latency predictions above the target are stragglers. We choose regression over classification because we need to reweight the latency predictions, while the outputs from classification are binary—0 for non-stragglers and 1 for stragglers. Thus reweighting classification output would result in one class always being 0. Through regression, Agatha can reweight the predictions effectively.

Agatha works with any correlation-based learner including linear models (although that is not recommended, due to low accuracy), or complicated, black-box models (e.g., Gradient Boosting [35] or neural networks [40]). We believe Agatha could even work with future black-box models as they become available. Our empirical analysis finds that Gradient Boosting produces the best results (see Section 5.6) as they become available. Our empirical analysis finds that Gradient Boosting produces the best results (see Section 5.6) and we thus use it as the default learner. Our evaluation also shows the generality of our approach by applying our framework with five different learners with online updates.

At the beginning of deployment, Agatha waits for the first small number of non-straggler tasks to complete. These tasks are used to train the gradient boosting model in which each training sample has task features and latency. The choice of task features depends on the specific system on which Agatha is deployed. Ideally, Agatha would use metrics related to scheduling, resource usage, and microarchitecture. It is important that the recorded features cover a wide range of possible causes for long-latency behavior [68]. For example, if all input features correspond to CPU usage, no framework could identify straggling tasks caused by memory behavior. Agatha also requires that the metrics can be collected live so that it can predict stragglers in real time.

To assemble its training data on the live job, Agatha waits for 4% of tasks to finish (this choice is justified empirically in Section 5.5). Those that finish early are non-stragglers, and Agatha trains the gradient boosting model on these tasks. Then at each time checkpoint, it uses the trained model to predict whether the unfinished tasks will exceed the user-specified latency threshold.

3.2.2 Straggler Prediction with Propensity Scores

The CP module records the features for all active tasks in a job and feeds these features into the latency predictor. Agatha identifies tasks that will straggle using the adjusted latency prediction, which reweights the latency prediction produced by the gradient boosting model. The key (Challenge (1) from above) is to mitigate prediction bias since the model is trained on data that has no stragglers. Agatha thus turns differences in task features into a weight that separates the predicted latency of stragglers from non-stragglers.

Consider the predicted latency from gradient boosting is \( \hat{y} \). We need a weighting function on features such that if the features are close to those of known non-stragglers, the straggler prediction stays almost unchanged, but if the features are vastly different from known non-stragglers then the latency prediction will be scaled proportionally to the difference. Specifically, the goal is to find \( ps(x) \) in Equation 1:

\[
\hat{y}_{adj} = \frac{\hat{y}}{1 - ps(x)},
\]

where \( ps(x) \) is a value between 0 and 1 such that it is larger when the features are less similar to known non-stragglers. In Agatha, \( ps(x) \) is the probability that a task is a straggler, given features \( x \).

Agatha finds \( ps(x) \) using logistic regression (LR) [17], a supervised learning approach that estimates the probability that a feature vector belongs to a certain binary class. Since it does not have labels for tasks that have not finished, Agatha assumes that any running task will straggle and labels it as such (Challenge (2) from above). This deliberate mislabeling is feasible because the number of stragglers is so small that PS will be low unless the new task behaves fundamentally differently than known non-stragglers tasks. Since logistic loss is convex, Agatha trains LR up to a linear rate of convergence and scales to large training sets using stochastic optimization [13, 62].

After estimating \( ps(x) \), Agatha gets \( \hat{y}_{adj} \) from Equation 1. For stragglers, \( \hat{y}_{adj} \) will be much longer than \( \hat{y} \) since \( 1 - ps(x) \) will be relatively smaller. For non-stragglers, \( \hat{y}_{adj} \) is almost the same as \( \hat{y} \) since \( 1 - ps(x) \) will be close to 1.

As more tasks are evaluated, Agatha builds up a number of mislabeled examples—non-stragglers labeled as stragglers—and it will slowly bias the results towards predicting all tasks as non-stragglers (Challenge (3) from above). Agatha corrects this by updating its models once a task’s true latency is revealed lively. Algorithm 1 summarizes causal prediction.

3.3 Model Interpretation

After predicting a straggler, Agatha immediately triggers MI to interpret models. Recent work shows that causal relations can be extracted via interpreting black-box models [101]. Our goal is to find important features contributing to the prediction results. Although logistic regression is interpretable due to its property of linearity, the correlation-based
learner Agatha relies on (e.g., Gradient Boosting) is black-box model, which makes the combination of the correlation-based model and the weight also black-box.

