POSUM: A Portfolio Scheduler for MapReduce Workloads

Maria A. Voinea
Technische Universiteit Delft
m.voinea@student.tudelft.nl

Alexandru Uta
Vrije Universiteit Amsterdam
a.uta@vu.nl

Alexandru Iosup
Vrije Universiteit Amsterdam
a.iosup@vu.nl

***Abstract*—MapReduce ecosystems are (still) widely popular for big data processing in data centers. To address the diverse non-functional requirements arising from many and increasingly more sophisticated users, the community has developed many scheduling policies for MapReduce workloads. Although some individual policies can dynamically optimize for single and stable performance objectives, such as minimizing runtime or cost, or meeting deadlines for realtime-jobs, it seems unlikely that individual policies will remain competitive for increasingly more dynamic workloads and objectives. In contrast, in this work we investigate the ability to dynamically balance performance and cost of a *portfolio scheduler* for MapReduce workloads. To this end, we design and implement a portfolio scheduling technique, that is, a system capable of adapting to the current workload characteristics and target objectives by periodically evaluating its set of potential policies, and of switching to ”the best” policy that targets the current system state. We implement and evaluate our system with real-world experiments on a workload containing a mixture of real-time and batch jobs, with the purpose of minimizing deadline violations, while keeping batch job slowdown in check. Our results show that POSUM is a promising alternative: it can out-perform the individual policies of its portfolio for the combined optimization goal, even without precise predictions.**

I. Introduction

Although not a universal panaceum for big data problems, MapReduce [1] systems are widely used in industry, governance, and academia [2]. Scheduling MapReduce workloads is thus important, and, as literature indicates [3], [4], [5], extremely challenging. Various schedulers and resource managers have already been proposed to address MapReduce stragglers [6], resource utilization [7], and job deadlines [8]. However, designing only one scheduler to optimize for multiple performance goals is difficult, error-prone, and ephemeral in that a change of goals can render the scheduler ineffective. In contrast, we propose a system that dynamically switches between a set of scheduling policies, to achieve the desired (and possibly changing) goals. This concept, of portfolio scheduling [9], [10], has never been investigated for big data workloads, and in particular has never been used for MapReduce.

Companies are increasingly relying on big data and business analytics to make their products and services customer-centric, improve operational performance, and even identify new business opportunities. Between 2015 and 2017, big data adoption in the industry has increased by 17%, and 53% of the companies that participated in an interview [11] declared that they used big data techniques. Moreover, concurrently, governments and municipalities [12], [13] are also investing in big data analytics for improving public health and safety, and sustainability.

MapReduce ecosystems [14], through many representatives [15], [16], [17], provide a widely-used and adopted programming model for big data processing, developed to achieve abstraction and scalability with reasonable drawbacks [1]. Already highly versatile, MapReduce clusters are used to run varied workloads: from batch jobs, to latency-sensitive and deadline-constrained jobs, spanning many application domains: analytics, business intelligence, ETL, and warehousing.

Such abundance of use cases has given rise to an abundance of scheduling policies built for reaching the respective operational and performance goals of individual usage scenarios. Most straightforward and common policies try to reduce the runtime of workloads. For MapReduce applications that are part of a more elaborate business pipeline, the scheduling policy needs to manage deadlines [5]. Other schedulers may focus on the financial cost of allocating resources [18], on reducing slowdown variability [19], on achieving resource fairness [20], on managing long delay tails [21], or on energy efficiency [22].

However, user needs are increasingly more sophisticated. And usage patterns are becoming more complex. Similarly, scheduling policies can be conceived to work at different levels and may have various effects depending on workload composition [23]. But to allocate data center resources and balance achieving functional and performance goals, current MapReduce frameworks, such as Hadoop YARN, select and then rely on a *single* scheduling policy, often based on simple heuristics. Once selected (by the system administrator or user), only one such policy is in use until the resources are deallocated.

We argue that leveraging a single scheduling policy in a complex, multi-user MapReduce system is ill-suited for catering to the needs of all its users, leading to poor (overall) performance. Moreover, designing and maintaining one such policy is difficult to achieve, given the highly dynamic and transient nature of data center workloads.

