Thamsen, L., Renner, T., Verbitskiy, I. and Kao, O. (2018) Adaptive Resource Management for Distributed Data Analytics. In: Grandinetti, L., Mirtaheri, S. L., Shahbazian, R., Sterling, T. and Voevodin, V. (eds.) *Big Data and HPC: Ecosystem and Convergence.* Series: Advances in Parallel Computing. IOS Press: Amsterdam, pp. 155-170. ISBN 9781614998815

(doi: 10.3233/978-1-61499-882-2-155)

This is the Author Accepted Manuscript.

There may be differences between this version and the published version. You are advised to consult the publisher's version if you wish to cite from it.

http://eprints.gla.ac.uk/268140/

Deposited on: 25 May 2022

Enlighten – Research publications by members of the University of Glasgow
http://eprints.gla.ac.uk

# Adaptive Resource Management for Distributed Data Analytics

Lauritz THAMSEN, Thomas RENNER, Ilya VERBITSKIY, and Odej KAO
*Technische Universität Berlin, Germany*
e-mail: {firstname.lastname}@tu-berlin.de

**Abstract.** Increasingly large datasets make scalable and distributed data analytics necessary. Frameworks such as Spark and Flink help users in efficiently utilizing cluster resources for their data analytics jobs. It is, however, usually difficult to anticipate the runtime behavior and resource demands of these distributed data analytics jobs. Yet, many resource management decisions would benefit from such information.
Addressing this general problem, this chapter presents our vision of adaptive resource management and reviews recent work in this area. The key idea is that workloads should be monitored for trends, patterns, and recurring jobs. These monitoring statistics should be analyzed and used for a cluster resource management calibrated to the actual workload. In this chapter, we motivate and present the idea of adaptive resource management. We also introduce a general system architecture and we review specific adaptive techniques for data placement, resource allocation, and job scheduling in the context of our architecture.

**Keywords.** Scalable Data Analytics, Resource Management, Distributed Dataflows, Distributed File Systems, Performance Modeling

## 1. Introduction

Businesses and the sciences have to manage and analyze increasingly large datasets. For this, distributed systems and more specifically distributed dataflow systems are used. Prominent examples include MapReduce [1], Spark [2], Flink [3], and Dataflow [4]. These systems help users in efficiently utilizing clusters of computers for their data analytics tasks. Users assemble programs by connecting operators, selected from a set of predefined data transformations, to form dataflow graphs. Users configure these operators and provide sequential user code. Distributed dataflow systems then automatically parallelize and distribute operators across workers. Consequently, distributed instances of operators process large datasets in parallel, which are usually stored in a distributed file system such as HDFS [5]. In these systems, datasets are split into series of data blocks with multiple replicas stored on different commodity nodes. Replication and file splitting can increase the performance of distributed dataflow systems, because operators can access and process parts of the datasets in parallel stored on different nodes. Analytical clusters typically run a resource management system like YARN [6] or Mesos [7] to support multiple concurrent jobs and users. Each single job is provided with a share of the cluster. Usually a job receives a number of homogeneous containers, each rep-

resenting an amount of reserved resources. Containers run on specific nodes. Multiple containers can be scheduled onto the same node and then share the available resources. That is, while containers can provide resource isolation, they often do not. The resource management system usually runs co-located with a distributed file system, so that jobs can potentially have local access to their input data.

Distributed dataflow jobs can exhibit significantly different runtime behavior. Some jobs require mainly compute resources, others are predominantly constrained by network capacities or disk performance. Therefore, some jobs benefit hugely from locally accessible input files, others less so. There are also considerable differences in how much a job's performance is improved by using more compute resources. For example, a job that is mainly constrained by network throughput, not by available compute resources in terms of CPU cores and main memory, will not speed-up significantly by scaling the job to more nodes. Yet, it is difficult to statically infer a job's runtime behavior from programs, available information on input data, and resource specifications. First, the performance of distributed jobs depends on many factors, including for example user code, complex task dependencies, program parameters, system configurations, hardware resources, dataset characteristics, and data locality. Second, some of this information such as, for example, information on data skew is often not even available when data is processed ad-hocly from distributed file systems.

At the same time many jobs are recurring [8,9,10,11]. These jobs are typically scheduled batch jobs, running on a daily or weekly basis. They are also often executed repeatedly on only updated or at least similar datasets. For example, up to 60% of the jobs running on the larger analytics clusters at Microsoft are recurring [11]. Other studies report 40.32% of the jobs as well as 39.71% of the cluster hours to be recurring production jobs [9]. This presents an opportunity to learn workload characteristics as well as the runtime behavior of specific jobs on production clusters over time. Runtime statistics aggregated into workload profiles can then be incorporated into resource management to co-locate, for example, jobs with complimentary resource demands or place datasets on exactly as many nodes as are subsequently used for processing. Moreover, users often have specific performance demands, for example due to Service Level Objectives (SLOs). As it is difficult for users to estimate runtime performance, currently users tend to overprovision heavily to ensure minimal performance goals. This, however, leads to low overall cluster utilization and unnecessarily high operational costs [12,13]. Instead of having users essentially guess necessary resources, runtime statistics allow to model a job's scale-out behavior and allocate resources based on prediction models.

This chapter presents our vision of adaptive resource management for distributed data analytics. After presenting the central idea, we introduce a general architecture that comprises the main components of adaptive resource management: data placement, resource allocation, and job scheduling. Finally, we review and discuss existing work in these areas in context of our vision and architecture.