To address this issue, Agatha applies the Permutation Feature Importance (PFI) [33] to interpret the models from CP and gain insights into the straggling behavior (addressing Challenge (4)). The intuition is that feature permutation breaks the association between a feature and the prediction. A feature is “important” if its prediction changes dramatically after permuting its values because the model relies on this feature for prediction. In contrast, a feature is “unimportant” if its prediction changes little after permutation because the model ignores that feature. Unlike the original PFI that computes the increase in prediction error to measure feature importance, Agatha measures the importance of a feature by calculating the change in prediction after permuting the feature. This approach allows Agatha to operate on live data, where the true prediction error is unknown until the task completes. Algorithm 2 summarizes model interpretation.

### Algorithm 2 Permutation feature importance for MI.

| Require: | h | → Trained correlation-based learner |
| Require: | LR | → LR model |
| Require: | X \text{train} | → Features for completed tasks |
| Require: | y \text{train} | → Latency for completed tasks |
| Require: | \delta t | → Latency threshold |
| Require: | y \text{str} | → Predicted latency for predicted straggler |
| Require: | y \text{perm} | → Predicted latency for predicted straggler |

for each feature \( j = 1, \ldots, p \) do

- Get \( X_{\text{train}} \) by permuting feature \( j \) in data matrix \( X_{\text{train}} \).
- Retrain \( h \) and \( LR \) on \( X_{\text{train}} \) and \( y_{\text{train}} \) to get new prediction \( \hat{y}_{\text{perm}} \).
- Record feature importance by computing \( \Delta_j = |\hat{y}_{\text{perm}} - y_{\text{train}}| \).

Sort feature importance \( \Delta_j, j = 1, \ldots, p \) in descending order.

Output: Sorted features in descending order.

### Algorithm 3 Scheduling with unlimited machines available.

| Require: | \( T \) | → Total time checkpoints |
| Require: | \( N \) | → Set of active tasks |

for each time checkpoint \( t = 1, \ldots, T \) do

for each active task \( n \in N \) do

- if task \( n \) is predicted to be a straggler then
  - Terminate \( n \) and relaunch it on a new machine.
  - Update set of active tasks \( N \leftarrow N \setminus n \).
- else
  - Go to next task.

end for

end for

3.4 Scheduling

After Agatha predicts a straggler and interprets the possible causes of its straggling behavior, these results can be used for any schedulers that relaunch the predicted stragglers on other machines or take other intervention actions that can mitigate the straggling behavior. To demonstrate how Agatha can be used in datacenters to reduce job completion time, we design schedulers to reassign tasks once a straggler is predicted. We consider two different situations: unlimited machines available and limited machines available. The unlimited scenario refers to the case where there are many more machines than tasks in a job.

We note that these approaches are included to demonstrate how Agatha’s predictions can be easily incorporated into existing schedulers and are not intended to present novel scheduling approaches in and of themselves. For example, a common approach to avoid tail latency is to set a timeout threshold and then relaunch any tasks that exceed that threshold [23, 92]. The schedulers below show how to improve upon this approach by replacing the timeout with Agatha’s straggler predictions.

#### 3.4.1 Unlimited Machines

Since Agatha acts on live data, we monitor runtime behavior at each time checkpoint. At each time checkpoint, Agatha evaluates each active task and predicts if it will straggle. When unlimited machines are available, a task that is predicted to be a straggler can be terminated and reassigned to a new machine immediately. Algorithm 3 summarizes the scheduling procedure with unlimited machines available.

#### 3.4.2 Limited Machines

In practice, it is not always the case that unlimited machines are available. In that event, the scheduler needs to check if there are new machines that just finished running tasks at each checkpoint. Should that be the case, these machines will also be considered for future assignment. Then, Agatha will evaluate each active task and predicts if it will straggle. If that is the case and there are machines available, this task will be terminated and relaunched on a new machine immediately. Otherwise, the scheduler will move on to the next active task and wait for new machines at next time checkpoint.
Algorithm 4: Scheduling with limited machines available.

