Concerto: Leveraging Ensembles for Timely, Accurate Model Training Over Voluminous Datasets

Walid Budgaga[†], Matthew Malensek[‡], Sangmi Lee Pallickara[†], and Shrideep Pallickara[†]
Department of Computer Science
[†] *Colorado State University, Fort Collins, USA*
[‡] *University of San Francisco, San Francisco, USA*
[†]{*wbudgaga, shrideep, sangmi*}*@cs.colostate.edu*, [‡]*mmalensek@usfca.edu*

***Abstract*—As data volumes increase, there is a pressing need to make sense of the data in a timely fashion. Voluminous datasets are often multidimensional with individual data points representing a vector of features. Data scientists fit models to the data — using all features or a subset thereof — and then use these models to inform their understanding of phenomena or make predictions. The performance of these analytical models is assessed based on their accuracy and ability to generalize on unseen data. Several frameworks exist for drawing insights from voluminous datasets, but have limited scalability (which leads to prolonged training times), poor resource utilization, narrow applicability across problem domains, and insufficient support for combining diverse model fitting algorithms.**

In this study, we describe our methodology for scalable supervised learning over voluminous datasets. The methodology explores the effect of controlled partitioning of the feature space, as well as how analytical models can be combined to preserve accuracy. Rather than build a single, all-encompassing model, we enable practitioners to construct an ensemble of models that are trained independently in parallel over different portions of the data space. This can provide faster training times and increased prediction accuracy overall; our empirical benchmarks demonstrate the suitability of our approach using real-world data.

***Index Terms*—Distributed ensemble learning, voluminous datasets, parallel model training**

1. Introduction

A confluence of factors has contributed to exponential growth in data volumes. Improvements in disk densities, capacities, and quality alongside falling costs have made it cheaper to store increasing amounts of data. Miniaturization and improvements in sensing equipment have led to a proliferation of observational devices and continuous sensing environments, such as the Internet of Things (IoT). Consumer devices (smartphones, tablets, and so on) also add substantially to overall data volumes, as well as simulations, sales tracking, and commercial data collection and harvesting activities, including social media.

Voluminous datasets offer a multitude of opportunities to extract insights. These insights can be used to understand phenomena, inform planning, and make forecasts based on current conditions. This often involves leveraging machine learning algorithms that fit models to the data, such as linear regression, non-linear structures (such as those in artificial neural networks), decision trees, and deep networks.

The model fitting process involves feature extraction, preprocessing, and transformation. A key component of this process is iterative *training* where *weights* are assigned to features, incrementally refined, and hyperparameters associated with the model are tuned. Model performance is assessed using cross validation (by generating multiple *folds* of the dataset) or with a separately held *validation* dataset. The model fitting process underpins our understanding of how features interact with each other, their relative importances, and the influence features have on particular phenomena. A performant model can then be used as a surrogate for the data.

Model fitting and training is time consuming. However, once trained, model evaluations to predict or classify phenomena can often be performed in real time. Training times, model accuracy, disk and network I/O, CPU utilization, and memory usage are all considered in the performance evaluations carried out in this study.

1.1. Research Challenges

Model creation over voluminous data poses unique I/O and CPU challenges:

1) **Scalability**: In this problem domain, datasets tend to grow continuously. Solutions must be able to scale up to meet the demands of growing data volumes.
2) **Computation**: model fitting is computationally expensive and often involves, inter alia, determining coefficients associated with features, the interactions between them, and addressing model complexity either as part of the algorithm or via regularization constraints.
3) **I/O Costs**: the I/O subsystem is several orders of magnitude slower than memory. Further, improvements to I/O hardware happen at a much slower pace [1]. Data must be read from disks prior to fitting models.
4) **Generalization**: our approach must cope with both overfitting and underfitting to ensure that the resulting model performs well with unseen data.

1.2. Research Questions

The primary objective of this study is to facilitate scalable creation of machine learning models over voluminous datasets. Specific research questions that we explore include:

RQ1 *How can we reduce model training times?* As data volumes increase, training times increase. Timeliness is critically important when making predictions that must be capitalized or acted upon.

RQ2 *How can we ensure that this reduction in training times is not at the expense of accuracy?* We will contrast the accuracy of the models using our methodology with that of a canonical model that was trained without regard for training times, and with a focus on accuracy.

RQ3 *How can we ensure that the methodology scales with increases in data volumes and the number of machines available?* Coping with increased data by adding machines should result in roughly constant training times.

1.3. Approach Summary

The crux of our methodology, *Concerto*, is to provide faster model creation/training over voluminous data. In particular, we focus on regression models. Our methodology targets: (1) alleviation of I/O constraints, (2) creation of independent models in a distributed environment, (3) ensuring data locality during the model creation process, and (4) combining several methods to preserve prediction accuracy.

Training a single, all-encompassing model may involve network I/O and synchronization barriers for updating model parameters because the data volumes are often far too big to be entirely resident on a single disk. In such cases, the learning algorithm is locally applied to data hosted by worker machines in parallel to find subsolutions that are synchronized and aggregated to update the model. The process is iterative and continues until a stopping condition is reached. Note that some algorithms are sequential in nature and cannot be easily parallelized.

Rather than build a single, all-encompassing model, we launch multiple training processes to fit several models in parallel. In our setting, data is partitioned and distributed over a collection of machines. Each model has data locality; the model is restricted to, and locally trained on, data available on a given machine without data transmission. Each model instance is trained independently, ranks features, performs feature selection, and so on.

Model creation leverages the distributed environment by ensuring that training executes in parallel on multiple machines. If there are N models, and assuming that the data is distributed more or less uniformly, then we can achieve an N-fold speedup in training times. The key reason for the speedup is that I/O and CPU costs are amortized over the distributed collection of machines. This also facilitates straightforward fault tolerance measures.