To address this problem, in this work we propose POSUM, a POrtfolio SchedUler for Mapreduce, as an alternative for tackling compound objectives for dynamic MapReduce workloads. At its core, POSUM relies on online simulation to evaluate, given the current state of the system, which policy will perform best, before switching to it for a given time period. This allows each policy to remain manageable in terms of complexity and hold true to its target use case, while still leaving room for the system to adapt at runtime and achieve the necessary performance objectives. In this work, our contribution is fourfold:

1) We design a generic portfolio scheduler architecture for MapReduce systems (Section III). We implement POSUM as a modular open-source addition to Hadoop YARN, which serves as a platform for practitioners and researchers to further investigate and extend portfolio scheduling for MapReduce ecosystems.
2) We propose an easily extensible set of scheduling policies, the portfolio of POSUM (Section IV). Our policies of choice represent state-of-the-art scheduling techniques for MapReduce systems.
3) We design and open-source a generic simulator for the online prediction of MapReduce task behavior (Section V).
4) We implement POSUM and evaluate it through real-world experiments (Section VI). Our workload is a mixture of real-time and batch jobs, with the purpose of minimizing deadline violations, while keeping batch job slowdown in check.

II. MapReduce and Portfolio Scheduling

In this section we introduce the core-concepts and techniques used in state-of-the art MapReduce scheduling.

Job scheduling is a well-developed subject in the data center context. For MapReduce specifically, multiple scheduling policies have been devised to prioritize and optimize job execution, considering the particular characteristics of these workloads. The first MapReduce schedulers operated on *First-In-First-Out (FIFO)* order. Later, *Fair Scheduling* [24] was introduced for using max-min fairness to share resources between *pools* of jobs or users.

Within the same job, tasks have always had higher priority on nodes holding their data. However, *delay scheduling* approaches [25] [3], relax inter-job priority even more to achieve better data locality. Cheriere et al. [26] argue that considering data locality when choosing which job should run on a given free slot leads to long wait times for small short jobs. Their *Shortest Remaining Time First (SRTF)* scheduling policies give higher priority to such jobs. Nguyen et al. [4] use a compound metric with variable coefficients to control the extent to which these short jobs are favored.

Meeting specific Service Level Objectives (SLOs) for jobs is another direction of optimization for MapReduce schedulers. Kc and Anyanwu [27], Polo et al. [28], Dong et al. [29], and Verma et al. [8] use historical information to predict future resource needs for achieving the required deadlines. Lim et al. [5] attempt to address the issue using offline constraint programming (CP). Other directions of research include mitigating stragglers (hanging tasks that need to be restarted), dynamic voltage scaling, virtualization, etc.

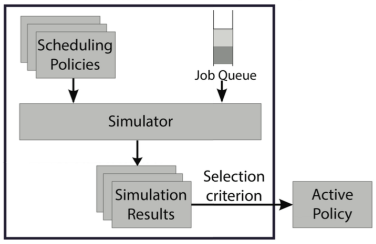

Fig. 1. The classic periodic portfolio scheduler for datacenters [9].

Our approach for achieving compound objectives under variable workloads relies on portfolio scheduling. The technique was borrowed from economics and first introduced to the field by Deng et al. [9]. As seen in Figure 1, the scheduler contains a portfolio of policies which it evaluates periodically using a simulator. The simulator predicts the behavior of each policy under the current system load, given the queued jobs. It returns a score as a combination of performance metrics. The scheduler chooses the policy with the highest score to switch to for the duration of the next period. The same concept has been successfully applied in previous studies to schedule scientific workloads on a compute cluster [30] and long-running virtual machines in data centers [10].

III. Design of a Generic Portfolio Scheduler for Datacenter-based MapReduce

In this section, we introduce and discuss the architecture of POSUM, and motivate the design decisions behind our proposed architecture. We start by presenting our requirements, and introduce the architecture that augments the MapReduce system with portfolio scheduling capabilities.

We start the design process of the portfolio scheduler with the following requirements. In short, POSUM should:

1) Include necessary conceptual elements of portfolio scheduling found in previous work;
2) Take into account the characteristics of MapReduce processing;
3) Adapt the resulting scheduler architecture to an actual, widely used MapReduce framework;
4) Keep the design flexible so as to explore and compare different approaches;
5) Follow or compare to state-of-the-art techniques wherever possible.

