Faster Jobs in Distributed Data Processing using Multi-Task Learning

Neeraja J. Yadwadkar* Bharath Hariharan* Joseph Gonzalez*
Randy Katz*
{neerajay,bharath2,jegonzal,randy}@cs.berkeley.edu

Abstract

Slow running or straggler tasks in distributed processing frameworks [1, 2] can be 6 to 8 times slower than the median task in a job on a production cluster [3], despite existing mitigation techniques. This leads to extended job completion times, inefficient use of resources, and increased costs. Recently, proactive straggler avoidance techniques [4] have explored the use of predictive models to improve task scheduling. However, to capture node and workload variability, separate models are constructed for every node and workload, requiring the time consuming collection of substantial training data and limiting the applicability to new nodes and workloads. In this work, we observe that predictors for similar nodes or workloads are likely to be similar and can share information, suggesting a multi-task learning (MTL) based approach. We generalize the MTL formulation of [5] to capture commonalities in arbitrary groups. Using our formulation to predict stragglers allows us to reduce job completion times by up to 59% over Wrangler [4]. This large reduction arises from a 7 percentage point increase in prediction accuracy. Further, we can get equal or better accuracy than [4] using a sixth of the training data, thus bringing the training time down from 4 hours to about 40 minutes. In addition, our formulation reduces the number of parameters by grouping our parameters into node- and workload-dependent factors. We show that, in the event of a particular task having insufficient data, this helps us generalize and achieve significant gains over a naive MTL formulation [5].

1 Introduction

Distributed processing frameworks [1, 2] split a data intensive computation job into multiple smaller tasks, which are then executed in parallel on commodity clusters to achieve faster job completion. A natural consequence of such a parallel execution model is that slow-running tasks, commonly called *stragglers* [1, 6, 3, 7, 8], potentially delay overall job completion. Stragglers form a major hurdle in achieving near optimal job completion times — a recent study [3] shows that straggler tasks are on average 6 to 8 times slower than the median task of the corresponding job.

Proactive straggler mitigation techniques [9, 10, 4] attempt to schedule tasks in a way that limits the effect of stragglers by modeling straggler behavior. Recently, Wrangler [4] showed that incorporating predictive models of straggler behavior in the scheduler can lead to large improvements in job completion times.

However, to address heterogeneity in the nodes and changing workload patterns, proactive model based approaches have previously modeled each workload[1] and node independently. Independent models pose two critical challenges: (1) each new node and workload requires new training data which can take hours to collect, delaying the application of model based scheduling, and (2) clusters with many nodes may have only limited data for a given workload on each node leading to lower quality models.

These shortcomings can be addressed if each classifier is able to leverage information gleaned at other nodes and from other workloads. For instance when there is not enough data at a node for a workload, we can gain from the data collected at that node while it was executing other workloads, or from other nodes running the same workload. Such information sharing falls in the ambit of multi-task learning (MTL), where the learner is embedded in an environment of related tasks, and the learner's aim is to leverage correlations between the tasks to improve performance of all tasks. *The aim of this paper is to adapt MTL for learning a generalized predictor with better prediction accuracy and ultimately improve job completion times.*

In this work, we exploit explicit knowledge about the dependencies between tasks to improve the performance of MTL. In particular, we can group classifiers by workload and by node. To incorporate this group structure we generalize the formulation proposed by Evgeniou, et al. [5], to include clusters of tasks. Using our formulation to predict stragglers allows us to reduce job completion times by up to 59% over Wrangler [4]. This large reduction arises from a 7 percentage point increase in prediction accuracy. Further, we can get equal or better accuracy than [4] using a sixth of the training data, thus bringing the training time down from 4 hours to about 40 minutes. In addition, our formulation reduces

*University of California, Berkeley

[1] Clusters are used for different purposes, and statistics such as the kinds of jobs submitted, their resource requirements and the frequency at which they are submitted vary depending upon the usage. We call one such distribution of jobs a workload.

the number of parameters by grouping our parameters into node- and workload-dependent factors. We show that, in the event of a particular task having insufficient data, this helps us generalize and achieve significant gains over a naive MTL formulation [5].

Finally, while we have shown experiments on straggler avoidance, our learning formulation is general and can be applied to other systems that train node or workload dependent classifiers [10, 11]. For instance, ThroughputScheduler [10] uses such classifiers to allot resources to tasks, and can benefit from such multitask reasoning. We leave these extensions to future work.
To summarize, our key contributions are:

1. We propose a generalized formulation for MTL that considers clusters of tasks and allows us to reduce parameters, improving generalization.

2. Instead of machine learning datasets, we show the benefits of MTL in general and our formulation in particular on a real world application, where we

   (a) avoid stragglers better, improving job completion times significantly and reducing net resource usage, and

   (b) can work even with a sixth of the training data and thus a much shorter training period.

2 Background and Motivation

Today, data is getting generated at an unprecedented scale due to popular Internet-based computer applications that serve millions of users, such as e-commerce websites, social networks. The rate at which this data is growing has rendered parallel processing on commodity compute clusters an inevitable and an attractive option. Google originally proposed its MapReduce framework [1] allowing them to process enormous amount of data generated by various applications. MapReduce is highly scalable to large clusters of inexpensive commodity computers. Hadoop, a popular open source implementation of MapReduce [12], has been widely adopted by industries of various sizes.