Individual model instances are used to construct an *ensemble* whose behavior depends on how the training dataset was partitioned. The ensemble exploits the insights gained from the individual models and partitioned data to generate a prediction for a given observation.

1.4. Paper Contributions

In this study, we describe our framework for fast, distributed creation of model ensembles over voluminous datasets. While it is possible to implement our methodology with other machine learning frameworks or distributed computation engines, *Concerto* is integrated and purpose-built for extreme-scale datasets and scenarios that warrant distributed ensembles. The methodology is demonstrably scalable while maintaining accuracy, and does not require synchronization barriers or network communication between participating machines. These advantages are facilitated by our ensemble data partitioning and training functionality. Specific contributions include:

1) Amortizing I/O and CPU costs by orchestrating model creation/training in a distributed environment. Models are built independently to ensure scalability and do not require synchronization.
2) Preserving accuracy by considering all observations in the training process across data partitions instead of sampling or reducing the dimensionality of the dataset.
3) Providing fault tolerance by relaunching failed learning processes on new machines without impacting the rest of the training process.
4) Allowing for combinations of diverse machine learning models to make a single prediction, and flexibility to easily incorporate new models in the future.
5) Supporting inherently sequential algorithms without modifications.

Finally, while this study focuses on regression models, we posit that our methodology is equally applicable to classification problems.

2. Related Work

Sampling is a commonly-used strategy for dealing with voluminous datasets when computational resources are insufficient. Sampling relies on a random process to create a smaller version of the original data; the subset can then be used as a surrogate for building the analytical models. For many domains, strategies [2], [3], [4] can create a smaller representative subset of the original dataset that enables building analytics models with performance similar to the original [5]. Although sampling enables analysis of voluminous data, studies have shown that increasing the sampling size of such data usually leads to better accuracy [6].

Another approach is to build a complex model using voluminous data distributed across several networked machines. Such approaches are effective in industry-scale settings involving training datasets with sizes ranging from 1TB to 1PB [7]. However, the cost of moving voluminous data among distributed machines is prohibitive, and cost concerns force the proposed systems to support data-centric computation. Collocating the computations with distributed

data portions involves exploiting the natural decomposability of the objective function over the training data or, if possible, reworking the learning algorithm to run in a distributed manner. Often, approaches must carefully consider both task- and data-level parallelism to be effective [8].

A broad spectrum of machine learning problems involves the execution of iterative computations. To enable such a feature on large-scale data, some approaches have relied on the optimization of the MapReduce framework [9], [10], and others employed a graph abstraction to express computations [11]. Additional frameworks [12], [13], [7], [14] have been proposed to support a broader class of machine learning algorithms.

The main issue with most distributed approaches is expensive synchronization barriers involved in updating model parameters for each refining step, which leads to long training times. To minimize this issue, distributed systems approaches often relax the synchronization step at the expense of accuracy. Additionally, some algorithms are sequential in nature and simply cannot be efficiently distributed.

3. Methodology

The primary challenges associated with training an all-encompassing, distributed model on a voluminous dataset are long training times, infeasibility for some learning algorithms, and inefficient utilization of computational resources. *Concerto* aims to overcome these challenges and consists of three tasks:

(A) Partitioning the dataset into several subsets with manageable sizes and dispersing them over the cluster. We support two methods: *input* and *output* partitioning.
(B) Launching parallel learning processes that train models over the partitions independently.
(C) Building ensembles by evaluating model performance and configuring a *gate function* to select appropriate models to use when making predictions.

These tasks are summarized in Figure 1. Our methodology enables the construction of scalable ensembles that have the potential to generalize well. The main intention behind our use of partitioning is to enable learning in parallel from voluminous datasets with high prediction accuracy by capturing localized patterns associated with different regions of the input/output space. In doing so, the resulting gate function will select models that are most specialized for particular subsets of observations.

3.1. Partitioning the Data

Before dispersing data across the cluster, we sample from the dataset to create a *guidance set* that is used for ensemble assessment. The guidance set also provides a representative snapshot of dataset properties in situations where new information continually streams into the system. We rely on k-fold cross-validation to train, assess, and tune the complexity of the individual models. Partitioning methods can provide high-quality training sets that ultimately lead

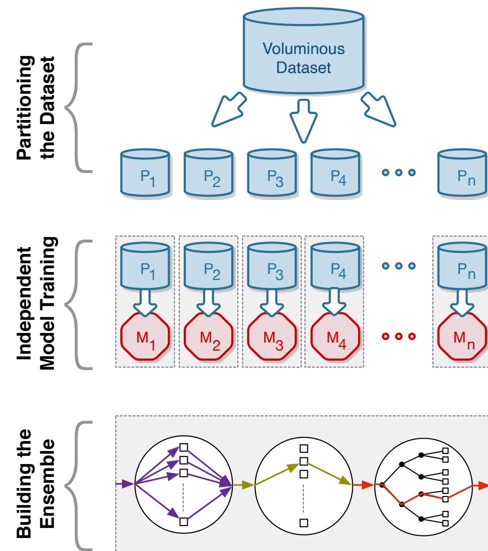


Figure 1: Tasks carried out by *Concerto*. Datasets are analyzed before being partitioned and distributed to the cluster. Next, models are built independently, and then their predictive performance is evaluated to select which models are used in the ensemble.

to ensembles that generalize well; partitioning has a direct impact on the contents of training sets used for building the models that comprise the ensemble. We support two partitioning methods, *input*- and *output*-based partitioning.