Addressing these goals, we design POSUM, an online meta-scheduler that can switch scheduling policies at runtime, based on real-time policy performance evaluations. POSUM is meant to be self-contained and exist alongside a MapReduce cluster, only interacting with it when gathering runtime information and for switching scheduling policies. This makes it possible to easily integrate it with any MapReduce runtime framework. To test compliance with the *third design goal*, our implementation is integrated with the Hadoop (YARN) stack.

A. POSUM Architecture

The POSUM system follows the high-level architecture illustrated in Figure 2. It is comprised of three main processes which interact during runtime: (i) the *Data Master*, (ii) the *Simulator*, and (iii) the *Orchestrator*.

The Data Master is responsible for monitoring the system and providing a coherent view of the stored statistics to the other processes. Two monitoring processes operate on a configurable heartbeat. First, the *Cluster Monitor* gets real-time information about applications and tasks that are running from the MapReduce framework. Second, the *POSUM Monitor*, gathers and interprets data on the operation of POSUM itself: simulation durations, the discrepancy between simulation scores and actual policy performance, etc.

The information gathered by the Data Master is used most intensively by the Simulator Master, a loosely coupled component that can simulate the outcome of a scheduling policy, given a certain queue composition, cluster state, and previous runtime statistics (see Section V).

All decisions regarding POSUM's operations are made by the Orchestrator. It triggers or stops policy scoring simulations, it handles the application of policies, and makes decisions based on the feedback gathered by the system monitor. It is also responsible for applying provisioning decisions, by reconfiguring the cluster to make use of or free recently added nodes.

However, actual policy application is done via the Portfolio Meta Scheduler, which is not an independent process, but a placeholder that extends the standard resource scheduler interface of the target framework (i.e., in our case, Hadoop). It delegates all its public and protected methods to the current policy that is being applied. It keeps evidence of the available scheduling policies and uses the logic of the currently plugged-in policy to reach each scheduling decision. This abstraction ensures that the transition between policies is seamless and does not disrupt the framework's operation. We have found that this component also needs access to the statistics on application progress, when policies need more information about the running jobs to take decisions.

Keeping the architecture modular achieves separation of concerns and enables both flexibility in approach exploration (demanded by the *fourth goal*) and runtime performance tweaking. Each process can be deployed to a separate machine, can have different JVM characteristics, and can be restarted independently on failure.

IV. Portfolio of Scheduling Policies

In this section we propose a portfolio of four scheduling policies to be used by POSUM during runtime. Our proposed set of policies is not exhaustive, but the POSUM implementation permits researchers and practitioners to easily implement their policy of choice. We start by motivating our choice of the four policies chosen and describe their functionality. Table I summarizes the policies implemented in POSUM.

The policies contained in the portfolio should be representative and capable of performing on different workload patterns and application types. However, their number should be relatively restricted, so as to minimize the exploration step during the selection and application phases. Seeing as this is the first exploratory study of portfolio scheduling on MapReduce, and the operational model is not confined to a particular domain, the set of policies is inspired by a survey of the field. The following two performance dimensions are kept in mind: respecting deadlines and lowering batch job runtime (see Section VI for the system model details). For deadline constrained (DC) jobs, the baseline policy is usually Earliest-Deadline-First (EDF) in the literature [27], [28], [29]. Following the same reasoning, for the second performance dimension, jobs with larger slowdown should have higher priority. Thus, for batch jobs (BC), the Largest-Slowdown-First (LSF) heuristic [30] is a good candidate. However, a form of prioritization needs to be established between the two categories as well.