Require: \( T \) → Total time checkpoints
Require: \( N \) → Set of active tasks
Require: \( M \) → Set of available machines

for each time checkpoint \( t = 1, \ldots, T \) do
    if new machines \( S \) available after finishing running tasks then
        Update set of available machines \( M \leftarrow M \cup S \).
    end
    for each active task \( n \in N \) do
        if task \( n \) is predicted to be a straggler then
            if machines are available \( M \neq \emptyset \) then
                Terminate \( n \) and relaunch it on a new machine \( m \).
                Update set of active tasks \( N \leftarrow N \setminus n \).
                Update set of available machines \( M \leftarrow M \setminus m \).
            else
                Go to next task.
        end
    end
end

Algorithm 4 summarizes the scheduling procedure with limited machines available.

3.5 Discussion and Limitations

Agatha uses a user-defined latency threshold to determine whether a task will straggle by comparing it to the adjusted predicted latency. This latency threshold can either be selected manually by users or automatically by other techniques such as those in MittOS [45] and LinnOS [46]. In our evaluation, the user-specified latency target is set to be equivalent to \( p90 \) per job (see Section 4.1) and we also demonstrate that Agatha is robust to different thresholds in Figure 7; i.e., Agatha’s benefits are independent of how the definition of straggler is set.

Although PS is defined as the probability that a task will straggle given its features, it cannot be directly used to classify stragglers (see Section 5). The reason is that due to pre-labeling, the computed PS for all unfinished tasks are greater than 0.5, which is useless in classification if a traditional threshold 0.5 is taken. Even if we use the computed PS for classification, an appropriate threshold is unknown because it is job-dependent. Therefore, PS only functions as reweighting in Agatha for final latency prediction.

To interpret models in real-time, Agatha runs PFI with extra cost, which is \( p \) times of model training (\( p \) is number of features). An alternative is to wait until a job completes and then interpret it with all tasks available. This strategy will require much less overhead but does not produce interpretations in real-time. This is the trade-off we consider for model interpretability, and we are willing to pay extra computation cost so that we can gain insights immediately when a straggler is identified. We evaluate overhead in Section 5.8.

Transfer learning is also a technique to deal with a lack of training data [72], but it is impractical for straggler prediction on live data. Transfer learning between tasks would be useful if tasks are similar to each other. However, it is hard to transfer if a new, unique workload is coming. In contrast, Agatha would still work with that unique workload, however, so it is more robust than the transfer learning approach.

4 Experimental Setup

4.1 Evaluation Methodology

We evaluate Agatha’s ability to predict stragglers as computations run. We construct a simulator by parsing publicly available data traces and converting them into a time-series format; i.e., a series of the statistics available for each timestamp. The simulator replicates real execution by sending Agatha the statistics that would be available at a given time.

We use two traces from different providers to demonstrate generality. Each has different computational structures and records different metrics, demonstrating that Agatha is not tied to a particular structure or set of metrics. Each trace has data for a number of computations that are divided into sub-computations. For each computation, we set a latency threshold such that any sub-computations higher than 90th percentile latency are considered stragglers (we evaluate sensitivity to this threshold below). For all evaluations, Agatha works on live data and makes predictions about which sub-computations will straggle without first seeing any stragglers during training. This setup differentiates Agatha from all prior work that requires appropriate representation of stragglers in training data. The simulator and Agatha are run on a dual socket server with two 32-core Intel Xeon Gold 6242 processors, 192GB RAM, and 2.80GHz clock speed.

4.2 Converting Traces to Time Series

Our evaluation uses two production cluster traces:

Google Trace. The Google trace includes 29 days of data from 12.5K machines [79]. The computations consist of a number of jobs, each of which has numerous tasks, from 100 to 9999. We filter to only include production jobs with 100 or more tasks, which reduces the 650K jobs and 25M tasks to 8425 jobs and 1.1M tasks. As detailed in Table 1, there are 15 metrics for each task categorized into (1) resource usage, (2) microarchitectural, and (3) scheduling. The trace contains time stamped data, which we arrange in time-order to form the time-series data set for our simulation framework.

Alibaba Trace. The Alibaba trace includes two data sets [4, 85]: (1) a 2017 trace consisting of 1.3K machines over 12 hours; (2) a 2018 trace consisting of 4K machines over 8 days. Both have the same format. We use the batch processing data set from these traces, which consists of jobs with various interdependent tasks. Each task is run as a set of instances, consisting of identical processing of different input data. Thus, for Alibaba, we treat tasks as the computation and instances as the sub-computation. Table 2 shows the metrics available for each instance. We arrange the tasks as a time-series of instances based on the time order, removing non-terminated instances. We filter the tasks to those with at least 100 instances reducing the data set size to 1M tasks.
Table 1: Task metrics in the Google Trace.