For accelerating a job's completion time, MapReduce divides a data intensive computation *job* in multiple smaller *tasks*. These tasks are executed in parallel on multiple machines (nodes) in a compute cluster. A job finishes when all its tasks have finished execution. A key benefit of such distributed parallel processing frameworks is that they automatically handle failures, without needing extra efforts from the programmer. Two basic modes of failures include, failure of a node and failure of a task. If a node crashes, MapReduce re-runs all the tasks it was executing on a different node. If a task fails, MapReduce automatically re-launches it.

However, a tricky situation arises when a node is available but is performing poorly. This causes tasks scheduled on that node to execute slower than other tasks of the same job scheduled on other nodes in the cluster. Since a job finishes execution only when all its tasks have finished execution, such slow-running tasks, called *stragglers*, extend the job's completion time. This, in turn, leads to increased user costs.

Existing approaches for dealing with stragglers broadly fall into reactive and proactive categories. The MapReduce paper [1] identified the problem of stragglers and suggested *speculative execution* as a mitigation mechanism. This is a reactive scheme that is dominantly used on production clusters including those at Facebook and Microsoft Bing [8]. It operates in two steps: (1) wait-and-speculate if a task is executing slower than other tasks of the same job, and (2) replicate or spawn multiple redundant copies of such tasks hoping a copy will reach completion before the original. Due to the wait-and-speculate step, this scheme is inefficient in time. Also, due to the second step that replicates tasks, such mechanisms lead to increased resource consumption without necessarily gaining performance benefits. LATE [7] improves over speculative execution using a notion of progress scores, but still results in a resource wastage. Cloning mechanisms [3], being replication-based, also incur extra resources.

Proactive approaches aim at predicting straggler tasks before they are launched [9, 10, 4]. Thus, they are time efficient. They are also efficient in reducing the resources consumed by smarter scheduling and avoiding replication of tasks. Hence, we use a recently proposed proactive approach, Wrangler [4], as our baseline. We first review Wrangler's pipeline below, and then discuss the avenues for improvement.

2.1 Our Baseline – Wrangler: Wrangler has two components. (1) A *model builder* that trains a classifier to predict if a task launched at a node will become a straggler, given the current resource usage counters on the node. Training data for each node and workload is collected by recording the resource usage counters at the time a task is launched along with the relative duration of the tasks (i.e., did it straggle). Each such task forms a data point; the resource usage counters form its feature vector, and the label is whether or not the task became a straggler. (2) A *model-informed scheduler*. Before launching a task on a node, the scheduler collects the node's resource usage counters and runs the classifier. If the model predicts that the task will be a straggler, the scheduler does not assign the task to that node. It is later assigned to a node that is not predicted to create a straggler.

Due to the heterogeneity of nodes in a cluster, the model builder trains a separate classifier for each node. Note that to build a training set per node, every node should have executed sufficient number of tasks. Wrangler takes a few hours (approximately 2-4 hours, depending on the workload) for this process. Additionally, because each workload might be different, these models are retrained for every new workload. Thus, for every new workload that is executed on the cluster, there is a 2-4 hour model building period. In typical large production clusters with tens of thousands of nodes, it might be a long time before a node collects enough data to train the classifier.

Moreover, we may not always get enough data for each node executing a workload. For example, in our case, each task of a workload executed on a node amounts to a training data point. Placement of input data to tasks on nodes in a cluster is managed by the underlying distributed file system [13]. To achieve locality for faster reading of input data, sophisticated locality-aware schedulers [7, 14] will try to assign tasks to nodes already having the appropriate data. Based on popularity of the data, number of tasks assigned to a node could vary. Hence we may not get uniform number of training data points, i.e., tasks executed, across all the nodes in a cluster. There could be other reasons behind skewed assignment of tasks to nodes [15]: even when every map task has the same amount of data, a task may take longer depending on the code path it takes due to the data it processes. Hence, the node slots will be busy due to such long running tasks. This could lead to insufficient number of tasks assigned to some nodes.

These observations suggest that our training should work even when very little data is available.

2.2 Need for multitask learning: Our proposal is to leverage the correlations between the classifiers to reduce this model building time. Concretely, a task executing on a node will be a straggler because of a combination of factors. Some of these factors involve the properties of the node where the task is executing (for instance, the node may be memory-constrained) and some others involve particular requirements that the tasks might have in terms of resources (for instance, the task may require a lot of memory). These are workload-related factors. When collecting data for a new workload executing on a given node, one must be able to use information about the workload collected while it executed on other nodes, and information about the node collected while it executed other workloads.

This kind of sharing of information is precisely the motivation for the machine learning paradigm known as multitask learning. In MTL, we are given a set of learning tasks and we want to learn a classifier for each one. Each task has its own training data set, although typically all training points of all tasks live in the same feature space. The tasks are related to each other, and the goal of multitask learning is to leverage this relationship to improve performance or generalization of all the tasks.

In our formulation, each node-workload pair will form a task. However, unlike typical MTL formulations, our tasks are not simply correlated with each other; they share a specific structure, clustering along node- or workload-dependent axes. With this in mind, we describe our MTL formulation below.

3 Proposed Formulation

Suppose there are T tasks, with the training set for the t-th task denoted by $D_t = \{(\mathbf{x}_{it}, y_{it}) : i = 1, \ldots, m_t\}$, with $\mathbf{x}_{it} \in \mathbb{R}^d$. We begin with the formulation proposed by Evgeniou, et al. [5]. Evgeniou, et al. write the classifier $\mathbf{w}_t$ for task t as:

$$\mathbf{w}_t = \mathbf{w}_0 + \mathbf{v}_t \tag{3.1}$$

Here, $\mathbf{w}_0$ is a weight vector shared between all tasks and captures information shared between tasks, and $\mathbf{v}_t$ is a vector that specifies how $\mathbf{w}_t$ deviates from $\mathbf{w}_0$.