3.1.1. Input-Based Partitioning.

this partitioning algorithm divides the dataset into groups based on their similarity in the input space. Each group includes observations occurring in proximate portions of the feature space. To simplify the partitioning process, we do not attempt to find the optimal number of groups (clusters). The number of groups is determined based on the size of the given dataset and available computational resources, where several groups may be collocated on the same physical machine.

Our framework can employ a variety of techniques to accomplish this objective, including locality-sensitive hashing methods [15], [16], ball tree algorithms [17], and clustering methods [18], [19] such as k-means. After the groups have been established, data is dispersed across the cluster. However, it is important to note that all of these algorithms are susceptible to the *curse of dimensionality*, i.e., when dimensionality is too high the input space becomes increasingly sparse and the Euclidean distances between all of the data points are more or less indistinguishable [20].

We use dimensionality reduction techniques to deal with high-dimensional datasets. Input features that contribute the largest portion of the model's explanatory power are identified and selected to represent the data; in general, this does not have an appreciable impact on model performance [21]. Methods such as principal component analysis (PCA), correlation analysis, least absolute shrinkage and selection operator (LASSO), and Random Forest ensembles can be used

for dimensionality reduction. Regardless of the technique used for dimensionality reduction, our framework provides a uniform interface that emits $\langle feature, importance~\% \rangle$ pairs for evaluation. Also note that selecting the groups/clusters and performing dimensionality reduction both occur on the guidance set **before** the entire dataset is ingested into the cluster.

3.1.2. Output-Based Partitioning. here, our algorithm aims to find the best *split thresholds* in the input space to create regions with maximum similarity in the output space. While our input-based partitioning builds groups of similar multidimensional observations (based on input *features*), this approach creates groups that ultimately lead to similar model outcomes (sometimes referred to as *label-*based partitioning).

We employ the *C4.5* decision tree generation algorithm [22], an extended version of ID3 (Iterative Dichotomiser 3) [23]. Briefly, the algorithm attempts to choose split thresholds that maximize Kullback–Leibler divergence (also known as *information gain*) [24] to produce branches that are as homogeneous as possible. In other words, split thresholds are chosen with the intent of reducing entropy in the resulting partitions of the output space.

The algorithm is applied recursively on each region until the desired number of partitions is reached. Once complete, the resulting decision tree is pruned to remove branches that ultimately were not useful in distinguishing between outcomes. This partitioning method puts all of the observations that lead to similar target values in a separate subset, which in turn enables the construction of an ensemble with models that are specialized for different areas of the output space. The split thresholds are later used to determine which models within the ensemble will likely make the best predictions overall.

3.2. Independent Model Training

Once the dataset has been partitioned and distributed, training individual models in parallel can begin. Our framework has two components, a **driver** and several **executors**. The driver's inputs include the ensemble type (input or output), the partitioning information generated during the partitioning phase (e.g., centroids or split thresholds), and a list of machines that host the training data. When the driver starts, it transmits a copy of the partitioning information to all machines in the cluster and then launches an executor on each machine. The executor runs multiple training processes in parallel on the host machine based on memory capacity and number of cores, as well as the size of the training sets. An overview of the training process is shown in Figure 2.

The training process begins by performing a correlation analysis and building *pilot models* on the local datasets to help pinpoint influential features. This information is useful both for building ensembles and providing diagnostic information. In some cases, datasets may exhibit *collinearity* where some features can be predicted linearly by others with a great degree of accuracy. Depending on the learning algorithm, this can produce noise that leads to decreased prediction or classification performance, so we optionally remove such features during this phase.

The framework supports a variety of machine learning models implemented as plugins that can be configured by the user on a case-by-case basis. We rely on configurable k-fold cross-validation ($k = 10$ by default) to assess model performance and iteratively tune each model's hyperparameters using general guidelines specific to the particular learning algorithm. Our current implementation supports the following machine learning models:

- Linear Regression
- Decision Tree Regression
- Random Forests
- Gradient Boosting

Note that the individual models themselves can be ensembles as well. Adding new models to the framework is straightforward and requires implementing a single interface.

Once local training is complete, serialized copies of the models that exhibited the best performance are checkpointed to the disk as a safeguard against outages or recoverable failures. Since the trained models are quite small, they are also broadcast to the rest of the cluster. This consumes a minimal amount of bandwidth and is leveraged during the ensemble creation process to build meta-models that achieve higher performance than the individual models. Note that this step happens asynchronously as training proceeds across the entire cluster. If a training process fails, the executor first retries executing it and then if the failure persists stores detailed logs to help diagnose parameterization or system issues.

3.3. Building the Ensemble

Once local training is complete, the final step of our methodology produces an ensemble using *meta learning* techniques, where models are trained on other models' outputs. These meta models serve as a *gate function* that determines which model(s) should be used under particular conditions. Our framework leverages *stacking* to produce the final ensemble.

The base ensemble is constructed by using specialized models that learn local patterns and trends, but none of these

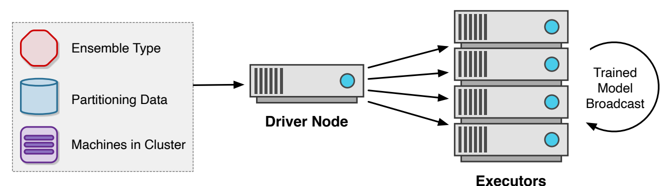


Figure 2: Overview of the training process. Inputs to the **driver** include the ensemble type, partitioning data, and list of machines in the cluster. The driver transmits this information to the **executors** that schedule and manage training processes.

models will be able to efficiently capture global trends and the patterns that stretch over multiple regions of the feature space. Our first improvement to baseline performance is allowing models that are specialized for neighboring regions to vote or average their predictions to provide a single accurate prediction for a given observation coming from one particular part of the feature space.