<table>
<thead>
<tr>
<th>Category</th>
<th>Task Metric</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Usage</td>
<td>MCU</td>
<td>Mean CPU usage</td>
</tr>
<tr>
<td></td>
<td>MAXCPU</td>
<td>Maximum CPU usage</td>
</tr>
<tr>
<td></td>
<td>SCPU</td>
<td>Sampled CPU usage</td>
</tr>
<tr>
<td></td>
<td>CMU</td>
<td>Canonical memory usage</td>
</tr>
<tr>
<td></td>
<td>AMU</td>
<td>Assigned memory usage</td>
</tr>
<tr>
<td></td>
<td>MAXMU</td>
<td>Maximum memory usage</td>
</tr>
<tr>
<td></td>
<td>UPC</td>
<td>Unmapped page cache memory usage</td>
</tr>
<tr>
<td></td>
<td>TPC</td>
<td>Total page cache memory usage</td>
</tr>
<tr>
<td></td>
<td>MIO</td>
<td>Mean disk I/O time</td>
</tr>
<tr>
<td></td>
<td>MAXIO</td>
<td>Maximum disk I/O time</td>
</tr>
<tr>
<td></td>
<td>MDK</td>
<td>Mean local disk space used</td>
</tr>
<tr>
<td>CPU-arch</td>
<td>CPI</td>
<td>Cycles per instruction</td>
</tr>
<tr>
<td></td>
<td>MAI</td>
<td>Memory accesses per instruction</td>
</tr>
<tr>
<td>SKD</td>
<td>EV</td>
<td>Number times task is evicted</td>
</tr>
<tr>
<td>patch</td>
<td>FL</td>
<td>Number times task fails</td>
</tr>
</tbody>
</table>

Table 2: Instance metrics in the Alibaba Trace.

<table>
<thead>
<tr>
<th>Inst. Metric</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACPU</td>
<td>Average CPU numbers of actual instance running</td>
</tr>
<tr>
<td>MCPU</td>
<td>Maximum CPU numbers of actual instance running</td>
</tr>
<tr>
<td>AMEM</td>
<td>Average normalized memory of actual instance running</td>
</tr>
<tr>
<td>MMEM</td>
<td>Maximum normalized memory of actual instance running</td>
</tr>
</tbody>
</table>

4.3 Points of Comparison

We compare Agatha to the following:
1. **Correlation (Corr)**: train a correlation-based model on observed tasks and predict stragglers on unseen tasks.
2. **Correlation-Oracle (Corr-O)**: train a correlation-based model with prior knowledge of stragglers.
3. **LogReg (Log)**: train a logistic regression model on observed tasks and predict stragglers on unseen tasks.
4. **Wrangler**: a complete solution for straggler prediction using linear support vector machines and oversampling [95].
5. **Causal-Oracle (Causal-O)**: apply proposed causal prediction using prior knowledge of stragglers.

We use Gradient Boosting (GB) as the default correlation learner due to its high accuracy. We update the GB model online as tasks resolve. Unlike the other methods, Wrangler is an offline method and assumes it has examples of straggling at training (which it then oversamples), while Agatha assumes no prior knowledge of straggling. In other words, for all results Agatha predicts straggling behavior with no positive examples of straggling for training. To replicate the setting in the original Wrangler paper, we randomly sample 2/3 non-straggler and straggler tasks respectively for training, which corresponds to training 2 hours and testing on the next hour as proposed in the Wrangler paper.

To summarize, **Correlation, LogReg and Agatha** are online, while Wrangler is offline. We include LogReg to show that logistic regression by itself is not useful, but only valuable when used as a weighting function within Agatha. The two **Oracle** techniques are not practical, but we include them to see the best correlation and causal learning could possibly achieve if they had prior knowledge of all tasks at training. The oracles are not 100% accurate because not all stragglers can be correctly predicted from the input features.

4.4 Evaluation Metrics

We evaluate prediction accuracy and model interpretation:

**Accuracy.** We report true positive rate (TPR) and false positive rate (FPR) averaged over all computations for straggler prediction. TPR and FPR are defined as

\[
TPR = \frac{TP}{TP + FN}, \quad FPR = \frac{FP}{FP + TN}
\]

where TP is true positive, FN is false negative, FP is false positive, and TN is true negative. Higher TPR and lower FPR values indicate the better model.