Learning then involves solving the following optimization problem:

$$\min_{\mathbf{w}_0, \mathbf{v}_t, b} = \lambda_0 \|\mathbf{w}_0\|^2 + \frac{\lambda_1}{T} \sum_{t=1}^{T} \|\mathbf{v}_t\|^2 + \sum_{t=1}^{T} \sum_{i=1}^{m_t} \xi_{it} \tag{3.2}$$

s.t

$$\begin{aligned} y_{it}((\mathbf{w}_0 + \mathbf{v}_t)^T \mathbf{x}_{it} + b) &\geq 1 - \xi_{it} \ \ \forall i, t \\ \xi_{it} &\geq 0 \ \ \forall i, t \end{aligned}$$

This formulation shares information equally among all the tasks. However, as argued before, our tasks cluster into groups along various axes. To capture such structure, we assume that the tasks are partitioned into G groups. Denote the group of the t-th task by $g(t)$. Then we can write the classifier $\mathbf{w}_t$ as:

$$\mathbf{w}_t = \mathbf{w}_0 + \mathbf{v}_t + \mathbf{w}_{g(t)} \tag{3.3}$$

In general, there may be multiple ways of splitting tasks into groups. In our application, one may split tasks into groups based on workload or on nodes. To formalize this, assume there are P ways of defining groups. The p-th partitioning has G_p groups, and the task t belongs to the $g_p(t)$ group under this partitioning. Now, we also have a separate set of weight vectors for

each partitioning p, and the weight vector of the g-th group of the p-th partitioning is denoted by $\mathbf{w}_{p,g}$. Then, we can write the classifier $\mathbf{w}_t$ as:

$$\mathbf{w}_t = \mathbf{w}_0 + \mathbf{v}_t + \sum_{p=1}^{P} \mathbf{w}_{p,g_p(t)} \tag{3.4}$$

Finally, note that $\mathbf{w}_0$ and $\mathbf{v}_t$ can also be seen as weight vectors corresponding to trivial partitions: $\mathbf{w}_0$ corresponds to the partition where all tasks belong to a single group, and $\mathbf{v}_t$ corresponds to the partition where each task is its own group. Thus, we can include $\mathbf{w}_0$ and $\mathbf{v}_t$ in our partitions and write Equation 3.4 as:

$$\mathbf{w}_t = \sum_{p=1}^{P} \mathbf{w}_{p,g_p(t)} \tag{3.5}$$

Intuitively, at test time, we get the classifier for the t-th task by summing weight vectors corresponding to each group to which t belongs.
The learning problem can then be generalized to:

$$\min_{\mathbf{w}_{p,g},b} \sum_{p=1}^{P}\sum_{g=1}^{G_p} \frac{\lambda_p \#(p,g)}{T} \|\mathbf{w}_{p,g}\|^2 + \sum_{t=1}^{T}\sum_{i=1}^{m_t} \xi_{it} \tag{3.6}$$

s.t.

$$\begin{aligned} y_{it}((\sum_{p=1}^{P} \mathbf{w}_{p,g_p(t)})^T \mathbf{x}_{it} + b) &\geq 1 - \xi_{it} \ \ \forall i,t \\ \xi_{it} &\geq 0 \ \ \forall i,t \end{aligned}$$

Here $\#(p,g)$ denotes the number of tasks assigned to the g-th group of the p-th partitioning. The scaling factor $\frac{\lambda_p \#(p,g)}{T}$ interpolates smoothly between λ_0 when all tasks belong to a single group, and $\frac{\lambda_1}{T}$, when each task is its own group.

3.1 Reduction to a standard SVM: One advantage of the formulation we use is that it can be reduced to a standard SVM, allowing the usage of off-the-shelf SVM solvers. Below, we show how this reduction can be achieved. For every group g of every partition p, define:

$$\tilde{\mathbf{w}}_{p,g} = \sqrt{\frac{\lambda_p \#(p,g)}{\lambda T}} \mathbf{w}_{p,g} \tag{3.7}$$

Now concatenate these vectors into one large weight vector $\tilde{\mathbf{w}}$:

$$\tilde{\mathbf{w}} = [\tilde{\mathbf{w}}_{1,1}^T, \ldots, \tilde{\mathbf{w}}_{p,g}^T, \ldots, \tilde{\mathbf{w}}_{P,G_P}^T]^T \tag{3.8}$$

Then, it can be seen that $\lambda\|\mathbf{w}\|^2 = \sum_{p=1}^{P}\sum_{g=1}^{G_p} \frac{\lambda_p \#(p,g)}{T} \|\mathbf{w}_{p,g}\|^2$. Thus with this change of variables, the regularizer in our optimization problem resembles a standard SVM. Next, we transform the data points $\mathbf{x}_{it}$ into $\phi(\mathbf{x}_{it})$ such that we can replace the scoring function with $\tilde{\mathbf{w}}^T \phi(\mathbf{x}_{it})$. This transformation is as follows. Again, define:

$$\phi_{p,g}(\mathbf{x}_{it}) = \delta_{g_p(t),g} \sqrt{\frac{\lambda T}{\lambda_p \#(p,g)}} \mathbf{x}_{it} \tag{3.9}$$

