CoBell: Runtime Prediction for Distributed Dataflow Jobs in Shared Clusters

Ilya Verbitskiy, Lauritz Thamsen, Thomas Renner, and Odej Kao
Technische Universität Berlin, Germany
{i.verbitskiy, lauritz.thamsen, thomas.renner, odej.kao}@tu-berlin.de

***Abstract*—Distributed dataflow systems have been developed to help users analyze and process large datasets. While they make it easier for users to develop massively-parallel programs, users still have to choose the amount of resources for the execution of their jobs. Yet, users do not necessarily understand workload and system dynamics, while they often have constraints like runtime targets and budgets. Addressing this problem, systems have been developed that automatically select the required amount of resources to fulfill the users' constraints. However, interference with co-located workloads can introduce a significant variance into the runtimes of jobs and make accurate runtime prediction harder.**

This paper presents *CoBell*, a resource allocation system that incorporates information about co-located workloads to improve the runtime prediction for jobs in shared clusters. CoBell receives jobs from users with runtime and scale-out constraints and then reserves resources based on predicted runtimes. We implemented CoBell as a job submission tool for YARN. As such, it works with existing YARN cluster setups. The paper evaluates CoBell using five different distributed dataflow jobs, showing that using CoBell results in runtimes that do not violate the runtime constraints by more than 7.2%.

***Index Terms*—Scalable Data Analytics, Distributed Dataflows, Runtime Prediction, Resource Allocation, Cluster Management**

I. Introduction

Analyzing datasets to discover relevant information is an important task for both research and industry. Examples of such applications include environmental modeling by analyzing sensor networks, energy saving by analyzing usage patterns, and evaluation of online courses to improve education [1]. However, the increasing size of the datasets complicates this task. Distributed systems have been developed to address this problem. These systems handle large-scale datasets by sharing data and computations among a set of nodes.

In particular, distributed file systems like Google File System (GFS) [2] and Hadoop Distributed File System (HDFS) [3] help users with the storage of large datasets that will not fit on a single computer. These system split files into chunks and replicate them across all available nodes. This way, many users can share one storage and files can be read in parallel. Distributed dataflow systems like Spark [4] and Flink [5] help users in processing large datasets. The systems provide a framework that lets users develop massively-parallel applications from sequential building blocks. In these systems, users express a distributed application in terms of operators known from functional programming like *map* and *reduce*. The frameworks then take care of parallelizing, distributing, and scheduling the application. Resource managers like Mesos [6] and YARN [7] allow multiple users and their jobs to share a single cluster. In these systems, cluster resources are represented by the notion of containers. A container represents resources on a single node, for example an amount of RAM and CPU cores. To run a job, users request containers from the resource manager. With the reserved containers, users can execute a distributed job using their preferred dataflow framework.

However, users often lack the understanding of system and job dynamics to accurately choose the required amount of resources for their needs [8, 9, 10]. In particular, they tend to defensively over-provision, especially for their business-critical jobs [11]. At the same time, users can often easily express their needs in terms of budget and runtime constraints [12]. For example, business-critical jobs with agreed upon Service Level Objectives (SLOs) induce a high priority on fulfilling runtime targets to prevent costly violations.

Statically inferring the runtime, though, is difficult. A job's runtime might be influenced by many factors like hardware configurations, program parameters, dataset characteristics, and data locality [13, 14, 15]. In addition, if multiple jobs are scheduled onto the same physical machines to improve the overall utilization, interference between the jobs leads to varying job runtimes. The effect is stronger the more co-located jobs exhibit similar resource usage characteristics [16]. As a result, several systems have been developed to assist users in selecting the right amount of resources by, for example, analyzing previous job executions [17, 9, 18] or dedicated profiling runs [8, 12]. Some systems also monitor the current job execution and, if necessary, adjust the resource allocations [19, 11, 10]. However, while systems like Ellis [18] and Ernest [12] learn a model that predicts the runtime of jobs given their scale-out, they do not incorporate any information about concurrently running jobs. Yet, interference between co-located jobs can influence the runtime significantly.

This paper presents *CoBell*, a runtime prediction system that uses information about concurrently running jobs to improve predictions. For this, our system models the runtime of recurring job combinations. Specifically, it uses a model that predicts how the runtime of a job changes given the interfering job and the interference duration. We implemented CoBell as a job submission tool that accepts jobs along with runtime and scale-out constraints. It then uses our runtime model to select the amount of resources that satisfy the constraints. Our implementation performs the resource reservation and actual job submission using YARN. As such, it works with existing YARN cluster setups. What is more, our system requires only basic information about previous job runs in order to build our runtime model. By doing so, we follow a black-box approach, so CoBell can be used with various dataflow framework.

Contributions: The contributions of this paper are as follows:

- An approach to improve the accuracy of runtime predictions for interfered workloads by incorporating knowledge about co-located jobs.
- An implementation of the CoBell prototype as a job submission tool for YARN.
- An evaluation of our implementation using five different distributed dataflow jobs and different co-location configurations.