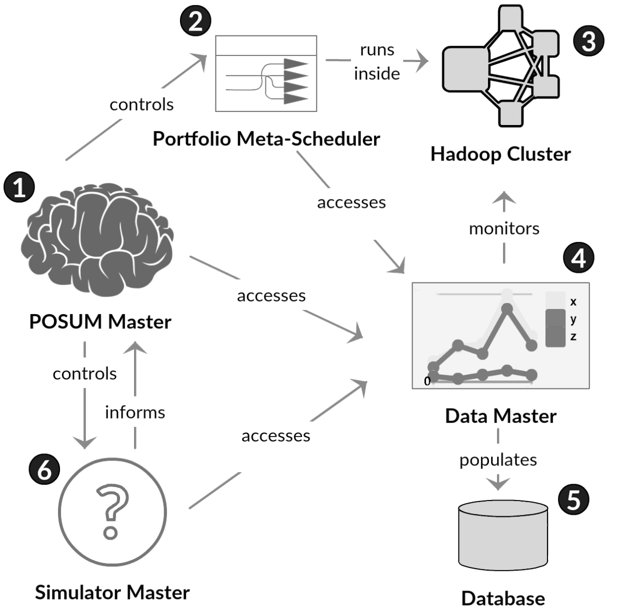

Fig. 2. POSUM modular architecture overview.

Two solutions are adopted for this, resulting in the pair of policies *EDLS-Sh* and *EDLS-Pr*. The δ parameter is configurable in the system (between 0 and 1) and represents the importance of DC jobs for the cluster owner. The two policies optimize by tailoring to the types of jobs in the workload. It is to be noted that although they derive conceptually from general cluster scheduling techniques, it is not entire jobs that are scheduled using the heuristics, but their constituent tasks. This results in constant reshuffling of the jobs in the priority queues even while they are running, a characteristic of MapReduce processing.

Another pair of policies is added to optimize for the size of the jobs in the workload: *hSRTF* and *LOCF*. The former should give preference to jobs that have smaller input and shorter execution times, while the latter favors the ones with large input sizes. This happens because a larger input would be distributed on a larger portion of the cluster, increasing the probability of at least one task being local on the target node (at least in the beginning of the job's execution).

The resulting set of four policies thus achieves a balance

TABLE I
THE PORTFOLIO OF POLICIES USED BY POSUM.

Name	Type	Description
EDLS-Sh	allocation	Share-based scheduler. DC jobs are ordered by EDF and get a share of the cluster equal to δ. BC jobs are ordered by LSF and get $(1-\delta)$ of the cluster.
EDLS-Pr	allocation	Order-based scheduler. DC jobs are ordered by EDF in one queue. The BC jobs are in LSF order in another queue. When resources become available, the DC queue has a chance equal to δ of receiving them. Otherwise, the BC queue does.
hSRTF	allocation	Share-based version of the Shortest-Remaining-Time-First scheduler [26].
LOCF	allocation	FIFO scheduler that enforces locality along the lines of [25] and [3].

between both types of priority enforcement, while tailoring to specific workload characteristics. Moreover, they enrich traditional job scheduling approaches with MapReduce-specific dimensions like task performance and data locality (in line with the *second design goal*). The full spectrum of policies is explored on each simulation. Once a decision is made, the sub-policy is plugged into the system as the main scheduler.

V. Design of a Generic Simulator for Datacenter-based MapReduce

In this section we introduce the design of the POSUM simulator component. POSUM uses a discrete-event simulator which mimics the same message-based event handling mechanism that enables MapReduce frameworks, like Hadoop, to operate. Resources are not modeled explicitly so as to reduce simulation time as much as possible (since the decisions need to be real-time). For further simplification, failure events are not considered in this version of POSUM. The simulator hypothesizes about the behavior of jobs as they come into the system, by looking at their configuration, their current behavior (if they are already running), and historical data gathered from jobs that have already run on the system, that may or may not have had similar characteristics.

The heart of the simulator is the Predictor, a component responsible for estimating how long each task will take. It calculates task runtimes by looking at *similar* tasks that have been run on the system, using a technique adapted from literature [28] [27]. We consider tasks similar if they run the same map/reduce function, are submitted by the same user, or at least have the same type (map or reduce). Task durations are considered directly dependent on the data they have to process. So to obtain the duration of a map task, we calculate the *average input processing rate* from the past map tasks and multiply it by the size of the input split of that specific task. For reduce tasks, we try to do the same, but we can only estimate the size of their input, by calculating the average *selectivity* (ratio of output size to input size) of map tasks that are similar to those of the current job. If no history is available at all, but some map tasks have already completed, the processing rate of reduces is considered equal to the map processing rate. The result is close to what describe, without differentiating between the reduce steps.