*Outline*. The remainder of this chapter is structured as follows. Section 2 provides some background on distributed data analytics. Section 3 presents the idea of adaptive resource management. Section 4 introduces a general architecture for adaptive resource management. The following three sections review existing adaptive approaches in the areas of data placement (Section 5), resource allocation (Section 6), and job scheduling (Section 7). Section 8 concludes this chapter.

## 2. Background and Related Work

This section describes how large datasets can be stored in a cluster using distributed file systems. Afterwards, we describe distributed dataflow systems and how they help users write massively-parallel jobs. Finally, we summarize how resource management systems enable the use of multiple such dataflow systems in shared cluster environments.

### *2.1. Distributed File Systems*

Distributed file systems handle large datasets by splitting them into small blocks. These blocks are distributed among a set of machines in a cluster. To cope with failures, typically, each block is replicated and redundantly stored on a few nodes. If a node stops working unexpectedly, then each block should still be available on other nodes. Figure 1 depicts how a large file is split into multiple blocks and distributed among nodes.

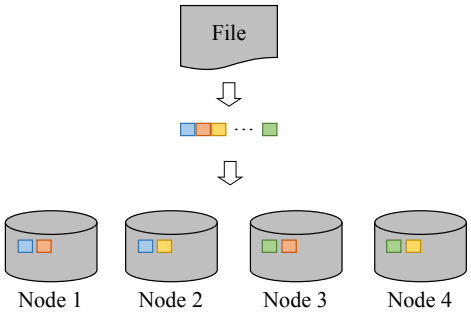


**Figure 1.** Saving a large file in a distributed file system.

When a machine wants to read parts of a file that are not locally available, then these parts must be transferred from a remote machine over the network. However, accessing file blocks that are locally available is in general significantly faster than fetching blocks from remote machines. This is known as data locality and is crucial for obtaining high performance [14].

Google File System (GFS) [15] and Hadoop Distributed File System (HDFS) [5] are optimized for distributed data processing. That is, they focus primarily on very large files and high throughput for sequential reads and writes. Both file systems utilize a single master node that handles the metadata and coordinates data storing as well as data access. Ceph [16] is a distributed file systems that provides a near-POSIX file system interface for its clients. The file system manages metadata using a distributed metadata cluster to improve scalability. GlusterFS [17] provides a fully POSIX-compliant distributed file system. It does not maintain any metadata servers. Instead, data is located algorithmically using an elastic hashing algorithm. Tachyon [18] is an in-memory distributed file system. As such, it provides high read and write performance for data-local tasks. Tachyon does not rely on replication for fault tolerance. Instead, lineage data is used to recompute lost datasets. In addition, a checkpointing algorithm is used to provide an upper bound for the time necessary to recompute data.

### 2.2. Distributed Dataflow Systems

Distributed dataflow frameworks help users develop distributed applications. Generally, they provide the users with predefined operators known from functional programming such as `map` and `reduce`. Users supply these operators with sequential code and connect them to form a directed acyclic graph (DAG), where edges represent the dataflow. After deploying a distributed dataflow system, users can submit dataflow graphs as jobs. The framework then takes care of parallelizing and distributing the jobs. For this, the framework translates the dataflow graph of a job into a task graph where each task is a data-parallel instance of an operator. Each of these task instances processes a split of the data. The splits are obtained by either reading data from a distributed file system or as the output from predecessor tasks. Figure 2 shows an example task graph for a word count job. The amount of parallel instances of an operator is called the degree of parallelism (DoP). Users can either manually configure the DoP or let the framework decide. Typically, the framework assigns tasks a DoP equal to the amount of cores available to the framework. For example, in Figure 2 the DoP for the map tasks is set to 2, while the reduce stage has a DoP of 3.

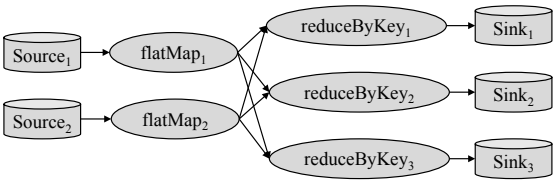


**Figure 2.** An exemplary task graph for a word count job.

MapReduce [1] introduced a programming and an execution model based on the two operators map and reduce. Both operators are supplied user-defined functions (UDFs). The map function reads key-value pairs in a data-parallel manner and transforms them into an intermediate list of mapped key-value pairs. Afterwards, the reduce function processes this intermediate list and merges all values with the same key into a list of reduced values. Intermediate data is stored to disk for fault tolerance in-between all stages of map and reduce tasks.

Dryad [19] and Nephele [20] allow users to express their distributed jobs by constructing arbitrary DAGs. The Stratosphere platform [21] extends Nephele with rich second-order functions [22], higher-level scripting language [23], dataflow optimization techniques [24,25], as well as native support for bulk and incremental iterations [26]. SCOPE [27,28] is based on a system similar to Dryad and features a DAG-based dataflow system with support for UDFs, an SQL-like scripting language, as well as automatic and continuous query optimizations [29,30,31,9].

Spark [2] also provides the user with a rich set of second-order functions. The system relies on Resilient Distributed Datasets (RDDs) [32] to efficiently execute iterative and interactive jobs. RDDs implement fault tolerance using lineage. This can speed up distributed computation significantly, compared to checkpointing intermediate data to disk. Spark also provides higher-level programming abstractions, including for processing relational data, in which case Spark also optimizes query plans automatically [33,34]. The D-Stream model [35] extends Spark with streaming capabilities by using micro-batches.