Outline: The remainder of the paper is structured as follows. Section II provides background on distributed dataflow systems. Section III presents the CoBell system and its models. Section IV visualizes CoBell's models and evaluates the quality of resource allocations. Section V presents the related work. Section VI concludes this paper.

II. Background

This section provides the background on distributed data analysis systems. In particular, it describes how large datasets can be stored and processed on a set of nodes and how such a cluster can be shared between users.

A. Distributed File Systems

Distributed files systems typically work by splitting large files into multiple smaller blocks. The blocks are then distributed among a set of machines in a cluster. In addition, each block is redundantly stored on a few other nodes. This can prevent data corruption in case a few data nodes fail.

When users want to access the data from a distributed file system, they must obtain all the needed data blocks. If a file block is not available on the local machine, the block must be transferred over the network. However, to obtain optimal performance data access should be coordinated in way such that the data is read locally as much as possible. This is known as data-locality [15].

B. Distributed Dataflow Systems

Distributed dataflow systems typically let user describe their application in terms of a directed acyclic graph (DAG). In such a dataflow graph each vertex describes an operator that processes incoming data and outputs the result of a computation. The edges between the vertices define the dataflows. The vertices execute second-order functions provided by the framework like *map* and *reduce*. The user supplies the second-order functions with arbitrary user-defined functions (UDFs).

After a user defined his job, he submits it to the framework scheduler. The framework automatically parallelizes, distributes, and schedules the application. For this, the framework transforms the job graph into a task graph where each task vertex is a parallel instance of a job vertex. The degree of parallelism (DoP) of a job vertex can either be set by the user or left open for the framework to decide.

C. Resource Management Systems

Resource management systems allow operators of a cluster to share the computing resources to multiple users. For this, the systems typically divide nodes in a cluster into smaller management units called containers. A container represents an amount of resources like a combination of CPU cores and the amount of RAM. Users can submit their applications by reserving an amount of containers. The resource manager then takes care of finding and assigning free resources as well as starting the application.

Besides simplifying the usage for multiple users, resource managers enable the use of multiple dataflow frameworks on the shared cluster. What is more, resource managers let users run their dataflow framework on a per-job basis.

III. CoBell

The section starts with an overview of the CoBell system and its core elements. Afterwards, the runtime models are presented. Finally, the resource allocation procedure is explained.

A. System Overview

Figure 1 visualizes the system components. CoBell accepts job submissions from users alongside their runtime and scale-out constraints. It then translates the runtime target into a resource reservation. This is done by keeping track of previously executed jobs. When a new job is submitted, CoBell queries its workload database for previous runs of this job with a similar input data size. These runs are then used for building the prediction model.

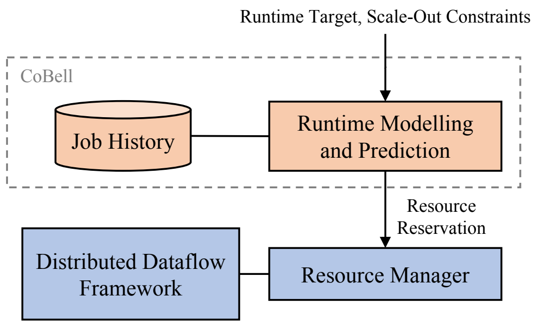

Runtime Target, Scale-Out Constraints
CoBell
Job History
Runtime Modelling and Prediction
Resource Reservation
Distributed Dataflow Framework
Resource Manager

Fig. 1. The CoBell prediction system.

After translating the constraints into a resource reservation, CoBell uses the respective submission client of the dataflow frameworks to start the submitted job. The clients are configured such that the jobs are submitted using YARN to run on a per-job basis. This way, CoBell does not rely on any control over the framework itself. What is more, it is independent of the dataflow framework as long as it is capable of running on YARN. In addition, CoBell only requires basic information about job executions. The information consist of scale-out and the runtime of the running jobs. That is, no framework-specific performance counters are required. As such, CoBell follows a black-box approach that enables its deployment for a wide range of frameworks.

B. Runtime Prediction Model

CoBell's models build upon our previous work Bell [17]. The Bell prediction system provides two ways of modeling the runtime behavior. The first approach uses a parametric model that is robust and enables interpolation and extrapolation. The parametric model is based on the approach from Ernest [12]. The runtime behavior is modeled by

$$f(x) = \theta_0 + \theta_1 \cdot \frac{1}{x} + \theta_2 \cdot \log(x) + \theta_3 \cdot x$$

where x is a scale-out and $f(x)$ the predicted runtime. Every term of the function corresponds to a specific data exchange pattern that depends on the amount of nodes. The first term is the amount of serial computation. The second term captures computation that can be parallelized with the addition of machines. The third term captures tree-like communication patterns. The last term represents collect-like communications. Sometimes, the parametric model is not able to capture the runtime behavior precisely. In such cases, local linear regression (LLR) is used to model the runtime behavior. Such nonparametric models allow to capture arbitrary relations by fitting to the data.