VI. POSUM Experimental Evaluation

In this section we present our experimental evaluation of POSUM. The scope of our empirical analysis is threefold. First, we perform an analysis of the simulator. Second, we present a qualitative performance comparison between dynamic policy application (i.e., portfolio scheduling) and running each policy of the portfolio policies separately. Finally, we present evidence for the resource utilization of the POSUM components. The main findings of our empirical analysis are:

MF1 The simulator is roughly accurate when forecasting tasks which do not exhibit variability (e.g., maps), but the quality of the simulation degrades for reduce tasks which exhibit variability. Also, simulations and policy changes add little overhead to the system's functioning.

MF2 Portfolio scheduling cannot outperform policies that target single performance objectives. However, it balances mixed performance goals well, even in the absence of accurate task runtime predictions.

Implementation and experimentation effort: POSUM is implemented in Java in 70,000 LoC during 12 person-months. Another 3 person-months have been used for testing, validation and the experimental analysis described in this section.

A. Hardware and Software Environment

POSUM is integrated into the Hadoop (YARN) 2.7.1 stack. Our implementation is open-source and can be accessed online in our Hadoop repository fork[1]. All the components are started as separate processes that communicate via the same RPC mechanism that Hadoop uses internally. Experiments are carried out using 10 nodes (8 worker nodes, 1 Hadoop master, and 1 node running the POSUM processes) in the TU Delft site of the DAS-5[2]. Each machine is equipped with two 8-core Intel E5-2630v3 CPUs, 64 GB memory, and 4 TB HDD. The nodes are interconnected with two networks: a 54-Gbps FDR InfiniBand, and a 1 Gbit Ethernet. The cluster nodes are running CentOS 7.2, Linux Kernel 3.10, and the JVM used to run Hadoop is OpenJDK 1.8.

B. Workload Generation

Our workloads are a mixture of real-time and batch jobs, with the purpose of minimizing deadline violations, while keeping batch job slowdown in check. As production trace data is often lacking in runtime system information, we run real-world experiments using the micro-benchmarking tool called BigDataBench[3] [31] (inspired by HiBench [32]). The

[1] https://atlarge.ewi.tudelft.nl/gitlab/m.voinea/hadoop

[2] http://www.cs.vu.nl/das5/

[3] http://prof.ict.ac.cn/BigDataBench/

TABLE II
WORKLOAD APPLICATION TYPES.

Application	CPU Utilization	I/O Utilization	Data Sizes
Sort	Low	High	8 - 56 GB
WordCount	High	Low	2 - 35 GB
Nutch Indexing	High	Medium	50K - 2M websites, 300 MB - 12 GB
Naive Bayes Classification	Low	High	2 - 18 GB

workloads are synthetic, created as a mix of four job types: Sort, WordCount, Naive Bayes classification, and Nutch page indexing. The input data for each job is generated with sizes drawn from an exponential distribution of predominantly small values, as per the findings by Chen et al. [23]). In the case of the real-time applications, deadlines are generated using a technique from literature [28], [5], i.e. the time it takes to run an application with the same data size alone on the cluster, multiplied by the a relaxation factor. The jobs arrive on average every minute for the duration of one hour. Using such techniques we randomly generate different workloads (i.e., the sequence of the jobs, and their count are different) to evaluate our portfolio scheduler. The specific application types within workloads and their descriptions can be found in Table II.

C. Simulator

Validating the accuracy of the simulator is not straightforward. The simulator needs to hypothesize about task-to-node allocation given a large number of tasks, fluctuating prediction accuracy, and a continuously evolving internal state of the simulated scheduling policy. While all this is hard to capture with a single evaluation function, we can, however create a basic visualization of the state of the cluster over time, comparing reality to the periodic simulations. For this we use a small workload of 7 mixed jobs arriving all at once, configure POSUM to run and simulate only the SRTF policy, and train the predictor on the trace of a previous run of the same workload on the real cluster.