Flink [3], which originated from Stratosphere, implements a unified engine for stream and batch processing. While batch computations use the same streaming engine, Flink provides special treatment to the batch case. This includes distinct query optimizations and implementations of second-order functions. Flink implements fault tolerance by periodically taking snapshots of the operators' state. Google's Dataflow Model [4] presents a set of concepts and core principles for dataflow systems in the context of unbounded, unordered data streams.

Given the amount of different dataflow systems, Tez [36] has emerged. Tez is a framework that provides a reusable set of building blocks to implement DAG-oriented dataflow systems.

### *2.3. Resource Management Systems*

Resource management systems enable a single cluster to host different dataflow frameworks and run jobs from different users simultaneously. This way, they can improve cluster utilizations. The systems abstract the resources of a cluster with the notion of containers. A container might represent, for example, the number of cores and the amount of memory. Users then make reservations in terms of containers and resource managers schedule containers on nodes. Machines can be assigned one or multiple such containers, depending on their capacities and the size of containers. Some resource management systems also provide performance and security isolation for their containers.

Figure 3 visualizes the general idea behind resource management systems. In this example, the resource manager maintains two nodes where each node is assigned four containers. Clients execute applications by submitting jobs to the resource manager. The resource manager assigns each client four containers and starts the framework's master process in one of these containers. Subsequently, the framework can schedule and execute the worker processes on the remaining three containers.

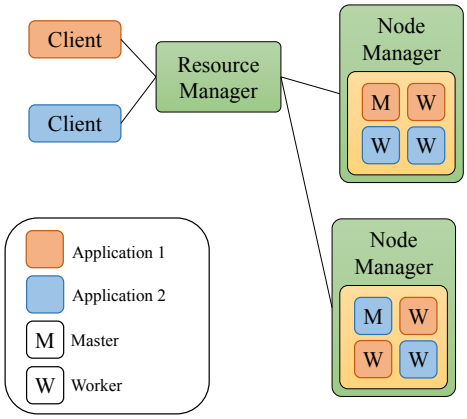


**Figure 3.** Cluster administration with resource management systems.

With resource managers, dataflow frameworks can run on a per-job basis. During job submission, the dataflow systems is automatically deployed on the reserved containers. After the job finishes, the dataflow system tears down. Resource managers can, however, also support longer sessions when users want to run multiple jobs on the same or related

data. Such long-running sessions are, for example, useful for interactive analysis, when multiple adhoc jobs are formulated and executed to examine the same dataset.

Mesos [7] is a resource manager that implements an offer-based resource assignment. Dataflow frameworks register designated scheduler applications that get notified when resources are available. A scheduler can accept or reject resource offers. Omega [37] is based on a lock-free protocol where all participating distributed schedulers can operate in parallel. All schedulers in Omega have full knowledge of the cluster state by holding a copy of the shared state. Acquiring resources is based on optimistic concurrency. Schedulers transactionally update the shared state and rollback when collisions are detected. YARN [6] utilizes a centralized design where a single resource manager keeps track of the cluster's state and handles resource requests. Users submit their jobs to the resource manager. When enough resources are available, the resource manager starts the job's master process for the given job. This application master typically hosts the central management unit of the dataflow framework. The dataflow framework then takes care of acquiring containers from the resource manager as well as scheduling and executing the actual job. Apollo [38] is a resource manager where each scheduler has access to the global cluster state. When a task is scheduled, it is put into the task queue of the respective node. Each node maintains a wait-time matrix that estimates when specific resource configuration will be available given the tasks in the queue. The job schedulers combine this matrix with other factors like data locality and task priority when making scheduling decisions. Similar to Omega, Apollo relies on optimistic concurrency control. However, Apollo features a late collision resolution where task assignments are checked after they are already put into the queue. Borg [39] uses a centralized resource manager that is replicated to achieve high availability. Submitted jobs are put into a pending queue. The scheduling components use the shared cluster state to process the job queue asynchronously and inform the master about task assignments. Optimistic concurrency is employed as the master rechecks the assignments.

## 3. Adaptive Resource Management

The central idea of adaptive resource management is to monitor a cluster and learn its workload characteristics over time.

Distributed dataflow jobs have significantly different resource requirements and runtime behavior. At the same time, production workloads often contain recurring jobs. Moreover, users tend to have specific performance goals specifically for these scheduled batch jobs. That is, a workload can also consist of largely similar jobs. For example, a production workload could consist of mainly I/O-heavy relational queries or mostly CPU-intensive machine learning algorithms. This information can be inferred from monitoring both applications and cluster resources. On the other hand, when specific jobs are executed repeatedly, for instance as daily scheduled batch jobs on continuously updated datasets, fine-grained monitoring allows to gather detailed statistics on the runtime behavior of these specific recurring jobs. Using this information resource management can adapt to the overall characteristics of a cluster workload as well as incorporate knowledge on particular jobs.

The general idea is depicted in Figure 4. Currently running data analytics jobs are monitored to obtain runtime statistics for these jobs. These statistics are analyzed to

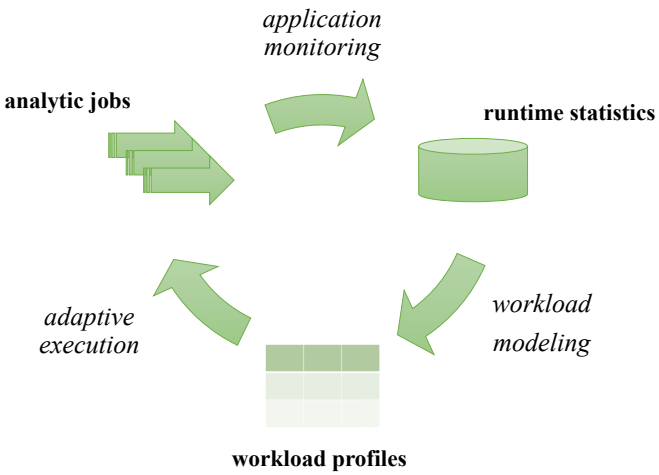


**Figure 4.** General idea behind adaptive resource management.

create profiles that model both the entire workload and specific jobs. These profiles are then incorporated into resource management decisions: placing data, allocating resources to jobs, as well as scheduling jobs to the cluster and on to particular cluster resources.