After building the specialized models for the base ensemble, new training sets are needed to build the meta models. We create a training set for each region of the feature space by applying all of the specialized models to the particular region's data for making predictions. To avoid overfitting, the region's entire dataset is not used for training specialized models. New training subsets are created in parallel by using the machines that host the original data. Subsequently, a training process is collocated with the new datasets to build the meta models in parallel independently.

Relying on *stacking* incorporates the insights of the specialized models to build a meta-learner (referred to as a stacked model) for each region. Stacking builds a model on the predictions of the other models to provide a prediction that is more accurate than any of the combined predictions [25]. In this approach, several different learning algorithms are combined (such as linear regression, decision trees, and so on) to create the meta models. Stacking has been applied in several domains of machine learning and shown an improvement of prediction accuracy and generalization [26], [27], [28].

4. Experimental Evaluation

Herein we describe the experiments we conducted to assess the effectiveness of our methodology for learning from voluminous datasets.

4.1. Experimental Setup

To evaluate *Concerto*, we used a real-world climate dataset obtained from the National Oceanic and Atmospheric Administration North American Mesoscale Forecast System [29]. This dataset includes readings of surface pressure, surface temperature, snow cover, snow depth, relative humidity, wind speed, etc. The dataset contained 3,532,225,177 observations with 59 features. Of the overall set of features, 58 were used as input features and the `precipitable_water_entire_atmosphere` feature was chosen as our target to predict. The evaluation dataset was divided into test and training data. The test dataset contained 5 million observations and was used to assess the models, while the remaining data were used to train the global model and the ensembles.

To build the global model and ensembles, we used 126 machines running Fedora 30: 34 Six-Core Intel Xeon E5-2620v3; 16GB RAM, 42 Eight-Core Intel Xeon E5-2620v4; 16GB RAM, and 50 Eight-Core Intel Xeon E5-2620v4; 64GB RAM.

4.2. Comparison: Apache Spark and MLlib

To serve as a point of comparison with our approach, we used Apache Spark to build distributed machine learning models for our dataset. We call these *global* models. Spark was deployed on our 126-machine test cluster alongside Hadoop Distributed File System (HDFS) with the replication factor set to 3. Using 12 external machines to stage the data into the cluster took 168 minutes.

We applied all four machine learning algorithms on the entirety of the training data with Spark. While we could successfully train a global distributed model on the entire datasets using linear regression and basic decision tree regression, we were unable to do so with random forests or gradient boosting. While training the random forest, we observed that Spark attempted to shuffle large amounts of data that caused many of the nodes to fail because there was no space left on the disk (250 GB was the maximum amount of free space available on some of the machines in our cluster after accounting for the dataset storage, which is still a large amount of headroom).

Similarly, the gradient boosting training process never completed because nodes that ran out of space would trigger Spark's fault tolerance measures to recompute the entire RDD lineage chain. It is worth noting, however, that gradient boosting is a sequential algorithm and not particularly well-suited for distributed learning, especially with a large number of trees. Table 1 summarizes the experimental settings used to train the global model.

TABLE 1: Configurations for the all-encompassing models built with Spark. *Indicates training did not complete.

Algorithm	Depth	Max. Iterations / Trees
Linear Regression	N/A	10,000
Decision Trees	30	1
Random Forests*	9	1,000
Gradient Boosting*	4	1,600

Note: *Since we could train the linear regression and the decision tree models on our dataset, we used them as our main points of comparison. In the interest of fairness, we determined these parameters through extensive experimentation with the intent of demonstrating the best possible performance of Spark. These models are distributed and were configured to create enough decision trees to ensure the entire cluster was fully utilized.*

4.3. Building the Ensembles

We created an input and output ensemble to test both of our partitioning strategies. To create the input partitions, we used the k-means algorithm on 20% of the data to create 1,000 clusters. It took 12 minutes to calculate the centroids and 68 minutes to disperse the data. The centroids were used as the basis for the gate function of the ensemble.

For building the output ensemble, we partitioned the data based on the similarities in the output space. The partitioning

algorithm was applied to 15% of the data and took 38 minutes to complete. In this case, split thresholds were used to build the gate function for the ensemble. Distributing the data across the cluster took 46 minutes.

Unless otherwise noted, we used the same machine learning algorithms and parameters as the Spark cluster. *Concerto* manages orchestrating and leveraging the ensembles. The size of these ensembles is constrained by two competing pulls, parallelism and regularization: increasing the degree of parallelism reduces training times, but increasing model count beyond a certain point adds more complexity. The number of constituent models within the ensemble also depends on two additional factors: (1) the data and (2) the type (input or output) of ensemble being built. However, if the size of the ensemble is the same the "systems" implications are similar. Our experimental evaluation assesses the performance implications of these different choices. The majority of our benchmarks include results for ensembles with 1,000 models (input ensemble) and 5,357 models (output ensemble), which provided the best balance of speed and accuracy. However, a direct comparison (1,000 models vs. 1,000 models) is also provided in Section 4.9.

4.4. Training Times, CPU Usage, and Scalability

We contrasted the training time of all-encompassing, distributed (global) models built with Spark against two ensembles of different sizes for our two partitioning types. Here, training time does not include time spent ingesting and storing the dataset (168 minutes with Spark and HDFS, 80 minutes for the input ensemble, and 84 minutes for the output ensemble). Table 2 contains the results for the linear regression model; linear regression is straightforward to parallelize with minimal communication, so in this case the global model is faster to build.