**Reduction in completion time.** We evaluate the percentage of reduction in job completion time by scheduling compared to those without any task rescheduling on new machines. Reduction is defined as

\[
\text{Reduction} = \frac{T - T_{sch}}{T},
\]

where \(T\) is the completion time without rescheduling, \(T_{sch}\) is the completion time with rescheduling.

**Model interpretation.** We measure the importance of features and rank their importance in descending order. To compare different models, we compare the relevance of each model’s rankings to that of the **Causal-Oracle**. The relevance can be assessed by Average Precision at \(k\) (AP@k) [82], which indicates how many of the relevant items are concentrated in the highest ranked predictions:

\[
\text{AP@k} = \frac{1}{\text{TG}} \sum_{i=1}^{k} \frac{\text{TP seen}}{i}
\]

where TG refers to the total number of ground truth positives for the **Causal-Oracle** list and TP seen refers to the number of true positives seen till the position \(k\) for the predicted list.

5 Experimental Evaluation

We empirically evaluate the following questions:
1. How accurate are Agatha’s predictions?
2. Does Agatha identify stragglers early?
3. Does Agatha predict extreme stragglers?
4. Does Agatha improve completion time?
5. How sensitive is Agatha to its design decisions?
6. Does Agatha generalize across learners?
7. Does Agatha correctly interpret models?
8. What is Agatha’s overhead?
Table 3: Average prediction results. Higher is better for TPR, p95, p99, and p99+. Lower is better for FPR.

<table>
<thead>
<tr>
<th></th>
<th>Corr</th>
<th>Corr-O</th>
<th>Log</th>
<th>Wrangler</th>
<th>Agatha</th>
<th>Causal-O</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPR</td>
<td>10%</td>
<td>10%</td>
<td>9%</td>
<td>58%</td>
<td>99%</td>
<td>94%</td>
</tr>
<tr>
<td>FPR</td>
<td>3%</td>
<td>1%</td>
<td>8%</td>
<td>32%</td>
<td>30%</td>
<td>16%</td>
</tr>
<tr>
<td>p95</td>
<td>8%</td>
<td>43%</td>
<td>8%</td>
<td>35%</td>
<td>86%</td>
<td>91%</td>
</tr>
<tr>
<td>p99</td>
<td>10%</td>
<td>76%</td>
<td>9%</td>
<td>63%</td>
<td>88%</td>
<td>95%</td>
</tr>
<tr>
<td>p99+</td>
<td>13%</td>
<td>90%</td>
<td>15%</td>
<td>67%</td>
<td>89%</td>
<td>97%</td>
</tr>
</tbody>
</table>

5.1 How accurate are Agatha’s predictions?

Figure 3(a) shows the prediction accuracy results after all tasks finish. If a task is predicted to be a non-straggler, it will be evaluated again at next time checkpoint. If a task is predicted to be a straggler, it will not be evaluated again later. The x-axis shows TPR (higher is better) and FPR (lower is better), while the y-axis shows the aggregated results across all jobs. Correlation achieves both the lowest true positive rate (TPR) and false positive rate (FPR), which is not surprising as it is vastly impacted by training imbalance: without many positive examples of stragglers, Correlation will be conservative and predict most tasks to be non-stragglers. LogReg performs similarly with Correlation since it also suffers from training imbalance. In contrast, Agatha has the highest TPR among non-oracle methods, and lower FPR than Wrangler. Wrangler’s TPR and FPR are consistent with prior results on different data sets [95]. Since they are built with prior knowledge, the oracle methods are better than their online counterparts, and Causal-Oracle is the best (as expected). Table 3(a) shows the average TPR and FPR over two data sets. From this table, Agatha and Causal-Oracle have the best effect on TPR, which validates the effectiveness of causal prediction. The two oracles have improved FPR due to their prior knowledge of the data. Agatha has higher FPR than Correlation because (1) Correlation is trained on data biased towards non-stragglers and (2) we design Agatha to achieve high TPR at the cost of a slight FPR increase. Wrangler, however, has the worst FPR because it overfits the over-sampled straggler examples (see Section 2.2). Overall, these results demonstrate that Agatha’s reweighting predictions by propensity score has a dramatic positive effect on straggler prediction, even when the data shows extreme bias; i.e., no positive examples in the training set.