Fine-grained monitoring of applications and cluster resources can, for example, include:

- the runtimes of jobs and individual job stages,
- input data sizes,
- input data locality,
- convergence of active intermediate datasets in iterative jobs,
- failure rates,
- resource reservations,
- and detailed statistics on resource utilization.

These runtime statistics can be stored in a database for a cluster, providing historic workload data. Such data allows to create models of, for example, the impact of data locality on jobs, the scale-out behavior of jobs, and interference between co-located jobs. These models then allow to, for instance, give jobs that are highly sensitive to data locality priority over other jobs, when placing their containers in large clusters. Further, scale-out models can be used for automatic resource allocation, so that users explicitly state their runtime targets, instead of having to essentially guess adequate sets of resources for their goals. Models of job interference in turn allow to co-locate containers of jobs with complimentary resource demands and little interference. Besides modeling single jobs, it is also possible to model the entire workload, assuming it consists of similar jobs, or to cluster and classify jobs into a finite number of job classes. This allows to favor, for example, for workloads with lots of communication in general compact scheduling of job containers onto as few nodes as possible. In contrast, workloads that scale well could be scattered more across nodes.

## 4. System Architecture

This section describes our general system architecture for an adaptive resource management for distributed data analytics.

Figure 5 shows an overview of the proposed architecture. The system follows a master/worker paradigm, where the master node manages a number of worker nodes. The master node typically provides container scheduling and data block scheduling functionality. The worker nodes form the execution platform for the data analytics jobs.

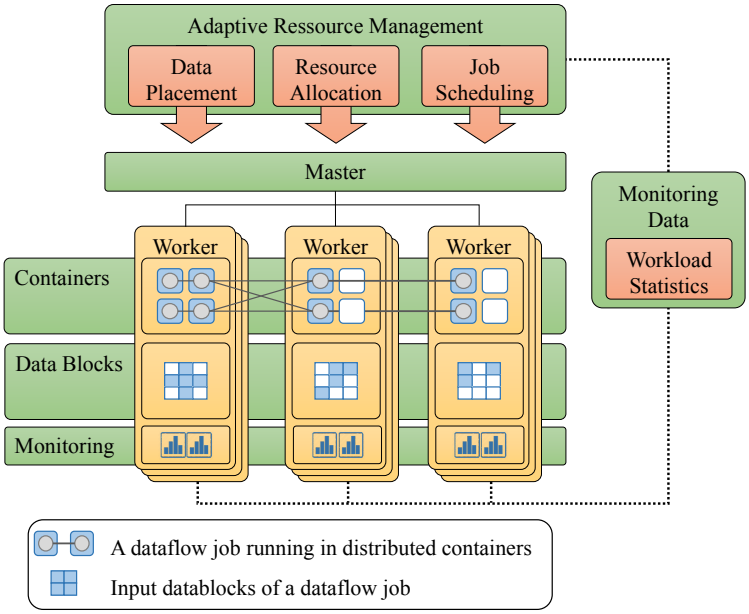


**Figure 5.** System architecture for adaptive resource management.

Containers provide data analytics jobs with compute resources such as CPU and memory. Containers are scheduled by a cluster resource manager. Datasets are stored in a distributed file system, which splits the dataset into series of replicated data blocks and distributes these across the worker nodes. Execution containers and distributed file system are often running co-located on all worker nodes to allow local data access. The idea of executing tasks on nodes where the input data can be accessed locally is called *data locality*.

Additionally, the system consists of a monitoring layer that provides container-level job monitoring of data analytics jobs. Specifically, this system tracks resource utilization associated with specific execution containers. The monitoring layer also observes application and data access logs to identify data access patterns for adaptive management of data placement. All monitoring data is stored in a workload repository that enables and provides input for our proposed adaptive resource management components. For example, the resource allocation component uses job resource utilization and runtimes of the past for automatic resource allocation according to users' performance targets. Different components of adaptive resource management are discussed in the following sections in more detail.

## 5. Data Placement

Distributed file systems such as HDFS [5] have become the de facto storage type for storing and accessing large amounts of data for analytical tasks. In these systems, input files are split into series of data blocks with multiple replicas stored on different data storage nodes. Besides scalability and fault tolerance, replication and file splitting can increase the performance of distributed dataflow engines, because tasks can access and process parts of the data stored on different nodes in parallel. However, the decoupling of storage and processing can also increase communication overhead if, for instance, the processing task is not directly scheduled on the node where the input data block relies on. In this case, the data block has to be transferred through the network to the node executing the processing task. Especially in large data analytic clusters that comprise of hundreds or thousands of nodes, storing a large number of different files and running many different jobs at the same time, the input data and execution container can be likely distributed on different nodes. Additionally, data-intensive jobs that, for instance, join or merge two or more files require a lot of network resources, because the related data blocks are likely not stored on the same set of nodes. Therefore, effective data placement is important in order to minimize the communication costs of data-intensive jobs. Recent work on data placement in data analytic clusters can be categorized in proactive data placement and active data placement.