TABLE 2: Training times for an all-encompassing linear regression model built with Spark compared against input and output ensembles trained with our framework.

Configuration	Training Time (Hours)
Spark (Global)	0.6
Ensemble (1000 Models)	1.7
Ensemble (5357 Models)	4.4

Note that while the global linear regression model outperforms our approach in terms of training time, it is one of the simplest models available and accuracy will generally be lower than other approaches. See Section 4.7 for an evaluation of prediction accuracy.

In the case of decision trees, the benefits of our methodology are much more apparent; Table 3 compares decision tree training times.

Figure 3 shows cumulative CPU utilization for the decision tree ensembles; in these results, steeper curves indicate better resource utilization (high CPU usage). Intuitively, the ensemble with more models also took longer to train. In fact, due to the load balancing characteristics of our approach, we could observe training times scaling up/down with changes to the number of models as well as the number of machines in the cluster.

TABLE 3: Comparison of decision tree training times.

Configuration	Training Time (Hours)
Spark (Global)	12.1
Ensemble (1000 Models)	0.4
Ensemble (5357 Models)	2.1

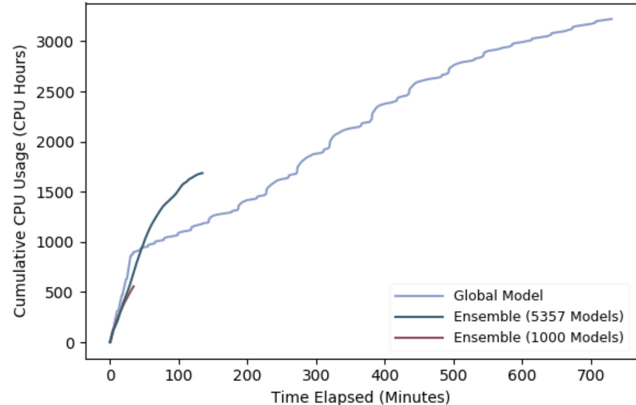


Figure 3: Cumulative CPU usage of the cluster for decision tree models. Steeper curves indicate higher CPU usage, while flatter curves indicate portions of the job that are likely I/O- or memory-bound. Here, the number of models in the ensembles trained by *Concerto* have a direct impact on training time.

Finally, Table 4 compares training times for gradient boosting. While the algorithm takes the longest amount of time to train, we found that it also usually provided the best prediction accuracy (see Section 4.8). Note that in this case, the global model built with Spark did not finish executing after running for several days.

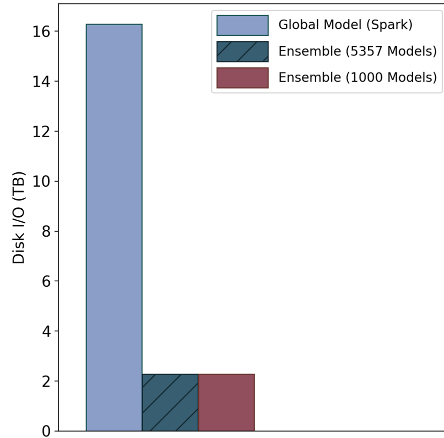
TABLE 4: Comparison of gradient boosting training times.

Configuration	Training Time (Hours)
Spark (Global)	Did not Complete
Ensemble (1000 Models)	26.1
Ensemble (5357 Models)	45.5

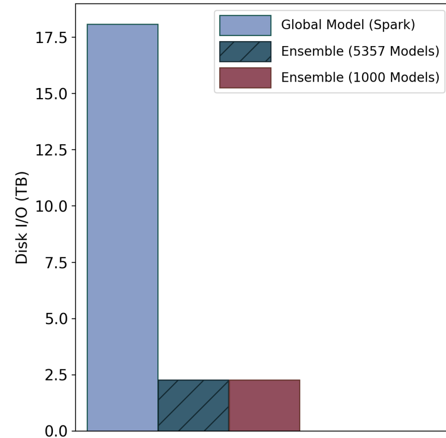
Based on these results, our approach has a clear advantage when the algorithm is not trivially parallelizable.

4.5. Disk and Network I/O

One key advantage of our approach is that it requires minimal I/O: intermediate data is not written to the disk, and while the partitions incur initial read costs, once they are resident in memory they will not have to be swapped out. This avoids expensive write operations. Figures 4-(a) and (b) compare the disk I/O usage of the two approaches (global vs.



(a) Disk I/O: Linear Regression



(b) Disk I/O: Decision Tree

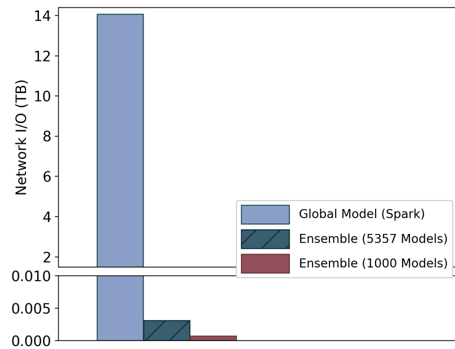
Figure 4: Comparison of Disk I/O (including both reads and writes) between the global model built with Spark and our ensembles for linear regression and decision trees. The ensembles read the dataset once, but do not require significant additional I/O operations.

ensemble-based). The high disk I/O exhibited in the Spark implementation includes reading from and writing to HDFS as well as any flushing of RDDs to the disk that may occur when nodes run out of main memory or checkpoints must be stored. We hypothesize that the primary reason we were unable to train a global random forest or gradient boosting model on the dataset using Spark is due to the higher I/O usage patterns shown here. Also note that the ensembles read the dataset once, but do not require significant additional I/O operations.