Here $\delta_{g_p(t),g}$ is a kronecker delta which is 1 if $g_p(t) = g$ (or, in other words, if the task t belongs to the group g in the p-th partitioning) and 0 otherwise. Our feature transformation is then the concatenation of all these vectors:

$$\phi(\mathbf{x}) = [\phi_{1,1}(\mathbf{x})^T, \ldots, \phi_{p,g}(\mathbf{x})^T, \ldots, \phi_{P,G_P}(\mathbf{x})^T]^T \tag{3.10}$$

It is easy to see that:

$$\tilde{\mathbf{w}}^T \phi(\mathbf{x}_{it}) = (\sum_{p=1}^{P} \mathbf{w}_{p,g_p(t)})^T \mathbf{x}_{it} \tag{3.11}$$

Intuitively, $\tilde{\mathbf{w}}$ concatenates all our parameters with their appropriate scalings into one long weight vector, with one block for every group of every partitioning. $\phi(\mathbf{x}_{it})$ transforms a data point into an equally long feature vector, by placing scaled copies of $\mathbf{x}_{it}$ in the appropriate blocks and zeros everywhere else.

With these transformations, we can now write our learning problem as :

$$\min_{\tilde{\mathbf{w}},b} \lambda\|\tilde{\mathbf{w}}\|^2 + \sum_{t=1}^{T}\sum_{i=1}^{m_t} \xi_{it} \tag{3.12}$$

s.t.

$$y_{it}(\tilde{\mathbf{w}}^T \phi(\mathbf{x}_{it}) + b) \geq 1 - \xi_{it} \ \ \forall i,t \tag{3.13}$$

$$\xi_{it} \geq 0 \ \ \forall i,t \tag{3.14}$$

which corresponds to a standard SVM. In practice, we use this transformation and change of variables both at train time and at test time.

3.2 Application to straggler avoidance: We apply this formulation to straggler avoidance as follows. Suppose there are N nodes and L workloads. Then there are NL tasks, and Wrangler trains as many models, one for each task. For our proposal, we consider four different notions of groups:

1. A single group consisting of all nodes and workloads. This gives us the single weight vector $\mathbf{w}_0$.
2. One group for each node, consisting of all L tasks belonging to that node. This gives us one weight vector for each node $\mathbf{w}_n, n = 1, \ldots, N$, that captures the heterogeneity of nodes.

3. One group for each workload, consisting of all N tasks belonging to that workload. This gives us one weight vector for each workload $\mathbf{w}_l, l = 1, \ldots, L$, that captures peculiarities of particular workloads.

4. Each task as its own group. Since there are NL tasks, we get NL weight vectors, which we denote as $\mathbf{v}_t$ (following the notation considered in [5]).

Thus, in our formulation, the weight vector $\mathbf{w}_t$ for a given workload l_t and a given node n_t is:

$$\mathbf{w}_t = \mathbf{w}_0 + \mathbf{w}_{n_t} + \mathbf{w}_{l_t} + \mathbf{v}_t \tag{3.15}$$

The corresponding training problem is then:

$$\begin{aligned}\min_{\mathbf{w},b} = \lambda_0\|\mathbf{w}_0\|^2 + \frac{\nu}{N}\sum_{n=1}^{N}\|\mathbf{w}_n\|^2 + \frac{\omega}{L}\sum_{l=1}^{L}\|\mathbf{w}_l\|^2 \\ + \frac{\tau}{T}\sum_{t=1}^{T}\|\mathbf{v}_t\|^2 + \sum_{t=1}^{T}\sum_{i=1}^{m_t}\xi_{it}\end{aligned} \tag{3.16}$$

s.t

$$\begin{aligned} y_{it}((\mathbf{w}_0 + \mathbf{w}_{n_t} + \mathbf{w}_{l_t} + \mathbf{v}_t)^T\mathbf{x}_{it} + b) \geq 1 - \xi_{it} \ \ \forall i,t \\ \xi_{it} \geq 0 \ \ \forall i,t \end{aligned}$$

where λ_0, ν, ω, τ are hyperparameters which we set using grid search on a validation set.

Different variants of this formulation can be achieved by removing one or more of the terms from Equation 3.15. Note that we can achieve this effect by setting the corresponding hyperparameters, λ_0, ν, ω, τ to ∞. For example, setting ω to ∞ will force all $\mathbf{w}_l$ to be set to 0. To be mathematically rigorous, one can take the equivalent feature transformation in Equation 3.10 and take the limit as one of the hyperparameters approaches ∞. The corresponding feature vector block will approach 0, and the corresponding weight vector block being a finite linear combination of the feature vectors will approach 0. At test time, these terms will not contribute.