The objective of proactive data placement is to place data blocks on desired nodes when it is loaded into the file system and afterwards schedule job executions on these nodes. Examples are recurring jobs, in which data is loaded from another system into the data analytic clusters and afterwards a data processing job is triggered on the new dataset. Examples can be found in click stream log analysis [40] or Surveillance Video Processing [41]. Coral [40] introduces a data and compute placement framework that jointly optimizes the location of data and tasks. It places input data and later tasks on a small number of racks to reduce the load on the often oversubscribed core network and to improve data locality. CoHadoop [42] enables the co-location of related files and their data blocks on the same set of data nodes based on a user-defined property. Therefore, the user or administrator has to tag which files are related and should be co-located. The first file and its data blocks are distributed with the default data placement scheduling approach. For the second, CoHadoop places the data blocks on the same set of nodes like the previous file. CoHadoop focuses on technical issues and leaves the task of tagging related files to the users. It furthermore does not take data locality into account. Golab et al. [43] automate the data placement process by proposing graph partitioning algorithms for computing nearly optimal data placement strategies for a given job. The objective is to decide where to store the data and where to place the tasks to minimize data communication costs. In contrast to CoHadoop, the workload must be known in advance. Additionally, they do not take care of parallel execution of tasks, which is an important feature of scalable data flow systems. CoLoc [44] works similar to CoHadoop. However, it focuses on recurring jobs and shared data-analytic cluster environments. The user has to annotate related files and can optionally specify the number of nodes to store these files on. The pool of data nodes is selected when the first file comes in. The second file and further ones are placed on the same set or subset of nodes. Additionally, CoLoc gives scheduling options to the container scheduler, where the input data is stored, and thus processing frameworks can benefit from high data locality and reduce communication overhead.

The objective in active data placement is to move and change the number of data block replications while files are already residing in the distributed file system. Techniques like Scarlett [45], ERMS (Elastic Replica Management System) [46], DARE (Adaptive Data Replication) [47], and the solution proposed by Bui et al. [48] use file system logs and application access patterns to increase and decrease the data replication factor dynamically in order to reduce job runtimes. Replications are spread across the whole cluster to avoid hot spots and increase the chance of high degrees of data locality. The replication factor of each file is derived from historical logs. One drawback of these techniques is the additional network communication and storage overhead for dynamic replication.

## 6. Resource Allocation

With basic resource management systems users typically specify the number and size of containers to be used for their distributed analytics jobs. However, estimating the resource demands and performance of such jobs is difficult [49,50,11,51]. This is due to the many factors involved such as complex task dependencies, user code, dataset characteristics, virtualization, and hardware resources. For this reason, a lot of work has focused on modeling job performance, dataflow runtime prediction, and resource allocation based on prediction models. Moreover, dynamic scaling can be used to adaptively allocate resources to approach user-defined performance targets at runtime and to address the inherent variance in job performance. These solutions typically require users to explicitly provide their performance goals besides programs and input parameters. Models for resource allocations are then typically build offline, either from previous executions of a job or from isolated profiling runs, yet may also be refined during the execution of a job. Systems then allocate the smallest or least expensive set of resources predicted to meet the performance goals. Some systems also infer performance goals from historic data or simply allocate as many resources as usable by a job, which is referred to as a job's *resource skyline*.

Many systems for automatic resource allocation use a white-box model of a particular dataflow engine. These include Aria [8,52], Elastisizer [53], AROMA [49], and Bazaar [50], which all model the performance of MapReduce jobs. Aria and Bazaar both use short sample runs to profile jobs for their MapReduce performance models. Aria automatically sets the DoP based on its model, while Bazaar sets the number of required instances and network bandwidth. Elastisizer also models MapReduce job performance based on detailed profiling information. It then simulates job executions given a user's specification of available resource types and system configurations, so users can see estimates of runtimes and costs before allocating resources. AROMA creates a distinct number of performance models to be used for allocating resources and configuring frameworks by clustering all previous executions. AROMA then performs short profiling runs with incoming jobs and matches them to one of the clusters. Jockey [10] also requires detailed statistics such as operator selectivity for its framework-specific performance model, yet was built for SCOPE, not MapReduce. Jockey furthermore monitors job performance at runtime and dynamically scales parallelism as well as resource usage to mitigate unexpected runtime behavior and variance. OptEx [54] estimates the runtime of Spark jobs given the size of the input data, the number of iterations, and the number of

nodes. OptEx has developers categorize jobs into distinct application classes and select a representative job per category. OptEx then creates a profile per job category by running the representative job. Each of these profile is then used to extrapolate from short sample runs on a single node to larger scales for jobs of the category.

In-between black-box and white-box approaches is PerfOrator [51]. PerfOrator first executes a set of pre-defined queries to model different hardware resources. It then uses profiling runs for modeling jobs and takes as another input a model of the analytical framework. PerfOrator currently supports Hadoop MapReduce and Tez, but these models can be extended to describe the parallelism of other distributed dataflow frameworks. PerfOrator requires more information than many black-box systems such as runtimes and CPU cycles spent processing for each stage measured in profiling runs. PerfOrator is, however, then able to make predictions for job runtimes and resource skylines on different hardware resources.