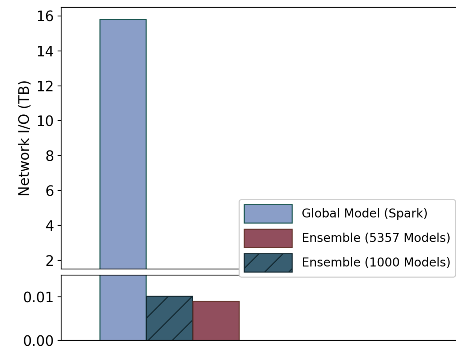
Figures 5-(a) and (b) compare network I/O measured in terabytes. The difference here can be primarily explained by the minimal amount of communication required by our framework; apart from initial partitioning information and broadcasting serialized model instances, very little coordination between nodes in the cluster is necessary. Variation between the different ensembles is primarily due to differences in the serialized model sizes and complexity. Minimal communication can be particularly beneficial when the links between nodes in the cluster incur high latency or are intermittently unreliable, which is often a reality in cloud deployments that are geographically distributed. In the case of Spark, a substantial amount of data must be transferred, likely due to training synchronization and the communications between HDFS and Spark.

4.6. Memory Usage

We tracked memory usage across the entire cluster during the training process. In the case of Spark, a large amount of memory was pre-allocated and then grew over time as RDDs were cached in memory. On the other hand, with *Concerto*, memory usage varied to a greater extent depending on the resource profiles of the training processes. This allows memory usage to decrease over time as training processes complete, potentially enabling other



(a) Network I/O: Linear Regression



(b) Network I/O: Decision Tree

Figure 5: Comparison of Network I/O (including both reads and writes) between the global model built with Spark and our ensembles for linear regression and decision trees.

processes to make use of the memory. Table 5 outlines the maximum and average memory usage observed on the cluster for each configuration. As usual, the input ensemble contained 1,000 models and the output ensemble contained 5,357 models for these benchmarks, which helps explain the difference in memory consumption.

TABLE 5: Memory usage exhibited while training linear regression models.

Configuration	Max. Memory (GB)	Avg. Mem. (GB)
Spark (Global)	965.14	858.18
Ensemble (1000 Models)	1065.27	638.36
Ensemble (5357 Models)	1769.85	1132.54

Similarly, Table 6 outlines the memory usage profile of the two systems while training decision tree models.

TABLE 6: Memory usage exhibited while training decision tree models.

Configuration	Max. Memory (GB)	Avg. Mem. (GB)
Spark (Global)	3490.47	3093.66
Ensemble (1000 Models)	1136.52	621.13
Ensemble (5357 Models)	430.05	262.80

In the case of the output ensemble, memory usage is lower due to the smaller partition sizes resulting in smaller models, but at a larger quantity (5,357). Since training is fairly fast, long-term memory consumption is low.

4.7. Prediction Accuracy

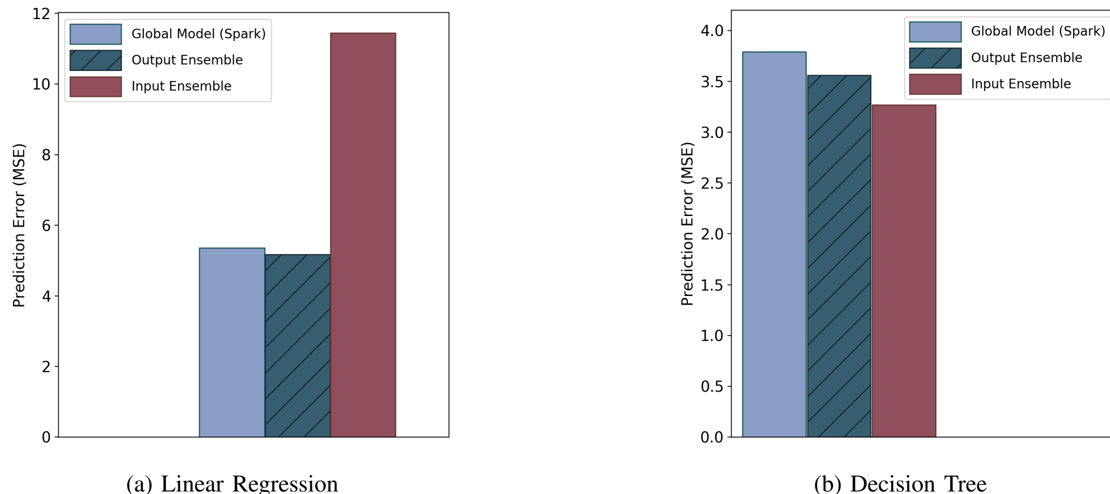
We measured the mean squared error (MSE), the average squared difference between estimated values and actual values, by making predictions with the global model and our ensembles on test data that was not used in the training processes. Figure 6 illustrates the prediction error measured in MSE for linear regression and decision tree models. In most cases, the predictions made by the input and output ensembles were more accurate than the global model; the global model is trained on the entire dataset and may not always be able to efficiently capture local patterns, while our methodology can do so by leveraging multiple models.

However, with the linear regression models, the prediction accuracy of the input ensemble was worse. We hypothesize that the training sets produced by partitioning the data based on the similarities in input space include non-linear relations that were poorly modeled at the local level. That is, patterns associated with different extents of input space could not efficiently be captured by the simple linear regression models.

Note that these results do not include the ensembles that featured gradient boosting models; since a direct comparison with the global model was not possible, gradient boosting results are provided in the next section.