CoBell's runtime model incorporates the knowledge about co-located jobs to improve the runtime prediction. Like Bell, CoBell provides two models. A robust parametric model is capable of performing extrapolation. A more flexible nonparametric model provides an alternative for interpolation and dense training data. When performing interpolation, cross-validation (CV) can be used to select between the two models.

1) Parametric Model: The idea of CoBell's parametric model is that the relative duration of an interference and a resulting relative change in runtime are proportional. This means that we can model the relative runtime change rc' as

$$\mathrm{rc}'(\mathrm{ov}) = \alpha \cdot \mathrm{ov} \tag{1}$$

where α is the slope of the linear relation and ov is the overlapping ratio. Here, overlapping ratio is the duration of job interference, relative to the job's runtime. For example, if a job suffers from interference over its whole runtime, then $\mathrm{ov} = 1$. The parameter α essentially describes how strong the interference effect is between two jobs. For example, a value of $\alpha = 1$ would mean that if a job is interfered over its full duration, its runtime will double. On the other hand, a value of $\alpha = 0$ suggest that the co-location has no negative effects on the job. This might happen, for example, if the co-located jobs have completely orthogonal resource usages.

Note that Equation 1 does not include an intercept parameter. The reasoning is that without any overlapping, i.e., $\mathrm{ov} = 0$, there should be no change in runtime. What is more, the parameter α is constrained to be positive. The idea here is that interference can only increase the runtime and not speed the execution up.

However, note that the parameter α in Equation 1 does not depend on the scale-out. This would imply that the change in runtime due to overlapping is always the same and independent of the scale-out. To relax this assumption a little bit, the runtime change model is extended with a slope that depends on the scale-out. The resulting *runtime change model* is

$$\mathrm{rc}(\mathrm{ov}, x) = \alpha(x) \cdot \mathrm{ov} = (a + b \cdot \frac{1}{x}) \cdot \mathrm{ov} \tag{2}$$

where ov is the overlapping ratio, x is the scale-out, and $a, b \geq 0$ are the parameters. The motivation leading to this particular α model is that interference impacts are higher when using only a few nodes as the framework has fewer possibilities to distribute the computational load.

Combining this model with Bell's parametric model leads to the *extended runtime model*

$$\begin{aligned} g(x, \mathrm{ov}) &= f(x) \cdot \big(1 + \mathrm{rc}(\mathrm{ov}, x)\big) \\ &= \big(\theta_0 + \theta_1 \cdot \frac{1}{x} + \theta_2 \cdot \log(x) + \theta_3 \cdot x\big) \\ &\quad \cdot \big(1 + (a + b \cdot \frac{1}{x}) \cdot \mathrm{ov}\big) \end{aligned}$$

that now integrates the overlapping ratio ov.

2) Nonparametric Model: CoBell's nonparametric model follows the same idea as the parametric model. The extended nonparametric runtime model is the combination of Bell's nonparametric model and Equation 2. However, instead of learning the function in a single pass, the extended nonparametric model works in two phases. In the first step, the job history data $((x_i, \mathrm{ov}_i, y_i))_{i=1}^{N}$ is fit using the extended parametric model. This way the parameters a, b for the runtime change model rc are obtained. The parameters are then used to normalize the training data with respect to the overlapping. That is, each runtime y_i is transformed to

$$\hat{y}_i = \frac{y_i}{1 + \mathrm{rc}(\mathrm{ov}_i, x_i)} \,.$$

The regular nonparametric regression from Bell is then used to fit the dataset $((x_i, \hat{y}_i))_{i=1}^{N}$.

Evaluating the nonparametric function proceeds in a similar manner. That is, the prediction for scale-out x and overlapping ratio ov equals to

$$g(x, \mathrm{ov}) = f(x) \cdot \big(1 + \mathrm{rc}(\mathrm{ov}, \mathrm{x})\big) \,,$$

where f is the learned nonparametric model.

C. Resource Allocation

Before the extended model can be used for prediction, the overlapping must be known. We assume that the job to run will have a runtime roughly equal to its runtime target. Based on this information we can calculate the overlapping between the job to run and an already running job. However, this overlapping leads to a longer runtime of the already running job. This in turn increases the overlapping. By repeating this computation iteratively, it will finally converge and return the expected overlapping. Algorithm 1 summarizes the procedure. The PREDICTOVERLAPPING function takes a job J and its

Algorithm 1 Prediction of the overlapping ratio.

```
function OVERLAPDURATION($t_1$, $t_2$, $d$)
    if $t_2 > t_1$ then
        return $0$
    if $t2 + d < t_1$ then
        return $d$
    return $t_1 - t_2$

function RUNTIME($r$, $f$, $x$, $t_{\text{start}}$, $t_{\text{now}}$, $C$)
    $t_{\text{end}} \leftarrow t_{\text{start}} + r$
    $d' \leftarrow$ OVERLAPDURATION($t_{\text{end}}$, $t_{\text{now}}$, $C$)
    $r' \leftarrow f(x, \frac{d'}{r})$
    return $r'$

function PREDICTOVERLAPPING($J$, $C$)
    $x \leftarrow$ scale-out of the currently running job
    $t_{\text{start}} \leftarrow$ start timestamp of the currently running job
    $t_{\text{now}} \leftarrow$ current timestamp
    $f \leftarrow$ fit CoBell model for the currently running job and the submitted job $J$
    $r_0 \leftarrow f(x, 0)$
    Repeat $r_{n+1} \leftarrow$ RUNTIME($r_n$, $f$, $x$, $t_{\text{start}}$, $t_{\text{now}}$, $C$) until convergence
    $r^* \leftarrow$ the converged fix-point with $r^* \approx$ RUNTIME($r^*$, $f$, $x$, $t_{\text{start}}$, $t_{\text{now}}$, $C$)
    return $\frac{r^*}{C}$
```

runtime constraint C as input and computes the overlapping ratio $\frac{r^*}{C}$ using fix-point iterations.