Black-box approaches to performance modeling allow to allocate resources for different dataflow systems. Solutions typically only require job runtimes and potentially resource utilization statistics as input, yet no framework-specific instrumentation data. Quasar [13] is a resource management system that jointly performs allocation and assignment of resources. For allocation, Quasar matches profiling data of jobs to distinct classes using collaborative filtering. Quasar then selects among heterogenous hardware for jobs based on user-provided performance constraints. At runtime, Quasar monitors job performance and, if necessary, re-classifies jobs and adjusts resource allocations dynamically. Ernest [55] fits samples to a simple model of distributed computation using regression. To train these models effectively using dedicated isolated profiling runs, Ernest applies methods for optimal experiment design [56]. Bell [57] is similar to Ernest in that it also uses regression to fit a function to a model for available samples. Bell, however, uses either a parametric model or nonparametric regression and automatically selects between these two methods using cross-validation. Bell models the performance of recurring jobs based on previous executions. Automatic model selection thereby allows Bell to use a robust parametric model for sparse training data and extrapolation, while nonparametric regression is used for interpolation tasks and dense training data, allowing Bell to model arbitrary scale-out behavior. Morpheus [11] is a system for resource allocation and scheduling of periodically running jobs. Morpheus infers deadlines for these repeatedly scheduled batch jobs from logs. It then uses resource utilization data from previous executions of these jobs to infer the maximal usable resources for the jobs, assuming overprovisioning by users. Consequently, Morpheus allocates these resource skylines for subsequent runs. It further attempts to run scheduled jobs together with the same other periodical jobs to decrease runtime variance due to interference. Morpheus also monitors jobs at runtime and dynamically adjusts resource allocations when resource usage deviates from expected behavior. A system by the authors of this chapter [58] monitors the performance of iterative distributed dataflow jobs and dynamically scales resource allocations in-between iterations. In particular it monitors resource utilization and scales towards user-defined utilization targets.

## 7. Job Scheduling

Job scheduling in the context of resource management for general-purpose distributed dataflow jobs comprises mainly two steps: selecting jobs to run on the cluster from a

queue of submitted jobs as well as assigning specific resources to specific jobs. That is, job scheduling determines the order of which jobs are executed and places the containers of a particular job onto particular cluster nodes. Adaptiveness can be introduced to this task in various ways. For example, knowledge of the resource demands or of the interference between co-located jobs can be used to select jobs with complimentary resource usage to run together on the cluster and, more fine-grained, on particular nodes. Another example is based on the observation that containers of jobs that require lots of communication and synchronization between parallel task instances fair better when scheduled compactly onto as few nodes as possible, whereas containers of jobs that mainly compute on single records require less communication between task instances and, thus, tend to run faster when scheduled onto many nodes. At the same time, containers of other jobs that mainly read and filter large datasets may benefit most from data locality. Knowledge of the resource demands of a particular job or a workload in general can thus be used for a more effective job scheduling, leading to a more efficient job execution. In the following we first present work on co-locating jobs with little interference. Subsequently we discuss Morpheus, which aims to run periodically scheduled batch jobs always together with the same jobs for increased runtime predictability.

Paragon [59] profiles incoming jobs, matches them to classes of similar jobs with respect to the impact of heterogeneous hardware and interference, and uses these classifications to assign jobs to specific cluster resources. Similarly to Paragon, Quasar [13] also uses classifications to determine the effect of resources and interference with other workloads when scheduling jobs to heterogenous cluster nodes. Quasar however, also takes users performance requirements into account and selects resources for these performance goals. Quasar further monitors job performance, re-classifies jobs when their performance deviates from expressed goals, and if necessary adjusts resource allocations at runtime.

A system by the authors of this chapter [60] learns over time which recurring jobs exhibit the least interference and achieve the highest resource utilization when running co-located on the same nodes. We use the Gradient Bandits method for estimating the distribution of co-location goodness and take for this metric CPU, disk, and network usage as well as I/O wait into account. In comparison to Paragon and Quasar, our system does not require isolated profiling of jobs, yet aims at improving throughput for repeatedly scheduled batch jobs. Paragon and Quasar though work with heterogeneous hardware, while our scheduler currently assumes a homogenous cluster setup.

Morpheus [11] is a system for both automatic resource allocation and job scheduling for recurring distributed dataflow jobs. Morpheus first infers required job completion times from historical data. It then assumes overprovisioning and determines the maximal used resources over time across all previous runs of a job. This amount of resources is initially allocated by the system, yet Morpheus also monitors resource utilization at runtime and adjusts reservations dynamically when the utilization exceeds expected demands. Furthermore, Morpheus attempts to reduce interference and improve runtime predictability by using the fact that periodical jobs can be executed with some flexibility and scheduling jobs repeatedly with the same recurring jobs.

## 8. Conclusion

In this chapter, we presented our vision of adaptive resource management for distributed data analytics. We presented a system architecture consisting of a resource management system, a distributed file system, and a monitoring system. The monitoring system tracks for each job runtimes, resource utilization on a container-level, and data access. This data can then be used to create statistical models for both the entire workload and specific recurring jobs. Such models can then be used to adapt resource management to the actual workload running on a specific cluster. In particular, three main components can make resource management decisions based on such models: data placement, resource allocation, and job scheduling. Reviewing work towards a more adaptive resource management in these three areas, we saw that lots of recent work has focused on performance prediction and resource allocation, less on scheduling data blocks and containers for distributed data analytics based on statistical models. Additionally, only little work has been done combining the discussed adaptive resource management components in a single and uniformed system. We believe as distributed data analytics becomes more and more important for businesses and research, so that consequently the number of jobs as well as cluster sizes increase, more adaptive resource management can help utilize cluster resources efficiently without requiring users to manually fine-tune systems.

## Acknowledgements

This work has been supported through grants by the German Science Foundation (DFG) as FOR 1306 Stratosphere and by the German Ministry for Education and Research (BMBF) as Berlin Big Data Center BBDC (funding mark 01IS14013A).