4.8. Evaluating Ensembles: Input vs. Output

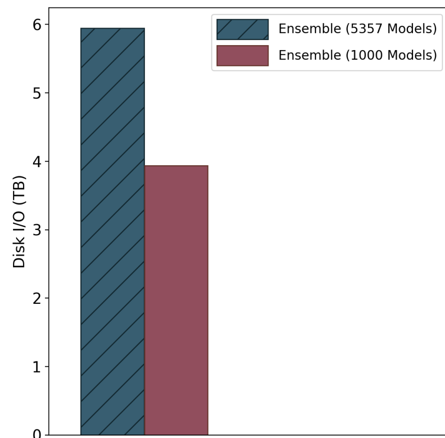
Due to the serial nature of the gradient boosting algorithm and the high amount of inter-node communication and disk I/O, we were unable to build a global model across the dataset using Spark. However, both our input and output ensembles with models trained using gradient boosting exhibited substantially better performance in terms of predictive accuracy; the best prediction MSE we achieved with Spark using decision trees was 3.75, whereas the MSE achieved with gradient boosting ensembles was 1.20 and 1.55 for the input and output ensembles, respectively. While the input ensemble achieved the highest prediction accuracy



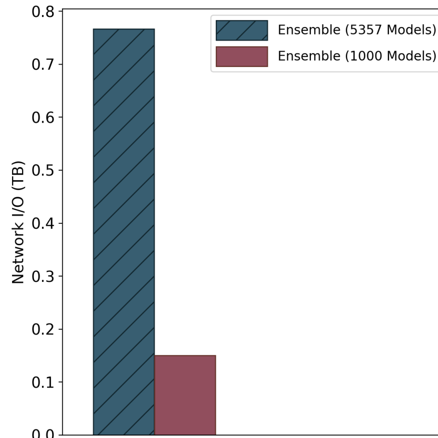
(a) Linear Regression

(b) Decision Tree

Figure 6: Mean squared error (MSE) evaluation of the predictions made by the global models and ensembles using linear regression and decision trees. Note the difference in scales; the models built with decision trees achieve higher prediction accuracy (lower MSE) for all three approaches.



(a) Disk I/O: Gradient Boosting



(b) Network I/O: Gradient Boosting

Figure 7: Disk and Network I/O with the gradient boosting ensembles.

in terms of MSE, both results are better than the global models and ensembles built with linear regression and decision trees. This result emphasizes one of the main advantages of our methodology: when a serial or difficult-to-parallelize machine learning algorithm offers the best performance for a particular dataset, it can still be trained in parallel and leveraged by our framework.

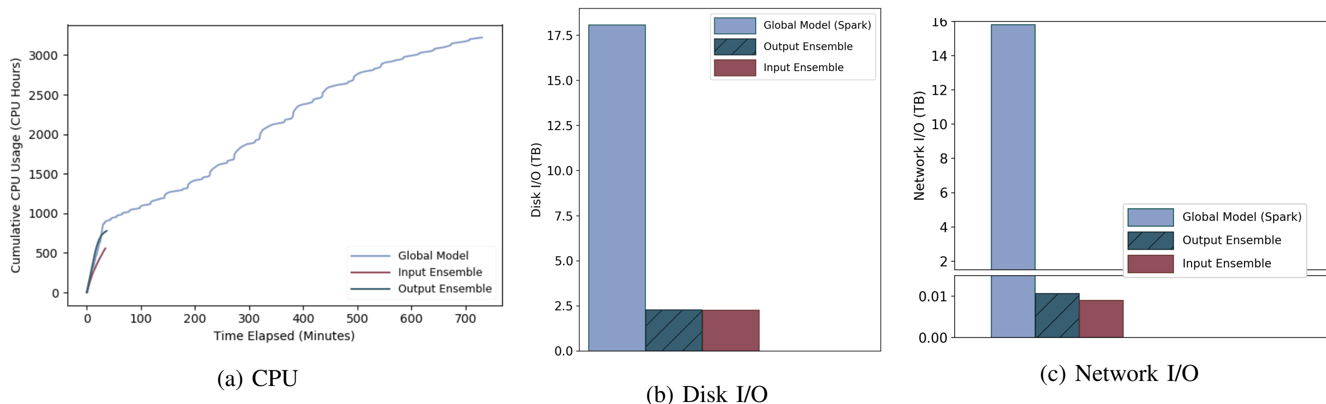
Figure 7 demonstrates the disk and network I/O of the gradient boosting ensembles. Both results closely mirror the performance observed for linear regression and decision tree ensembles. On the other hand, ensembles that leveraged gradient boosting had a higher upper bound for memory usage and used more memory on average, as shown in Table 7.

TABLE 7: Memory usage exhibited while training gradient boosting models.

Configuration	Max. Memory (GB)	Avg. Mem. (GB)
Ensemble (1000 Models)	1814.62	1153.43
Ensemble (5357 Models)	2058.20	1425.17

4.9. Systems Implications of Ensemble Sizes

As noted previously, we chose the best possible models and meta models for Spark and *Concerto*. Specifically, we chose ensembles of 1,000 and 5,357 models to balance training speed and prediction accuracy. While higher prediction accuracy with our approach is certainly possible, it comes at the cost of timeliness and rapidly approaches the point of diminishing returns. However, from a systems perspective, the number of models per ensemble does not have a significant impact on performance metrics; given a particular number of features, labels, dataset size, and model configuration, the disk I/O, network I/O, and CPU usage will all exhibit behavior similar to our previous benchmarks. Figure 8 demonstrates this by repeating our previous benchmarks using 1,000-model input and output ensembles with decision tree regression (linear regression models produced similar results as well, but are omitted for brevity).



(a) CPU

(b) Disk I/O

(c) Network I/O

Figure 8: CPU, Disk I/O, and Network I/O benchmarks re-run with 1000-model decision tree input and output ensembles.