Having obtained the overlapping ratio, CoBell can finally use the extended models to predict the runtime. After predicting the runtimes, CoBell selects the smallest possible scale-out that fulfills the runtime constraint. This completes the resource allocation sequence.

IV. EVALUATION

This section starts with a description of our CoBell implementation and the test workload that were used for conducting the experiments. Further, it presents an evaluation of the system's data fitting performance and resource allocation quality.

A. Implementation

The implementation is based on the pseudocode from Section III-C and is implemented in Python. SciPy's[1] `curve_fit` function is used to solve the bounded, non-linear optimization problem of CoBell's extended runtime model. The function is configured such that a trust-region approach is used to solve the problem which is based on the work from Branch et al. [20].

B. Experimental Setup

The cluster consists of 26 machines. Each computer in the cluster is equipped with an Intel Xeon X3450 @2.67GHz CPU (4 physical cores, 8 hardware contexts) and 16 GB of RAM. Each node runs Linux (kernel version 3.10.0), Java 1.8.0, HDFS 2.7.2, and YARN 2.7.2. Flink 1.3.2 is used as the dataflow framework and is configured to submit jobs via YARN.

[1]SciPy, https://www.scipy.org/ [accessed 2017-12-28]

C. Test Workload

This section provides details about the test workload. The jobs and the respective input data sets are summarized in Table I. The following subsections provide more detailed descriptions.

TABLE I
OVERVIEW OF THE BENCHMARK JOBS

Job	Dataset	Input size	Parameters
PageRank	Graph	3.3 GB	10 iterations
Word Count	Wiki	244.7 GB	–
CC	Twitter	24.4 GB	–
SGD	Features	107.8 GB	100 iterations, step size = 1.0
K-Means	Points	1.8 GB	30 clusters, 10 iterations

1) Jobs: Five jobs were selected for benchmarking such that they cover different domains like machine learning, text processing, and graph analysis. All jobs are implemented using the Flink framework. The implementations are based on libraries and examples provided by Flink.

Connected Components (CC): A connected component of an undirected graph is a subgraph such that for every two vertices in this subgraph there exists a path. This job takes a graph as input and outputs the connected component sizes.

K-Means: K-means clustering is a method that subdivides given observations or data points into a set of k clusters. The cluster are selected such that the sum of squared distances of observations to the nearest cluster mean is minimized. Lloyd's algorithms [21] is used for this job to heuristically solve the problem.

Stochastic Gradient Descent (SGD): In machine learning one typical objective function has the form $Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w)$ where w is a weight or parameter vector and the function Q_i depends on the i-th observation. One example that has such objective function is the least squares problem. SGD is a optimization technique that iteratively updates the weight vector w using a single random data point per iteration. While named stochastic gradient descent, Flink's implementation actually performs batch gradient descent[2].

PageRank: PageRank [22] is a method to rank web pages according to their link structure. The idea behind the algorithm is that the higher the PageRank of the linking pages the higher is the PageRank of the linked page.

Word Count: The Word Count job processes a text-based input dataset and outputs the frequency of every unique word.

[2]For more details refer to https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/libs/ml/optimization.html#stochastic-gradient-descent [accessed 11-12-2017].

2) Datasets: Five different datasets were used for benchmarking. Four of them were generated synthetically. What follows is a more detailed description of the datasets.

Graph: The Graph dataset was generated using the Big Data Generator Suite (BDGS) [23]. It uses the Kronecker graph model to generate a graph based on some initial dataset. In this case a Google Web graph is used initially, a graph that represents the link structure for a set of web pages. By applying 25 Kronecker iterations, a 3.4 GB large graph with 33,554,432 nodes and 213,614,240 directed edges was created.

Wiki: The Wiki dataset was also generated using the BDGS. For this, BDGS trains a LDA model from a real dataset of documents from Wikipedia. Then, the trained model is used to synthetically generate 244.7 GB data.

Twitter: The Twitter dataset is a graph that contains user-follower relations from the Twitter site. It was obtained by crawling 41.7 million user profiles on Twitter [24] resulting in 24.4 GB of data.

Features: The Features dataset was generated by creating a random weight vector of size 600. Random vectors of the same size were created and multiplied by the weight vector. Finally, Gaussian noise was added to the resulting points. The procedure was repeated to generate 100,000,000 points resulting in 107.8 GB of data.