5.2 Does Agatha identify stragglers early?

Since Agatha works on live data, it should identify stragglers early in execution. To test this, we compute the time when all stragglers are correctly identified, measured from the start of job execution. Identification can occur when a true positive is registered or when a false negative exceeds the latency threshold. For example, if the latency threshold is 0.9, the task takes one second and Agatha predicts it to straggle at 0.2 seconds, we count the time of identification as 0.2. If that task was predicted not to straggle, its time of identification is 0.9, when the latency threshold is exceeded. We compare the time to identify stragglers for Correlation, LogReg, Wrangler, and Agatha.

Figure 4 shows the results, with the learner on the x-axis and the normalized time (100% means all tasks were only identified after exceeding the threshold) to identification on the y-axis. The results show that Agatha is, by far, the fastest approach: identifying stragglers 1.91× faster than Correlation, 1.97× faster than LogReg, and 1.46× faster than Wrangler. These results include Agatha’s overhead, measured in detail below. Agatha achieves this speedup because weighting by propensity scores separates latency predictions for stragglers from non-stragglers early in task execution. These results show that Agatha’s causal approach identifies stragglers much earlier than other approaches, even though Agatha has no examples of stragglers in its training set when it makes predictions. Agatha’s speed makes it possible to intervene with the stragglers much earlier than prior work.

5.3 Does Agatha predict extreme stragglers?

Since extreme-tail tasks are major bottlenecks, we examine Agatha’s prediction performance for tail tasks. Here, we show TPR on three latency percentile intervals respectively: [90, 95]%, [95, 99]%, and [99, 100]%, which are denoted as p95, p99, and p99+, in Figure 3(b) (all tail tasks
are stragglers so FPR is not available here). From this figure, we see that Agatha achieves higher TPR than Correlation, LogReg, and Wrangler in all three intervals, and Causal-Oracle is the best. These trends are quantitatively visible in Table 3(b), where causal prediction methods—Agatha and Causal-Oracle—outperform their correlation-based counterparts. These results show that Agatha’s causal methods are, by far, the most effective at identifying extreme stragglers.

5.4 Does Agatha improve completion time?

We evaluate how Agatha contributes to reducing job completion time by augmenting existing schedulers with improved straggler predictions. The key idea of the schedulers described in Section 3.4 is to relaunch the task on a new machine once the task is predicted to straggle. In our experiments, the new completion time for a rescheduled task is randomly sampled from the existing execution times. We show results in the following two different situations.

5.4.1 Unlimited machines

When unlimited machines are available (i.e., more machines than tasks), a task that is predicted to straggle can always be terminated and relaunched on a new machine immediately (see Algorithm 3). Figure 5 shows the results, with the learner on the x-axis and the reduction in completion time on the y-axis. The results show that Agatha has the greatest reduction: 16% more than Correlation, 17% more than LogReg, and 11% more than Wrangler. Agatha achieves this improvement due to its early and accurate predictions for stragglers. These results show that Agatha can be easily incorporated with the schedulers and effectively reduce the completion time.

Figure 5: Reduction in completion time (higher is better).

5.4.2 Limited machines

When limited machines are available, the scheduler needs to check if a new machine is available for relaunch if a task is predicted to straggle (see Algorithm 4). Since the job sizes range from hundreds to thousands in the traces, we see how reduction will change as a function of the number of machines available. Figure 6 shows the results, where the x-axis shows the number of machines changes from 100 to 900, and the y-axis shows the reduction. As the number of machines increases, the reductions for different approaches increase. Agatha increases fastest because with more machines available, more predicted stragglers can be relaunched on new machines as early as possible. Correlation and LogReg increase much less because they only predict a small amount of true stragglers. Wrangler is in-between mainly due to its high false positive rate meaning that many non-stragglers are relaunched. These results show that Agatha achieves the best tradeoff between the true positive rate and false positive rate, and the most reduction in completion time with various numbers of machines available.

Figure 6: Reduction with different number of machines.

5.5 How sensitive is Agatha?

How sensitive is Agatha to the latency threshold? We explore how Agatha’s accuracy changes as we change the threshold for straggling. Our default latency threshold is 90% and now we vary it from 70% to 95%, and re-evaluate accuracy. Figure 7(a) shows TPR and FPR as a function of latency threshold. With increasing latency threshold, TPR barely changes and FPR decreases around 10%, which indicates improvement of FPR. These results show that Agatha is robust to the definition of stragglers.