3.3 Exploring the relationships between the weight vectors: Before getting into the experiments, we can get some insights on what our formulation will learn by looking at the KKT conditions. The lagrangian of the formulation in Equation 3.16 is:

$$\begin{aligned}\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \lambda_0\|\mathbf{w}_0\|^2 + \frac{\nu}{N}\sum_{n=1}^{N}\|\mathbf{w}_n\|^2 + \frac{\omega}{L}\sum_{l=1}^{L}\|\mathbf{w}_l\|^2 \\ + \frac{\tau}{T}\sum_{t=1}^{T}\|\mathbf{v}_t\|^2 + \sum_{t=1}^{T}\sum_{i=1}^{m_t}\xi_{it} - \sum_{t=1}^{T}\sum_{i=1}^{m_t}\beta_{it}\xi_{it} \\ + \sum_{t=1}^{T}\sum_{i=1}^{m_t}\alpha_{it}(1 - \xi_{it} - y_{it}(\mathbf{w}_t^T\mathbf{x}_{it} + b))\end{aligned} \tag{3.17}$$

Taking derivatives w.r.t the primal variables and setting to 0 gives us relationships between $\mathbf{w}_0, \mathbf{v}_t, \mathbf{w}_n$ and $\mathbf{w}_l$:

$$\lambda_0\mathbf{w}_0^* = \frac{\tau}{T}\sum_t \mathbf{v}_t^* \tag{3.18}$$

$$\nu\mathbf{w}_n^* = \frac{\tau}{T/N}\sum_{t:n_t=n} \mathbf{v}_t^* \tag{3.19}$$

$$\omega\mathbf{w}_l^* = \frac{\tau}{T/L}\sum_{t:l_t=l} \mathbf{v}_t^* \tag{3.20}$$

$$\lambda_0\mathbf{w}_0^* = \frac{\nu}{N}\sum_n \mathbf{w}_n^* \tag{3.21}$$

$$\lambda_0\mathbf{w}_0^* = \frac{\omega}{L}\sum_l \mathbf{w}_l^* \tag{3.22}$$

Evgeniou et al., [5] also obtain Equation 3.18 in their formulation, but the other relationships are specific to ours. These relationships imply that these variables shouldn't be considered independent. $\mathbf{w}_n$, $\mathbf{w}_l$ and $\mathbf{w}_0$ are scaled means of the $\mathbf{v}_t$'s of the group they capture.

3.4 Discussion and comparison to other formulations: While our formulation (Equation 3.6) is more general than that of [5], one might ask what we have gained. After all, a lot of prior work [16, 17, 18] has focussed on MTL using known or latent clusters of tasks. However, to see the advantage our formulation offers, consider the variant of Equation 3.15 where we remove $\mathbf{v}_t$. This variant does not have any task-specific parameters, but still captures both node- and workload-dependent properties of the learning problem. It is thus similar to a factorized model where the node and workload dependent factors are grouped into separate blocks. It has $(N + L)d$ parameters, whereas a formulation like that of [5, 16, 17] will still have NLd parameters (here d is the input dimensionality). Thus, it *reduces* the number of parameters while still capturing the essential properties of the learning problems.

In addition, since this variant no longer has a separate weight vector for each task, we can generalize to tasks (i.e node-workload pairs) that are completely unseen at train time: the classifier for such an unseen task t will simply be $\mathbf{w}_0 + \mathbf{w}_{n_t} + \mathbf{w}_{l_t}$. It thus explicitly uses knowledge gleaned from prior workloads run on this node (through $\mathbf{w}_{n_t}$) and other nodes running this workload (through $\mathbf{w}_{l_t}$). On the other hand, formulations such as [5] will have to fall back on the generic $\mathbf{w}_0$ in such situations, while it is unclear how [16, 17] can be adapted to such a case. This is especially an issue in our application where there may be a large number of nodes and workloads. In such cases collecting data for each task (i.e node-workload pair) will be time consuming, and generalizing to unseen tasks will be a significant advantage. We show such generalization in Section 4.

% Training Data	Wrangler [4]	f_0	f_n	f_l	$f_{0,n,l}$	$f_{0,t}$	$f_{0,t,l}$
1	Insufficient data	66.88%	63.47%	66.52%	65.58%	63.71%	66.22%
2	Insufficient Data	67.1%	63.31%	67.7%	67.54%	64.33%	67.71%
5	Insufficient Data	67.54%	68.07%	69.1%	69.75%	69.59%	69.06%
10	63.91%	67.79%	70.91%	69.39%	72.3%	73.09%	72.9%
20	67.19%	67.97%	72.6%	70.1%	72.94%	74.72%	74.8%
30	68.45%	68.52%	73.18%	70.31%	74.08%	75.87%	75.79%
40	69.65%	68.17%	73.93%	70.49%	74.33%	76.43%	76.38%
50	70.08%	67.96%	73.73%	70.74%	74.72%	76.87%	76.69%
66	70.78%	68.17%	73.74%	70.1%	75.39%	77.34%	77.32%

Table 1: Prediction accuracies of various MTL formulations for straggler prediction with varying amount of training data. See Section 4.2 for details.

4 Empirical Evaluation

Our dataset consists of real world traces from production clusters at Facebook and Cloudera's customers and has 4 workloads, which we denote as $FB2009, FB2010, CC\_b$ and CC_e. See [19, 4] for details on data and replay methodology. We evaluate our approach using two metrics: first, the classification accuracy, and second, the improvement in overall job completion time. Below, we describe (1) how we use different MTL formulations and prediction accuracy achieved by these formulations, (2) how we learn a classifier for previously unseen node and/or workload and prediction accuracy it achieves, (3) the improvement in the overall job completion times achieved by our formulation over Wrangler, and (4) reduction in resources consumed using our formulation compared to Wrangler.