Points: The Points dataset was generated by sampling 10,000,000 two-dimensional points from a Gaussian mixture model resulting in 1.8 GB of data. The mixture model consists of 30 normal distributions with randomly generated centers and a uniform prior. The normal distributions have identical and diagonal covariance matrices.

D. Experiments

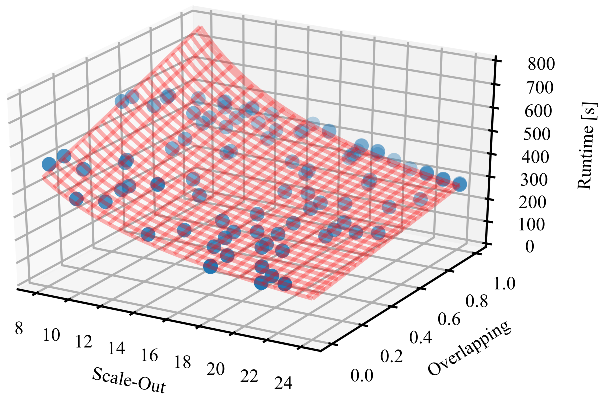
Two experiments were executed. The aim of the first experiment was to obtain training data in order to visualize CoBell's parametric model and then show a comparison to Bell. Bell is the baseline prediction system that forms the basis of our system, yet does not take job co-location into account. For the first experiment, 100 random runs of the job combination SGD and Word Count. The delay between the two jobs was selected randomly. The scale-out of the interfering Word Count job was set to 24. The scale-out of the SGD job was chosen randomly.

The second experiment evaluated the resource allocation of the CoBell system. For this, we selected the two jobs CC and SGD. The runtime target for both jobs was set to 350 seconds. Before a job is submitted for resource allocation, another job is started to provide the interference. This first job is one of the remaining four jobs. After a random delay, the second job is submitted and the CoBell system performs the allocation to fulfill the provided runtime target. With the chosen job the experiment executes 20 runs for each possible job combination starting with an empty job history. This results in 80 runs for each of the two jobs that are subject to the resource allocation procedure.

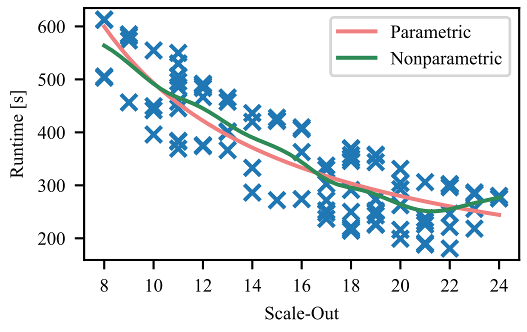
E. Results

The results of the experiments are presented in the following. The first subsection visualizes how CoBell's parametric runtime model fits the data and show a comparison to Bell. The performance of the resource allocation is presented in the second subsection.

1) Runtime Model: Figure 2a visualizes both, the data obtained from the experiment and CoBell's parametric model. As expected, the runtime of a job depends on both, the scale-out and the overlapping ratio with an already running job. For comparison, Figure 2b shows how Bell would interpret this data. Clearly, Bell does not consider interference in its model. As a result, Bell cannot explain the high variation of the data points and interprets it as noise. This is in contrast to CoBell's model. Its model considers the overlapping ratio of interfering jobs and, as a result, the data shows a much lower variation around the fitted curve.



(a) CoBell's parametric model. The blue circles represent the different executions. The red mesh visualizes CoBell's parametric model fit to the data.



(b) Bell's models. The blue crosses represent the different runs by excluding the "Overlapping" dimension.

Fig. 2. Job execution data from the SGD job with Word Count as the interfering job along with models fitted to this data.

The three error metrics root mean square error (RMSE), mean absolute error (MAE), and mean absolute percentage error (MAPE) are chosen to quantify the performance of CoBell's and Bell's models for this particular job combination.

They are defined as

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} |\hat{y}_i - y_i|^2}\ ,$$

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N} |\hat{y}_i - y_i|\ , \text{ and}$$

$$\text{MAPE} = \frac{100}{N}\sum_{i=1}^{N} \left|\frac{\hat{y}_i - y_i}{y_i}\right|\ ,$$

respectively. The $\hat{y}_i$ is the predicted and y_i the true runtime of the i-th input sample. Table II summarizes the in-sample errors.

TABLE II
IN-SAMPLE MODEL PERFORMANCE

Model	RMSE	MAE	MAPE
CoBell parametric	12.07	8.74	2.54
CoBell nonparametric	12.00	8.75	2.47
Bell parametric	56.33	50.72	15.69
Bell nonparametric	53.11	47.13	14.41

The error metrics show significant improvement by using CoBell's prediction. For example, the MAPE is roughly six times smaller with CoBell's parametric model compared to Bell's counterpart. Note that CoBell's nonparametric model does not outperform the parametric model. This might be because the parametric model already captures the scale-out behavior of this specific job accurately enough.