Figure 7: Sensitivity to (a) threshold and (b) training set size.

How sensitive is Agatha to the training set size? Agatha’s default training size is 4%. Here we evaluate a range 2% to 20%. Figure 7(b) shows TPR and FPR as a function of the number of tasks observed in the initial training. With more training data available, TPR barely changes and FPR gets slightly better. These results show that Agatha is robust to the changes of training set size.

Do online updates benefit Agatha? Agatha updates its model as true labels are revealed on live data. We explore...
how this affects Agatha’s performance. We compare TPR and FPR of Agatha both with and without online updates as a function of training set size in Figure 8. With updates Agatha yields accurate results even when the training set is small. In contrast, without online updates, Agatha performs poorly; i.e., barely identifying stragglers when training size is small and increasing FPR with more training data. These results verify that online updates have a significant effect on Agatha’s performance; making this a good design choice for addressing Challenge (3) from Section 3.1.

Figure 8: Agatha with and without online updates.

How does Agatha compare to oversampling stragglers? Agatha uses propensity scoring to overcome training imbalance rather than oversampling; i.e., simply counting each straggler multiple times in training as is done by Wrangler [95]. We justify this design choice by comparing Agatha to Correlation, and Correlation with oversampling, i.e., Oversample-1/6 and Oversample-2/3, which randomly sample one sixth and two thirds of non-stragglers and stragglers for training respectively, and then count each straggler multiple times to balance training set. Figure 9 shows that oversampling improves TPR but worsens FPR compared to Correlation. However, Agatha achieves both better TPR and FPR than oversampling. These results justify that propensity scoring is a better approach than oversampling and it helps explain Agatha’s advantage over Wrangler.

Figure 9: Comparison with oversampling techniques.

5.6 Does Agatha generalize across learners?

Agatha’s results thus far are with the best learner (Gradient Boosting). Here, we examine its ability to generalize to different learners with online updates. In other words, we allow each to update its model after every task completes. This approach isolates the advantages of Agatha’s weighting procedure as all models are updated with all available information. For each learner we provide a reference to the technique itself, as well as to a prior use in a systems context. To promote reproducibility, we use the implementations from Scikit-learn for all learning models [73]. The details for each learner are shown as follows and all hyperparameters are chosen via cross validations.

1. Ridge: $\ell_2$ regularized (or ridge) linear regression [49, 66].
2. SVR: support vector regressor with RBF kernel [20, 77].
3. NN: multi-layer perceptron, a neural network model [40, 88]. We use three hidden layers with size 100, 50, and 10.
4. RF: random forest for regression [91, 19], an ensemble method. The number of trees is 100.
5. GB: gradient boosting for regression [35, 28], an ensemble method. The number of boosting stages is 100. This is also the default learner for all other evaluation.

Figure 10 shows how Agatha improves all these learners with online updates. Agatha achieves overwhelmingly better TPR, especially for nonlinear and ensemble learners including SVR, RF, and GB. Agatha NN is not as good as expected likely because it is quite difficult to optimally tune a neural network architecture for different jobs. These results demonstrate Agatha’s generality and they are significant because they show that Agatha’s results are not tied to a particular online learning algorithm, but are a general benefit of augmenting existing learners with causal analysis.

5.7 Does Agatha correctly interpret models?

We validate whether Agatha interprets models accurately with both aggregated interpretation results and case studies. Aggregated Interpretation Results. To evaluate Agatha’s interpretability we apply permutation feature importance to Correlation and both Oracles. We also compare to the features identified by Wrangler, corresponding to the highest magnitude weights in its linear model. We compare the relevance of features from each approach to that of Causal-Oracle—the best predictor—in terms of average precision at $k$ (AP@k). For the Google trace, we group the total 15 features into 6 classes; e.g., “canonical memory usage” and “assigned memory usage” are grouped in the memory category. The Alibaba trace has 4 features and thus 4 classes. Figure 11 reports the relevance of top-3 and top-5 for Google and top-1 and top-3 for Alibaba. It shows that relevance with smaller $k$ is lower, suggesting that the higher ranked features are harder to align. In particular, Correlation-Oracle is the best due to its oracle property. For non-oracle methods, Agatha is the best—slightly better than Wrangler—meaning that Agatha finds the most relevant features. These results demonstrate
that Agatha’s combination of propensity scoring with PFI provide insights into the straggling behavior in real-time.