Figure 3 shows the evolution of the number of running tasks on the cluster, grouped by job. We notice that the first simulation starts only a few seconds into the run. The distribution of tasks in the beginning of the simulation follow roughly the same pattern as on the real cluster, however, the predictor underestimates the total time it would take Job 2 to finish, giving it priority over the rest of the jobs. In the second simulation (roughly ten minutes into the run), prediction is more accurate for Job 2, sending it to the end of the job queue, but Job 1 is expected to finish much earlier than in reality. This causes Job 6 to take over the clusters slots 2 minutes before this happens in the real cluster, and all following jobs to start earlier. Also, the final reduce tasks of Job 2 are greatly underestimated for the entire run of the workload, creating a large discrepancy between simulations and reality, with respect to the finish time of the workload. Upon a more detailed analysis, we can attribute this to the fact that most reduces suffer from significant performance variability that is not dependent on input size. For example, sort reduce tasks can take anywhere from 30 seconds to 15 minutes to process 13.6 GB of data.

Even though this analysis gives no quantitative evidence of the performance of the simulator, we believe that our visualization supports **MF1**. It shows that the allocation decisions of the simulated policy will be similar to reality, as long as task runtime predictions are accurate enough. We consider this result to be sufficient for carrying out the performance experiments that follow.

Next, we looked at the overhead of simulations and policy application. Simulations last for an average of 14 seconds on the hour-long workloads that we experimented with. After the policy decision is made, it takes the meta-scheduler around 2 seconds to transfer state to the next policy. These values are not significant with respect to the average runtime of a job in the cluster (6.5 minutes).

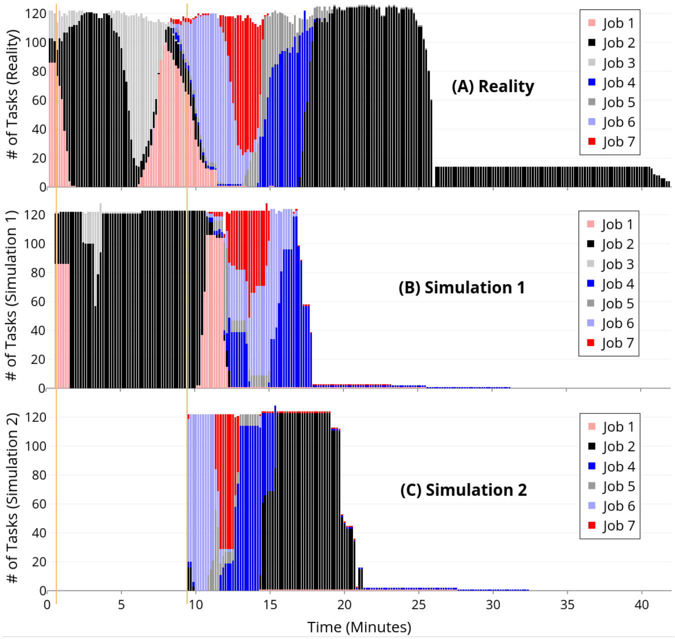

Fig. 3. Visualization of two simulation outcomes in comparison to the real-world execution.

D. Performance Objectives

Throughout the experiments, the performance objective comprises two aspects: the root mean square average of SLO violations (V) and total batch job bounded slowdown (S). Violations are calculated as the time in milliseconds between the finish time of a job and its deadline (or zero if no violation occurred). Bounded slowdown is the ratio between the time spent by a job in the system and the cumulated execution time of its tasks, with a minimum of 10 seconds). We use normalization factors, α and β, to compensate both the value range differences, and the relative importance of each metric to our hypothetical data center customer. Thus, the total performance score for a workload run (either real or simulated) is: $P = \alpha V + \beta S$. We note that it is difficult to find optimal values for the α and β parameters, and, furthermore, these

TABLE III
THE TYPES OF SCORE FUNCTIONS AND THEIR NORMALIZATION FACTORS.

Score Function	Slowdown Normalization Factor (α)	Deadline Violation Normalization Factor (β)	Meta-Scheduler Mode
BC Optimization	1000	1E-6	DYN_BC
DC Optimization	1E-6	1000	DYN_DC
Mixed Optimization	100	1	DYN_MID

parameters could vary depending on the user's requirements. Thus, an in-depth study into the choice of such parameters, or a sensitivity study is out of the scope of this paper.