2) Resource Allocation: The results of the resource allocation are presented in Figure 3 for the SGD and CC job, respectively. The interfering job is annotated row-wise. Since CoBell assumes recurring jobs, a job history is required in order to learn the model. However, when conducting the experiments no such data is available. Therefore, CoBell resorts to a simple heuristic for the first six runs. The initial three runs use a binary search inspired approach for selecting the scale-out. The runs four to six use Bell's parametric model to perform the prediction.

After the first six runs, CoBell's parametric model is used. As a result, the runtime closely follows the runtime target for both jobs. That is, CoBell is not only able to follow the runtime target with different overlapping ratios but also for different interfering jobs. None of the runs after bootstrapping exceeded the constraint by more than 7.2 percent.

Figure 4 shows an example of the changes of overlapping ratios between the runs and how CoBell adjusts the scale-out to fulfill the runtime target. Note how the scale-out increases and decreases with the overlapping ratios. For example, from run 8 to run 9 the SGD job changes from full overlapping to almost no overlapping. As a result, the scale-out decreases from 19 to 11 nodes. However, this is not the case for the first six runs since the bootstrap procedure does not consider overlapping.

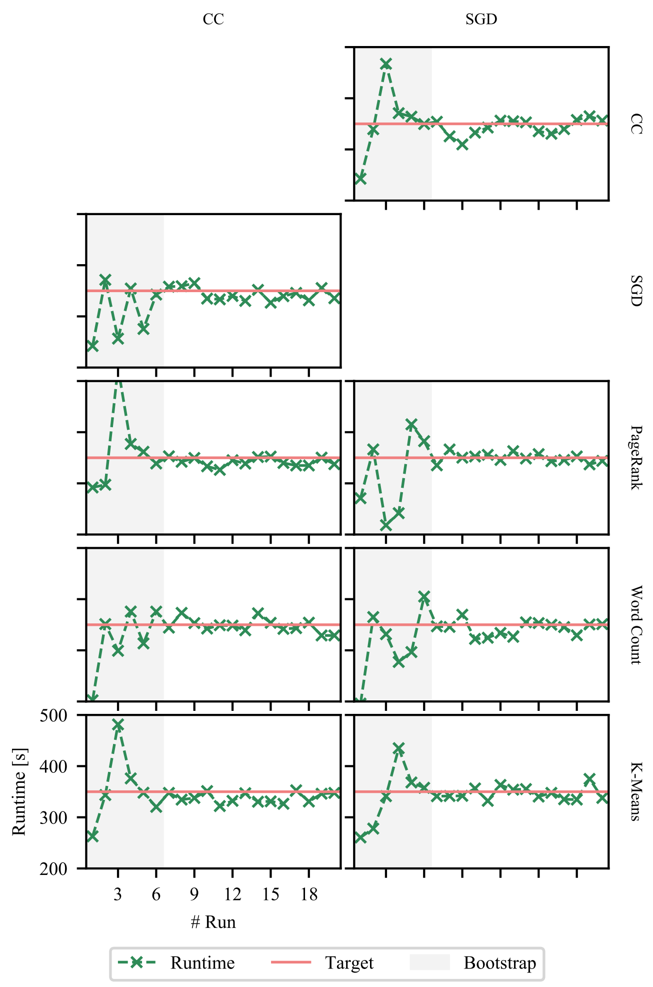


Fig. 3. Runtime of the SGD and CC job using CoBell with a runtime target of 350 seconds.

V. RELATED WORK

A number of resource prediction and allocation systems use a white-box model that is tailored to a specific dataflow framework. Examples of such systems include Aria [25, 26], Elastisizer [27], AROMA [8], and Bazaar [28] which are all specific to Hadoop's MapReduce.

Aria creates job profiles using past executions or sample runs. The job profiles are then supplied to a MapReduce performance model along with the amount of input data and a target runtime. The system then computes the required amount of map and reduce slots that are required to fulfill the target runtime. In addition, it estimates the impact of node failures on the completion time.

Similarly, Elastisizer uses job profiles to predict the runtime and cost of dataflow jobs. The system incorporates information about the input data, resource configuration, and job configuration to make a prediction. The profiles are generated by direct measurement. What is more, they can be extended to virtual profiles that enables performance prediction for hypothetical cluster and job configurations using simulation.

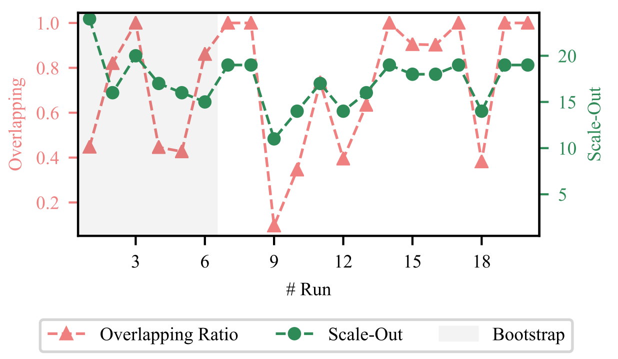


Fig. 4. Scale-out and overlapping ratio during resource allocation for the SGD job alongside the execution of the Word Count job.