**Case Studies.** We compare Agatha’s model interpretations to human expert analyses with case studies. Prior work categorized the causes for Google trace into six classes: limited memory, limited processor, I/O, data skew, cache bottleneck, and scheduling [102]. We do not have case studies for Alibaba as it has only four features [42].

Figure 12 shows four case studies where a job in the Google trace has known causes [102]. We compare Wrangler and Agatha to Hound [102], which diagnoses causes of stragglers (although it operates after the job has completed and does not make straggler predictions). We use radar charts to display multi-dimensional causes and their relative strength, where bold labels are causes identified by human experts. The area covered by Agatha and Wrangler heavily overlaps the area covered by Hound, and their strengths are consistent as well. In practice, causes depend on each other, and therefore it is likely for Agatha and Wrangler to cover wider areas than the human experts, indicating more possible causes than human labels. These case studies further demonstrate Agatha’s ability to diagnose causes by discovering important features. Agatha achieves roughly equal causal analysis to Hound and Wrangler, but does so much earlier (see Figure 4) and is thus more useful in practice.

### 5.8 What is Agatha’s overhead?

Table 4 shows the number of tasks per second predicted and interpreted on a single server with both a single core and all available cores on our testbed (see Section 4.1). Since Agatha trains online, its overheads include training time. In Table 4(a), Wrangler is the computationally cheapest due to linearity, while Correlation and Agatha are built on Gradient Boosting—a nonlinear ensemble with better accuracy at the cost of overhead. The extra overhead from Agatha compared to Correlation is due to logistic regression. We find the overhead tolerable for the benefit of more accurate—and earlier—straggler prediction in real time. Indeed, the scheduling results from Section 5.4 include this overhead.

<table>
<thead>
<tr>
<th></th>
<th>(a) Prediction</th>
<th>(b) Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Google</td>
<td>Alibaba</td>
</tr>
<tr>
<td>Corr</td>
<td>19,939</td>
<td>227,395</td>
</tr>
<tr>
<td>LogReg</td>
<td>89,204</td>
<td>844,260</td>
</tr>
</tbody>
</table>

Table 4(b) shows Agatha’s performance for model interpretation, illustrating the number of tasks diagnosed per second with both a single core and all available cores. The performance heavily depends on the number of features, so Alibaba is much faster than Google. Interpretation is obviously expensive compared to prediction, but it enables Agatha to diagnose causes in real-time. In addition, the number of predicted stragglers is low, so the overhead of PFI will be paid much less frequently than that of the causal prediction.

Finally, Agatha’s overhead is much smaller than the latency of production traces operation, which is usually more than hundreds of seconds. For an individual task, Agatha produces predictions in microseconds and diagnoses causes of straggling in less than a second.

### 6 Conclusion

This paper introduces Agatha, a straggler prediction framework that makes accurate live predictions without assuming prior knowledge or carefully curated training data using techniques from causal analysis. Agatha’s key insight is that the measurable behavior (features) of non-stragglers and stragglers are different even before the long latency reveals itself. As such, Agatha employs propensity scores (PS) to quantify such difference. Although PS has been applied in post-facto causal inference in other sciences, Agatha is the first to show that it can be used for live predictions and easily incorporate with the schedulers to reduce completion time. Agatha also proposes permutation feature to gain insights into the straggling behavior. We find that Agatha predicts stragglers accurately and offers reliable model interpretability on live data before it sees any positive stragglers examples in training.
This work is strong evidence that causal modeling has advantages over conventional correlation-based modeling when training data is not carefully curated. We hope this work inspires systems researchers to consider the nature of training data before applying learning, and incorporate causal analysis to solve systems problems.

References


[9] Grant Ayers, Nayana Prasad Nagendra, David I August, Hyoun Kyu Cho, Svilen Kanev, Christos Kozyrakis, Trivikram Krishnamurthy, Heiner Litz,


[22] Jeffrey Dean. Achieving rapid response times in large online services. 2012.


[33] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: Learning a variable’s importance by studying an entire class of prediction models simultaneously. *Journal of Machine Learning Research*, 20(177):1–81, 2019.


[80] Xiaqi Ren, Ganesh Ananthanarayanan, Adam Wierman, and Minlan Yu. Hopper: Decentralized speculation-aware cluster scheduling at scale. In