4.1 Variants of proposed formulation: As specified in Section 3, we learn a weight vector of the form $\mathbf{w}_t = \mathbf{w}_0 + \mathbf{v}_t + \mathbf{w}_{n_t} + \mathbf{w}_{l_t}$ as shown in Equation 3.15. We consider several variants of this general formulation. We first consider individually $\mathbf{w}_0$, $\mathbf{w}_n$ and $\mathbf{w}_l$:

- f_0: In this formulation we set τ, ν and ω to ∞. This corresponds to removing $\mathbf{v}_t$, $\mathbf{w}_n$ and $\mathbf{w}_l$. This formulation thus learns a single global weight vector, $\mathbf{w}_0$, for all the nodes and all the workloads.

- f_n: We set τ, λ_0 and ω to ∞. This corresponds to only learning a $\mathbf{w}_n$, that is, one model for each node. This model learns to predict stragglers based on a node's resource usage counters across workloads, but it cannot capture any workload-dependent properties.

- f_l: We set τ, λ_0 and ν to ∞. This means we only learn $\mathbf{w}_l$, i.e., a workload dependent model across nodes executing a particular workload. This model learns to predict stragglers based on the resource usage pattern caused due to a workload across nodes, but ignores the node's characteristics.

The above three formulations either discard the node information, the workload information, or both. We now consider multi-task variants that capture both node and workload properties:

- $f_{0,n,l}$: We set τ to ∞, removing $\mathbf{v}_t$ entirely and only learning $\mathbf{w}_0$, $\mathbf{w}_l$ and $\mathbf{w}_n$. As described above, this formulation reduces the total number of parameters and can also generalize to unseen tasks.

- $f_{0,t}$: This is the formulation proposed by Evgeniou, et al. [5], and corresponds to setting ν and ω to ∞. Note that this formulation still has to learn on the order of NLd different parameters and hence, might generalize worse than $f_{0,n,l}$.

- $f_{0,t,l}$: This formulation extends the formulation in $f_{0,t}$ by additionally learning a weight vector for each of the workloads executing across a set of nodes. Thus, only ν is set to ∞.

4.2 Prediction accuracy: Our aim is to learn to predict stragglers using as small amount of data as feasible, as this corresponds to shorter data capture time. Table 1 shows the percentage accuracy of predicting stragglers with varying amount of training data. We are primarily interested in comparing the accuracy of these MTL formulations with that achieved by Wrangler's node-and-workload-specific classifiers. We observe that:

- With very small amounts of data, all MTL variants outperform Wrangler. In fact, all of f_0 to $f_{0,t,l}$ need only one sixth of the training data to achieve the same or better accuracy.

- It is important to capture both node- and workload-dependent aspects of the problem: $f_{0,n,l}, f_{0,t}$ and $f_{0,t,l}$ consistently outperform f_0, f_n and f_l.

FB2009		FB2010		CC_b		CC_e	
$f_{0,n,l}$	$f_{0,t}$	$f_{0,n,l}$	$f_{0,t}$	$f_{0,n,l}$	$f_{0,t}$	$f_{0,n,l}$	$f_{0,t}$
73.07%	45.29%	46.66%	48.33%	50.18%	49.43%	52.78%	68.15%
56.2%	57.51%	57.27%	58.68%	60.96%	53.45%	64.37%	48.88%
63.87%	55.51%	50%	48.82%	59.39%	53.38%	48.85%	65.12%
63.17%	47.67%	60.63%	57.44%	55.72%	49.53%	47.33%	73.9%
50.66%	42.38%	51.42%	56.19%	50.77%	44.58%	71.2%	59.85%

Table 2: Straggler Prediction accuracies of $f_{0,n,l}$ and $f_{0,t}$ on test data from an unseen node-workload pair. See Section 4.3 for details.

	FB2009		FB2010		CC_b		CC_e	
	Wrangler	$f_{0,n,l}$	Wrangler	$f_{0,n,l}$	Wrangler	$f_{0,n,l}$	Wrangler	$f_{0,n,l}$
Average	56.75%	96.37%	10.60%	21.77%	43.59%	44.67%	16.17%	17.72%
50p	5.29%	36.09%	-1.07%	7.43%	6.62%	0.66%	-10.61%	-3.52%
75p	62.38%	80.99%	2.21%	6.58%	45.22%	34.44%	0.20%	-2.49%
80p	62.07%	82.76%	3.74%	11.81%	50.41%	44.06%	3.33%	-1.48%
85p	74.30%	89.12%	5.60%	19.87%	56.79%	52.81%	5.17%	0.84%
90p	75.00%	90.48%	9.61%	41.78%	56.05%	54.51%	11.01%	-6.55%
95p	68.51%	88.48%	27.51%	41.08%	58.87%	63.70%	32.08%	2.16%
97p	65.81%	86.19%	39.66%	44.30%	62.09%	71.22%	13.07%	38.27%
98p	64.42%	84.84%	41.72%	43.35%	71.03%	72.98%	25.58%	31.19%
99p	59.98%	83.12%	27.77%	53.61%	43.12%	76.62%	15.84%	20.65%

Table 3: Improvement in the overall job completion times achieved by $f_{0,n,l}$ and Wrangler over speculative execution.

- $f_{0,t}$ and $f_{0,t,l}$ perform up to 7 percentage points better than Wrangler with the same amount of training data, with $f_{0,n,l}$ not far behind.

Note that $f_{0,n,l}$, $f_{0,t}$ and $f_{0,t,l}$ seem to perform similarly, with $f_{0,n,l}$ performing slightly worse. However, as mentioned earlier, formulation $f_{0,n,l}$ has reduced number of total parameters and, because it has no task-specific weight vector, can generalize to new tasks unseen at train time. This is in contrast to $f_{0,t}$ which has to fall back on $\mathbf{w}_0$ in such a situation, and thus may not generalize as well. We see next if this is indeed true.