5. Conclusions and Future Work

Our methodology, Concerto, merges the predictive power of ensembles and meta models generated via stacking while relying on data partitioning methods to maximize the insights that can be gained in a timely fashion.

While partitioning is used by several algorithms to learn from data, our methodology exploits these methods to divide the problem into smaller pieces that are sufficiently simple to learn from quickly (**RQ1**). By relying on independent models using manageable subsets of the data, our methodology allows leveraging and combining desirable algorithms to exploit their individual strengths for solving a particular problem in parallel (**RQ3**). We compared our methodology with a well-known distributed computation framework and were able to build (1) ensembles with higher predictive accuracy in many cases than a single distributed model, and (2) ensembles at scales that are not feasible with a single model, even with a machine learning algorithm that does not parallelize well (**RQ2**).

Our future work will focus on developing new partitioning methods that are optimized for domain-specific data types and adding support for GPU-based training. To make our work more broadly applicable, we also plan on adding support for lightweight distributed computations.

References

[1] A. Lebre, Y. Denneulin, G. Huard, and P. Sowa, “Adaptive i/o scheduling for distributed multi-applications environments.” in *HPDC*, 2006, pp. 343–344.

[2] H. Wang, R. Zhu, and P. Ma, “Optimal subsampling for large sample logistic regression,” *Journal of the American Statistical Association*, no. just-accepted, 2017.

[3] R. Zhu, “Gradient-based sampling: An adaptive importance sampling for least-squares,” in *Advances in Neural Information Processing Systems*, 2016, pp. 406–414.

[4] L. Han, T. Yang, and T. Zhang, “Local uncertainty sampling for large-scale multi-class logistic regression,” *arXiv preprint arXiv:1604.08098*, 2016.

[5] T. Borovicka, M. Jirina, P. Kordik, and M. Jiri, “Selecting representative data sets,” in *Advances in Data Mining Knowledge Discovery and Applications*. InTech, Sep. 2012.

[6] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford, “A reliable effective terascale linear learning system,” *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1111–1133, 2014.

[7] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, “Scaling distributed machine learning with the parameter server.” in *OSDI*, vol. 1, no. 10.4, 2014, p. 3.

[8] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, and K. Li, “A parallel random forest algorithm for big data in a spark cloud computing environment,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 919–933, 2017.

[9] Y. Zhang, Q. Gao, L. Gao, and C. Wang, “imapreduce: A distributed computing framework for iterative computation,” *Journal of Grid Computing*, vol. 10, no. 1, pp. 47–68, 2012.

[10] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing,” in *Proceedings of USENIX NSDI*. USENIX Association, 2012, pp. 2–2.

[11] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, “Pregel: a system for large-scale graph processing,” in *ACM SIGMOD*. ACM, 2010, pp. 135–146.

[12] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, “Tensorflow: A system for large-scale machine learning,” in *12th USENIXsymposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[13] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi, “Naiad: a timely dataflow system,” in *SOSP*. ACM, 2013, pp. 439–455.

[14] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu, “Petuum: a new platform for distributed machine learning on big data,” *Big Data, IEEE Transactions on*, vol. 1, no. 2, pp. 49–67, 2015.

[15] P. Indyk and R. Motwani, “Approximate nearest neighbors: Towards removing the curse of dimensionality,” in *Proc. of the 30th Annual ACM Symposium on Theory of Computing*. New York, NY, USA: Association for Computing Machinery, 1998, p. 604–613.

[16] H. Koga, T. Ishibashi, and T. Watanabe, “Fast hierarchical clustering algorithm using locality-sensitive hashing,” in *Discovery Science*, E. Suzuki and S. Arikawa, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 114–128.

[17] B. Leibe, K. Mikolajczyk, and B. Schiele, “Efficient clustering and matching for object class recognition.” in *BMVC*, 2006, pp. 789–798.

[18] P. Berkhin, *A Survey of Clustering Data Mining Techniques*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 25–71.

[19] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A. Y. Zomaya, S. Foufou, and A. Bouras, “A survey of clustering algorithms for big data: Taxonomy and empirical analysis,” *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 3, pp. 267–279, 2014.

[20] M. Steinbach, L. Ertöz, and V. Kumar, “The challenges of clustering high dimensional data,” in *New directions in statistical physics*. Springer, 2004, pp. 273–309.

[21] M. L. Bermingham, R. Pong-Wong, A. Spiliopoulou, C. Hayward, I. Rudan, H. Campbell, A. F. Wright, J. F. Wilson, F. Agakov, P. Navarro *et al.*, “Application of high-dimensional feature selection: evaluation for genomic prediction in man,” *Scientific reports*, vol. 5, p. 10312, 2015.

[22] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[23] ——, “Induction of decision trees,” *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986. [Online]. Available: https://doi.org/10.1007/BF00116251

[24] S. Kullback and R. A. Leibler, “On information and sufficiency,” *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, Mar. 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729694

[25] D. H. Wolpert, “Stacked generalization,” *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[26] S. Džeroski and B. Ženko, “Is combining classifiers with stacking better than selecting the best one?” *Machine learning*, vol. 54, no. 3, pp. 255–273, 2004.

[27] D. H. Wolpert, *The mathematics of generalization*. CRC Press, 2018.

[28] Z.-H. Zhou and J. Feng, “Deep forest: Towards an alternative to deep neural networks,” *arXiv preprint arXiv:1702.08835*, 2017.

[29] National Oceanic and Atmospheric Administration. (2016) The north american mesoscale forecast system. [Online]. Available: http://www.emc.ncep.noaa.gov/index.php?branch=NAM