AROMA uses a two-phase approach for resource allocation. In the first phase the resource utilization of finished jobs is analyzed. Jobs that exhibit similar resource usage are grouped by using a clustering algorithm. In the second online phase the actual resource allocation takes place. AROMA's performance model is based on SVM regression accompanied by a systematic feature selection and is used to perform the runtime prediction The resources are then selected in way such that the runtime target is fulfilled and the cost is minimized.

Bazaar predicts the runtime of a job as a function of the amount of nodes, the available network bandwidth, and the input data size. To obtain the model parameters, it performs dedicated sample runs on a subset of the input data. To cope with outliers and failures, Bazaar relies on additional slack.

Jockey [19] is a prediction system for SCOPE. It consists of three components. A job simulator simulates the job runtime given its progress and the allocated resources. A progress estimator is responsible for capturing the state of the job. Finally, a control loop monitors the job's executions and adjusts the resources, if necessary.

In contrast to these systems, CoBell follows a black-box approach. As such, it requires only very generic information about job executions like the runtime, the scale-out, and the concurrently running jobs. This makes CoBell applicable to a wide range of different dataflow frameworks.

Black-box prediction systems, on the other hand, support multiple different dataflow systems. Quasar [10] is such a system that jointly performs resource allocation and assignment. It uses classification techniques to determine the amount of resources, type of resource, and interference impact on a job's runtime. For training, Quasar uses a combination of profile runs and previously scheduled jobs. What is more, job execution is monitored and the resources are adjusted in case of deviation from the expected runtime target. CoBell, on the other hand, does not assume control over resource assignment, but instead relies on existing resource managers. In addition, our system does not require the job execution to be dynamically scalable. Furthermore, CoBell is based entirely on previous job executions and does not depend on dedicated profile runs.

Ernest [12] models scale-out behavior using parametric regression. To train the model, Ernest executes incoming jobs on a small set of dedicated machines and a subset of the input data. The system then extrapolates the runtimes for the actual amount of input data and all possible scale-outs. The trained model is then used to select the resources such that a runtime target is fulfilled. In contrast, CoBell does not depend upon profile runs, but instead uses previous executions for training. While our system's parametric model is based on Ernest's model, it also provides a flexible nonparametric model. Finally, Ernest does not take job interference into account for the prediction.

Similarly, Bell [17] models a function that predicts the runtime of a job given its scale-out. However, Bell trains the model using previous executions of the recurring job. The systems uses two models, a robust parametric model for inter- and extrapolation and a flexible nonparametric model for interpolation only. For interpolation, Bell select between the two models using CV. CoBell builds upon Bell and incorporates interference between co-located jobs into models.

PerfOrator [9] predicts runtime performance of queries using various profiles. To begin with, a hardware profile is computed that summarizes key metrics. This is used to estimate the time it takes for performing reads, writes, and data shuffles during computation. What is more, a framework parallelization profile is used to analyze the effectiveness of parallelization optimizations. PerfOrator starts with an incoming query. The query is the profiled on an input sample. The previous profiles and non-linear regression are then used to predict the runtime and transform performance requirements to resource allocations. PerfOrator requires framework-specific white-box models to accurately perform performance prediction. This is contrast to the black-box approach of CoBell. What is more, our system is based on previous runs and does not require profile runs.

CherryPick [29] is another black-box allocation system. It selects the most cost-efficient cluster configuration subject to a given runtime constraint. The system uses the Bayesian optimization framework along with a Gaussian process prior to search for an optimal cloud configuration using sample runs. The resulting model is nonparametric and, thus, does not expect a specific performance model. Compared to our system, CherryPick depends on sample runs to build its performance model. Moreover, CherryPick does not include any information about co-located jobs to provide runtime predictions in the face of interferences.

VI. Conclusion

This paper introduced CoBell, a resource allocation system that incorporates knowledge about interference with co-located jobs. At its core, CoBell trains separate models for different job combination and considers the interference durations of jobs for runtime prediction. The paper further showed how the models fit the exemplary data obtained from co-located job executions and compared it to models that do not consider interference information.

Our evaluation showed that CoBell's models can explain the runtime significantly better than models that do not integrate job interference. We further evaluated the resource allocation of CoBell using five different jobs. The resulting runtimes were within 7.2% of the target runtime for all resource allocations.

In the future, we want to relax some of the assumptions used for CoBell. In particular, CoBell should be able to deal with interference from an arbitrary amount of different jobs. Nonetheless, CoBell already shows that incorporating knowledge about concurrently running jobs can improve the runtime prediction in shared clusters.

References

[1] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi, "Big Data and Its Technical Challenges," *Commun. ACM*, vol. 57, no. 7, pp. 86–94, Jul. 2014.

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. ACM, October 2003, pp. 29–43.

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, May 2010, pp. 1–10.

[4] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. USENIX Association, June 2010, pp. 10–10.

[5] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache Flink: Stream and Batch Processing in a Single Engine," *IEEE Data Engineering Bulletin*, vol. 38, no. 4, pp. 28–38, July 2015.

[6] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A Platform for Fine-grained Resource Sharing in the Data Center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. USENIX Association, March 2011, pp. 295–308.