4.3 Prediction accuracy for a task with insufficient data: We trained classifiers based $f_{0,n,l}$ and $f_{0,t}$ leaving out 95% of the data of one node-workload pair every time. We then test the models on the left-out data. Table 2 shows the percentage classification accuracy from 20 such runs. We note the following:

- For 13 out of 20 classification experiments, $f_{0,n,l}$ performs better than $f_{0,t}$. For 10 out of these 13 cases, the difference in performance is more than 5 percentage points.
- For workloads $FB2009$ and CC_b, we see $f_{0,n,l}$ performs better consistently.
- $f_{0,n,l}$ sometimes performs worse, but in only 3 of these cases is it significantly worse (worse by more than 5 percentage points). All 3 of these instances are in case of the CC_e workload. In general, for this workload, we also notice a huge variance in the numbers obtained across multiple nodes. See [4] for a discussion of some of the issues in this workload.

This shows that $f_{0,n,l}$ works better in real-world settings where one cannot expect enough data for all node-workload pairs. We, therefore, evaluate $f_{0,n,l}$ in our next experiment (Section 4.4) to see if it improves job completion times.

4.4 Improvement in overall job completion time: We now evaluate our formulation, $f_{0,n,l}$, using the second metric, improvement in the overall job completion times over speculative execution. We compare these improvements to that achieved by Wrangler (Table 3). Improvement at the 99^{th} percentile is a strong indicator of straggler mitigation techniques. We see that $f_{0,n,l}$ significantly improves over Wrangler, reflecting the improvements in prediction accuracy. At the 99^{th} percentile, we improve Wrangler's job completion times by $57.8\%, 35.8\%, 58.9\%$ and 5.7% for $FB2009, FB2010, CC\_b$ and CC_e respectively.

Workload	% Reduction in total task-seconds	
	(MTL)	(Wrangler)
FB-2009	73.33	55.09
FB-2010	8.9	24.77
CC_b	64.12	40.15
CC_e	13.04	8.24

Table 4: Resource utilization with $f_{0,n,l}$ and with *Wrangler* over speculative execution, in terms of total task execution times (in seconds) across all the jobs. $f_{0,n,l}$ reduces resources consumed over Wrangler for $FB2009$, CC_b and CC_e.

4.5 Reduction in resources consumed: When a job is launched in the cluster, it will be broken into small tasks and each task will be run in a distributed fashion. Thus, to calculate the resources used, we can sum the resources used by all the tasks. As in [4], we use the time taken by each task as a measure of the resources used by the task. Note that, because these tasks will likely be executing in parallel, the total time taken by the tasks will be much larger than the time taken for the whole job to finish, which is what job completion time measures (shown in Table 3). Ideally, straggler prediction will prevent tasks from becoming stragglers. Fewer stragglers means fewer tasks that need to be replicated by straggler mitigation mechanisms (like speculative execution) and thus lower resource consumption. Thus, improved straggler prediction should also reduce the total task-seconds i.e., resources consumed.

Table 4 compares the percentage reduction in resources consumed in terms of total task-seconds achieved by $f_{0,n,l}$ and Wrangler over speculative execution. We see that the improved predictions of $f_{0,n,l}$ reduce resource consumption significantly more than Wrangler for 3 out of 4 workloads, thus confirming our intuitions. In particular, for $FB2009$ and CC_b, $f_{0,n,l}$ reduces Wrangler's resource consumption by about 40%, while for CC_b the reduction is about 5%.

5 Prior Work

Straggler mitigation: Reducing the completion times for jobs running on distributed processing frameworks [1, 2] is a well studied problem [20, 21, 22]. Stragglers are one of the major contributors to elongated job completions, and several approaches [6, 7, 8, 3, 23] attempt to mitigate them. Most of these approaches are reactive and replicative - they act only when tasks are already running slow and launch redundant copies of such tasks leading to additional resource consumption.

Scheduling or load-balancing approaches [20, 14, 24, 25, 22, 21] though proactive, rely on advance knowledge of causes behind stragglers or situations causing stragglers. Thus, they could miss dynamically changing loads on the nodes, resource requirements and communication patterns that are likely to cause stragglers [6, 4].

Other proactive approaches [9, 10, 4] use machine learning to predict slow-down of tasks. Wrangler showed a model-informed scheduler that uses the predictions to avoid stragglers by adapting to dynamically changing cluster resource usage patterns. In this work, we improve the accuracy of these predictions by sharing data across nodes in a cluster, thus using lesser training data per node and per workload. We also show how to predict stragglers for new nodes and for new workloads.
Multitask learning: The idea that multiple learning problems might be related and can gain from each other dates back to Thrun [26] and Caruana [27]. This notion was formalized by, among others, Baxter [28] and Ando, et al. [29], who considered a learner embedded in an environment of related learning problems and quantified the gain it could get by sharing data between these learning problems.