## References

[1] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation*, OSDI'04, pages 10–10. USENIX Association, January 2004.

[2] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10. USENIX Association, June 2010.

[3] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache Flink: Stream and Batch Processing in a Single Engine. *IEEE Data Engineering Bulletin*, 38(4):28–38, July 2015.

[4] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle. The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-scale, Unbounded, Out-of-order Data Processing. *Proc. VLDB Endow.*, 8(12):1792–1803, August 2015.

[5] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, May 2010.

[6] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16. ACM, September 2013.

[7] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion 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*, NSDI'11, pages 295–308. USENIX Association, March 2011.

[8] Abhishek Verma, Ludmila Cherkasova, and Roy H. Campbell. ARIA: Automatic Resource Inference and Allocation for Mapreduce Environments. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 235–244. ACM, June 2011.

[9] Sameer Agarwal, Srikanth Kandula, Nico Bruno, Ming-Chuan Wu, Ion Stoica, and Jingren Zhou. Reoptimizing Data Parallel Computing. In *In Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 281–294. USENIX Association, April 2012.

[10] Andrew D. Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. Jockey: Guaranteed Job Latency in Data Parallel Clusters. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 99–112. ACM, April 2012.

[11] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Íñigo Goiri, Subru Krishnan, Janardhan Kulkarni, and Sriram Rao. Morpheus: Towards Automated SLOs for Enterprise Clusters. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 117–134. USENIX Association, November 2016.

[12] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 7:1–7:13. ACM, October 2012.

[13] Christina Delimitrou and Christos 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*, ASPLOS '14, pages 127–144. ACM, March 2014.

[14] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, J. Majors, A. Manzanares, and Xiao 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)*, pages 1–9, April 2010.

[15] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43. ACM, October 2003.

[16] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: A Scalable, High-performance Distributed File System. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 307–320. USENIX, October 2006.

[17] Alex Davies and Alessandro Orsaria. Scale Out with GlusterFS. *Linux Journal*, 2013(235), nov 2013.

[18] Haoyuan Li, Ali Ghodsi, Matei Zaharia, Scott Shenker, and Ion Stoica. Tachyon: Reliable, memory speed storage for cluster computing frameworks. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–15. ACM, 2014.

[19] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 59–72. ACM, March 2007.

[20] Daniel Warneke and Odej Kao. Nephele: Efficient Parallel Data Processing in the Cloud. In *Proceedings of the 2Nd Workshop on Many-Task Computing on Grids and Supercomputers*, MTAGS '09, pages 8:1–8:10. ACM, November 2009.

[21] Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, Felix Naumann, Mathias Peters, Astrid Rheinländer, Matthias J. Sax, Sebastian Schelter, Mareike Höger, Kostas Tzoumas, and Daniel Warneke. The Stratosphere Platform for Big Data Analytics. *The VLDB Journal*, 23(6):939–964, December 2014.

[22] Dominic Battré, Stephan Ewen, Fabian Hueske, Odej Kao, Volker Markl, and Daniel Warneke. Nephele/PACTs: A Programming Model and Execution Framework for Web-scale Analytical Processing. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 119–130. ACM, June 2010.

[23] Arvid Heise, Astrid Rheinländer, Marcus Leich, Ulf Leser, and Felix Naumann. Meteor/Sopremo: An Extensible Query Language and Operator Model. In *Proceedings of the International Workshop on End-to-end Management of Big Data (BigData) in conjunction with VLDB 2012*, BigData '2012, pages 1–10. VLDB Endowmen, August 2012.

[24] Fabian Hueske, Mathias Peters, Matthias J. Sax, Astrid Rheinländer, Rico Bergmann, Aljoscha Krettek, and Kostas Tzoumas. Opening the Black Boxes in Data Flow Optimization. *Proc. VLDB Endow.*, 5(11):1256–1267, July 2012.

[25] Astrid Rheinländer, Arvid Heise, Fabian Hueske, Ulf Leser, and Felix Naumann. SOFA. *Information Systems*, 52(C):96–125, August 2015.

[26] Stephan Ewen, Kostas Tzoumas, Moritz Kaufmann, and Volker Markl. Spinning Fast Iterative Data Flows. *Proc. VLDB Endow.*, 5(11):1268–1279, July 2012.

[27] Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets. *Proc. VLDB Endow.*, 1(2):1265–1276, August 2008.

[28] Jingren Zhou, Nicolas Bruno, Ming-Chuan Wu, Per-Ake Larson, Ronnie Chaiken, and Darren Shakib. SCOPE: Parallel Databases Meet MapReduce. *The VLDB Journal*, 21(5):611–636, October 2012.

[29] Jingren Zhou, Per-Åke Larson, and Ronnie Chaiken. Incorporating Partitioning and Parallel Plans into the SCOPE Optimizer. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 1060–1071. IEEE, March 2010.

[30] Nicolas Bruno, Sameer Agarwal, Srikanth Kandula, Bing Shi, Ming-Chuan Wu, and Jingren Zhou. Recurring Job Optimization in SCOPE. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 805–806. ACM, May 2012.

[31] Nicolas Bruno, Sapna Jain, and Jingren Zhou. Continuous Cloud-scale Query Optimization and Processing. *Proc. VLDB Endow.*, 6(11):961–972, August 2013.

[32] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2. USENIX Association, April 2012.

[33] Reynold S. Xin, Josh Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, and Ion Stoica. Shark: SQL and Rich Analytics at Scale. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 13–24. ACM, June 2013.