[7] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet Another Resource Negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. ACM, September 2013, pp. 5:1–5:16.

[8] P. Lama and X. Zhou, "AROMA: Automated Resource Allocation and Configuration of Mapreduce Environment in the Cloud," in *Proceedings of the 9th International Conference on Autonomic Computing*, ser. ICAC '12. ACM, September 2012, pp. 63–72.

[9] K. Rajan, D. Kakadia, C. Curino, and S. Krishnan, "PerfOrator: Eloquent Performance Models for Resource Optimization," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, ser. SoCC '16. ACM, 2016, pp. 415–427.

[10] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware Cluster Management," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. ACM, March 2014, pp. 127–144.

[11] S. A. Jyothi, C. Curino, I. Menache, S. M. Narayanamurthy, A. Tumanov, J. Yaniv, R. Mavlyutov, I. n. Goiri, S. Krishnan, J. Kulkarni, and S. Rao, "Morpheus: Towards Automated SLOs for Enterprise Clusters," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. USENIX Association, November 2016, pp. 117–134.

[12] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica, "Ernest: Efficient Performance Prediction for Large-scale Advanced Analytics," in *Proceedings of the 13th Usenix Conference on Networked Systems*

[14] N. Bruno, Y. Kwon, and M.-C. Wu, "Advanced Join Strategies for Large-scale Distributed Computation," *Proc. VLDB Endow.*, vol. 7, no. 13, pp. 1484–1495, Aug. 2014.

Design and Implementation, ser. NSDI'16. USENIX Association, March 2016, pp. 363–378.

[13] M. Elseidy, A. Elguindy, A. Vitorovic, and C. Koch, "Scalable and Adaptive Online Joins," *Proc. VLDB Endow.*, vol. 7, no. 6, pp. 441–452, Feb. 2014.

[15] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving MapReduce Performance Through Data Placement in Heterogeneous Hadoop Clusters," in *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, April 2010, pp. 1–9.

[16] L. Thamsen, B. Rabier, F. Schmidt, T. Renner, and O. Kao, "Scheduling Recurring Distributed Dataflow Jobs Based on Resource Utilization and Interference." in *2017 IEEE International Congress on Big Data (BigData Congress)*. IEEE, June 2017, pp. 145–152.

[17] L. Thamsen, I. Verbitskiy, F. Schmidt, T. Renner, and O. Kao, "Selecting Resources for Distributed Dataflow Systems According to Runtime Targets," in *International Performance Computing and Communications Conference (IPCCC), 2016 IEEE 35th International Conference on*. IEEE, 2016, pp. 1–6.

[18] L. Thamsen, I. Verbitskiy, J. Beilharz, T. Renner, A. Polze, and O. Kao, "Ellis: Dynamically Scaling Distributed Dataflows to Meet Runtime Targets," in *Proceedings of the 2017 IEEE 9th International Conference on Cloud Computing Technology and Science*, ser. CloudCom 2017. IEEE, December 2017, pp. 146–153.

[19] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca, "Jockey: Guaranteed Job Latency in Data Parallel Clusters," in *Proceedings of the 7th ACM European Conference on Computer Systems*, ser. EuroSys '12. ACM, April 2012, pp. 99–112.

[20] M. A. Branch, T. F. Coleman, and Y. Li, "A Subspace, Interior, and Conjugate Gradient Method for Large-Scale Bound-Constrained Minimization Problems," *SIAM Journal on Scientific Computing*, vol. 21, no. 1, pp. 1–23, 1999.

[21] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An Efficient k-Means Clustering Algorithm: Analysis and Implementation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 881–892, Jul. 2002.

[22] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web." Stanford InfoLab, Technical Report 1999-66, November 1999.

[23] Z. Ming, C. Luo, W. Gao, R. Han, Q. Yang, L. Wang, and J. Zhan, *BDGS: A Scalable Big Data Generator Suite in Big Data Benchmarking*. Cham: Springer International Publishing, 2014, pp. 138–154.

[24] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a social network or a news media?" in *WWW '10: Proceedings of the 19th international conference on World wide web*. New York, NY, USA: ACM, 2010, pp. 591–600.

[25] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic Resource Inference and Allocation for MapReduce Environments," in *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ser. ICAC '11. ACM, June 2011, pp. 235–244.

[26] ——, "Resource Provisioning Framework for MapReduce Jobs with Performance Goals," in *Proceedings of the 12th ACM/IFIP/USENIX International Conference on Middleware*, ser. Middleware'11. Springer, December 2011, pp. 165–186.

[27] H. Herodotou, F. Dong, and S. Babu, "No One (Cluster) Size Fits All: Automatic Cluster Sizing for Data-intensive Analytics," in *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, ser. SOCC '11. ACM, October 2011, pp. 18:1–18:14.

[28] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Bridging the Tenant-provider Gap in Cloud Services," in *Proceedings of the Third ACM Symposium on Cloud Computing*, ser. SoCC '12. ACM, October 2012, pp. 10:1–10:14.

[29] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 469–482.