Much of this early work relied on neural networks as a means of learning these shared representations. However, contemporary work has also focussed on SVMs and kernel machines. Our work is an extension of the work of Evgeniou, et al. [5], who proposed an additive model for MTL that decomposes classifiers into a shared component and a task-specific component. In later work, Evgeniou, et al. [16], propose a general framework for MTL that uses a general quadratic form as a regularizer. They show that if the different tasks can be grouped into clusters, they can use a regularizer that encourages all the weight vectors of the group to be closer to each other. Jacob, et al. [17], extend this formulation when the group structure is not known a priori. Xue, et al. [18], also infer the group structure, but use a Bayesian approach instead. The formulation we propose is also designed to handle group structure, but offers us several advantages over this prior work. In particular, our formulation allows us to dispense with task-specific classifiers entirely, reducing the number of parameters drastically. This allows us to handle tasks that have very little training data by transferring parameters learnt on other tasks. There are other ways of controlling parameters, such as by using a $2,1$-norm to encourage similar features to be selected [30] or by learning a distance metric [31]. Our model is also similar to using low rank regularizers [32].

6 Conclusion

Through this work, we have shown the utility of multitask learning in solving the real-world problem of avoiding stragglers in distributed data processing. We have presented a novel MTL formulation that captures the structure of our tasks and reduces job completion times by up to 59% over prior work [4]. This reduction comes

from a 7 percentage point increase in prediction accuracy. Our formulation can achieve better accuracy with only a sixth of the training data and can generalize better than other MTL approaches for tasks with little or no data. Finally, we note that, although we use straggler avoidance as the motivation, our formulation is more generally applicable, especially for other prediction problems in distributed computing frameworks, such as resource allocation [10, 11].

References

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[2] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *EuroSys*, 2007.

[3] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Effective straggler mitigation: Attack of the clones. In *NSDI*, 2013.

[4] Neeraja J. Yadwadkar, Ganesh Ananthanarayanan, and Randy Katz. Wrangler: Predictable and faster jobs using fewer resources. In *Symposium on Cloud Computing (To appear; short version included in supplementary)*. 2014.

[5] Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *KDD*, 2004.

[6] Ganesh Ananthanarayanan, Srikanth Kandula, Albert Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris. Reining in the outliers in map-reduce clusters using mantri. In *OSDI*, 2010.

[7] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy H. Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *OSDI*, 2008.

[8] Ganesh Ananthanarayanan, Michael Chien-Chun Hung, Xiaoqi Ren, Ion Stoica, Adam Wierman, and Minlan Yu. Grass: Trimming stragglers in approximation analytics. In *NSDI*, 2014.

[9] Edward Bortnikov, Ari Frank, Eshcar Hillel, and Sriram Rao. Predicting execution bottlenecks in map-reduce clusters. In *HotCloud*, 2012.

[10] Shekhar Gupta, Christian Fritz, Bob Price, Roger Hoover, Johan Dekleer, and Cees Witteveen. Throughputscheduler: Learning to schedule on heterogeneous hadoop clusters. In *ICAC*, 2013.

[11] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. In *ASPLOS*, 2014.

[12] Hadoop. http://hadoop.apache.org.

[13] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *SOSP*, 2003.

[14] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Eurosys*, 2010.

[15] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. Skewtune: Mitigating skew in mapreduce applications. In *SIGMOD*, 2012.

[16] Theodoros Evgeniou, Charles A Micchelli, Massimiliano Pontil, and John Shawe-Taylor. Learning multiple tasks with kernel methods. *JMLR*, 6(4), 2005.

[17] Laurent Jacob, Jean philippe Vert, and Francis R. Bach. Clustered multi-task learning: A convex formulation. In *NIPS*. 2009.

[18] Ya Xue, Xuejun Liao, Lawrence Carin, and Balaji Krishnapuram. Multi-task learning for classification with dirichlet process priors. *JMLR*, 8, 2007.

[19] Yanpei Chen, Sara Alspaugh, and Randy H. Katz. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *PVLDB*, 5(12), 2012.

[20] Matei Zaharia. The Hadoop Fair Scheduler. http://developer.yahoo.net/blogs/hadoop/FairSharePres.ppt.

[21] Ganesh Ananthanarayanan, Ali Ghodsi, Andrew Wang, Dhruba Borthakur, Srikanth Kandula, Scott Shenker, and Ion Stoica. PACMan: Coordinated memory caching for parallel jobs. In *NSDI*, 2012.

[22] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. Quincy: Fair scheduling for distributed computing clusters. In *SOSP*, 2009.

[23] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. Skewtune: Mitigating skew in mapreduce applications. In *SIGMOD*, 2012.

[24] Hadoop's Capacity Scheduler. http://hadoop.apache.org/core/docs/current/capacity_scheduler.html.

[25] Faraz Ahmad, Srimat T. Chakradhar, Anand Raghunathan, and T. N. Vijaykumar. Tarazu: Optimizing mapreduce on heterogeneous clusters. In *ASPLOS*, 2012.

[26] Sebastian Thrun. Is learning the n-th thing any easier than learning the first? *NIPS*, 1996.

[27] Richard A Caruana. Multitask learning: A knowledge-based source of inductive bias. In *Proceedings of the 10th International Conference of Cognitive Science*.

[28] Jonathan Baxter. A model of inductive bias learning. *JAIR*, 12, 2000.

[29] Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *JMLR*, 6, 2005.

[30] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3), 2008.

[31] Shibin Parameswaran and Kilian Q. Weinberger. Large margin multi-task metric learning. In *NIPS*. 2010.

[32] Ting Kei Pong, Paul Tseng, Shuiwang Ji, and Jieping Ye. Trace norm regularization: reformulations, algorithms, and multi-task learning. *SIAM Journal on Optimization*, 20(6), 2010.