[34] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. Spark SQL: Relational Data Processing in Spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1383–1394. ACM, May 2015.

[35] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. Discretized Streams: Fault-tolerant Streaming Computation at Scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 423–438. ACM, October 2013.

[36] Saha, Bikas and Shah, Hitesh and Seth, Siddharth and Vijayaraghavan, Gopal and Murthy, Arun and Curino, Carlo. Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1357–1369. ACM, June 2015.

[37] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: Flexible, Scalable Schedulers for Large Compute Clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 351–364. ACM, April 2013.

[38] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. Apollo: Scalable and Coordinated Scheduling for Cloud-scale Computing. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 285–300. USENIX Association, October 2014.

[39] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale Cluster Management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, pages 18:1–18:17. ACM, April 2015.

[40] Virajith Jalaparti, Peter Bodik, Ishai Menache, Sriram Rao, Konstantin Makarychev, and Matthew Caesar. Network-aware scheduling for data-parallel jobs: Plan when you can. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 407–420. ACM, 2015.

[41] Haitao Zhang, Bin Xu, Jin Yan, Lujie Liu, and Huadong Ma. Proactive Data Placement for Surveillance Video Processing in Heterogeneous Cluster. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 206–213, Dec 2016.

[42] Mohamed Y Eltabakh, Yuanyuan Tian, Fatma Özcan, Rainer Gemulla, Aljoscha Krettek, and John McPherson. CoHadoop: Flexible Data Placement and Its Exploitation in Hadoop. *Proceedings of the*

*VLDB Endowment*, 4(9):575–585, 2011.

[43] Lukasz Golab, Marios Hadjieleftheriou, Howard Karloff, and Barna Saha. Distributed Data Placement to Minimize Communication Costs via Graph Partitioning. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, SSDBM '14, pages 20:1–20:12. ACM, June 2014.

[44] Thomas Renner, Lauritz Thamsen, and Odej Kao. CoLoc: Distributed Data and Container Colocation for Data-Intensive Applications. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1–6. IEEE, December 2016.

[45] Ganesh Ananthanarayanan, Sameer Agarwal, Srikanth Kandula, Albert Greenberg, Ion Stoica, Duke Harlan, and Ed Harris. Scarlett: Coping with Skewed Content Popularity in Mapreduce Clusters. In *Proceedings of the Sixth Conference on Computer Systems*, pages 287–300. ACM, April 2011.

[46] Zhendong Cheng, Zhongzhi Luan, You Meng, Depei Qian, Alain Roy, and Gang Guan. ERMS: An Elastic Replication Management System for HDFS. In *2012 IEEE International Conference on Cluster Computing Workshops*, pages 32–40. IEEE, September 2012.

[47] Cristina L. Abad, Yi Lu, and Roy H. Campbell. DARE: Adaptive Data Replication for Efficient Cluster Scheduling. In *2011 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 159–168. IEEE, September 2011.

[48] Dinh-Mao Bui, Shujaat Hussain, Eui-Nam Huh, and Sungyoung Lee. Adaptive Replication Management in HDFS Based on Supervised Learning. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1369–1382, June 2016.

[49] Palden Lama and Xiaobo Zhou. AROMA: Automated Resource Allocation and Configuration of Mapreduce Environment in the Cloud. In *Proceedings of the 9th International Conference on Autonomic Computing*, ICAC '12, pages 63–72. ACM, September 2012.

[50] Virajith Jalaparti, Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Bridging the Tenant-provider Gap in Cloud Services. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 10:1–10:14. ACM, October 2012.

[51] Kaushik Rajan, Dharmesh Kakadia, Carlo Curino, and Subru Krishnan. PerfOrator: Eloquent Performance Models for Resource Optimization. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, pages 415–427. ACM, 2016.

[52] Abhishek Verma, Ludmila Cherkasova, and Roy H. Campbell. Resource Provisioning Framework for Mapreduce Jobs with Performance Goals. In *Proceedings of the 12th ACM/IFIP/USENIX International Conference on Middleware*, Middleware'11, pages 165–186. Springer, December 2011.

[53] Herodotos Herodotou, Fei Dong, and Shivnath Babu. No One (Cluster) Size Fits All: Automatic Cluster Sizing for Data-intensive Analytics. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pages 18:1–18:14. ACM, October 2011.

[54] Subhajit Sidhanta, Wojciech Golab, and Supratik Mukhopadhyay. OptEx: A Deadline-Aware Cost Optimization Model for Spark. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, CCGrid, pages 193–202. IEEE, May 2016.

[55] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. Ernest: Efficient Performance Prediction for Large-scale Advanced Analytics. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, pages 363–378. USENIX Association, March 2016.

[56] Pukelsheim, Friedrich. *Optimal Design of Experiments*, volume 50. SIAM, March 1993.

[57] Lauritz Thamsen, Ilya Verbitskiy, Florian Schmidt, Thomas Renner, and Odej 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*, pages 1–6. IEEE, 2016.

[58] Lauritz Thamsen, Thomas Renner, and Odej Kao. Continuously improving the resource utilization of iterative parallel dataflows. In *Distributed Computing Systems Workshops (ICDCSW), 2016 IEEE 36th International Conference on*, pages 1–6. IEEE, 2016.

[59] Christina Delimitrou and Christos Kozyrakis. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 77–88. ACM, March 2013.

[60] Lauritz Thamsen, Benjamin Rabier, Florian Schmidt, Thomas Renner, and Odej. Kao. Scheduling Recurring Distributed Dataflow Jobs Based on Resource Utilization and Interference. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 145–152. IEEE, June 2017.