E. Portfolio Scheduler

The second type of experiments is focused on system performance. Two workloads are run with each of the allocation policies mentioned in Table I, and the resulting compound scores are compared with those of obtained when enabling dynamic policy switching. Three different sets of values are chosen for the α and β normalization factors of the compound score formula to measure the sensitivity of the system to administrator tuning, as seen in Table III. The first favors batch jobs (BC Optimization), the second deadline constrained jobs (DC Optimization), and the third targets a mixture of the two (Mixed Optimization).

Figure 4 plots the results for one of the workloads generated as described in Section VI-B. The lower the score, the better the performance. The chart has been cut off at 2500 for better visualization. The SRTF policy gives the best results when optimizing for batch job slowdown, followed by the batch-favoring DYN_BC mode of the meta-scheduler and then LOCF. The deadline-aware policies, EDLS-Sh and EDLS-Pr give less importance to slowdown in favor of meeting job deadlines, resulting in much higher values for this type of score. As EDLS-Sh reserves an entire section of the cluster for deadline-constrained jobs that remains idle while only batch jobs are running in the system, its results are particularly poor in this respect. When prioritizing for deadline-constrained jobs, however, EDLS_Sh has the best results of all, followed closely by EDLS-Pr. All three dynamic scheduling modes also perform better on this goal than SRTF and LOCF, but there is no clear distinction between the modes. For the mixed optimization goal, DYN_BC seems to perform best, even over DYN_MID. SRTF is again a strong contender, but the deadline-aware policies suffer from the same skew.

In general, our results show that the meta-scheduler cannot out-perform SRTF in slowdown optimization or EDLS-Sh in reducing deadline violations, but it does come close. Interesting to note is that DYN_BC and DYN_MID meta-scheduler modes seem to sometimes perform better for each other's optimization goals, as was the case in the workload presented above. Our findings support **MF2**.

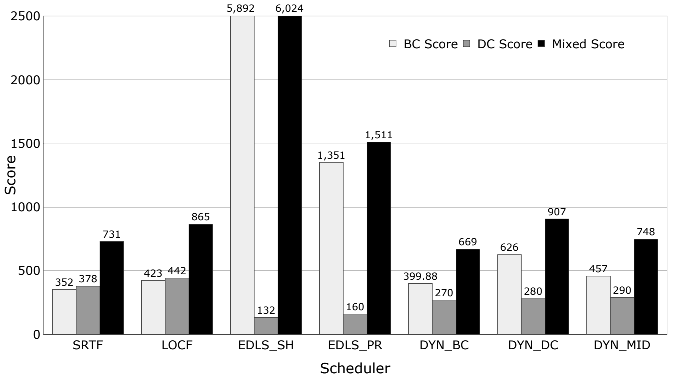

Fig. 4. Performance objective evaluation of POSUM on one workload.

VII. Related Work

While the concept of portfolio scheduling has been used before, previous work is not directly compatibile with the described MapReduce cluster. The approach of van Beek et al. [10] took into consideration only the provisioning of long-running VMs, not the scheduling of individual jobs. Closer to the current model is the work of Deng et al. [30]. However, it differs in both workload type and operation. MapReduce workloads are generally data-intensive, as opposed to scientific computing, which are generally more concerned with CPU-RAM interaction. Furthermore, where in the model of Deng et al., jobs were considered independent and were assigned VMs from the pool, the current model divides each job into several tasks that have dependencies and communication needs between them.

This work also draws inspiration from previous research in the design of the simulator. However, none of the existing solutions fully match our simulation requirements. SLS[4], MRPerf [33] and MRSim [34] have very low-level resource models that result in heavy time costs, while MRSG [35] and YARNsim [36] are designed to run a specific job based on a manually-constructed behavior configuration, and not an entire workload. Mumak, SimMR and Starfish are not compatible with the YARN architecture and require upgrading, but Mumak is available for source modification. However, Mumak is not capable of running a new workload: it only replays previous traces with a given scheduler. In conclusion, we have designed and implemented a new MapReduce simulator for use with the portfolio scheduler, based on techniques and results of previous work.

With regard to the body of existing work for scheduling MapReduce applications (briefly described in Section II), the focus of this work is not devising a single new and effective scheduling policy, but rather adapting existing solutions to work in a complementary fashion so as to cater to different workload compositions and arrival patterns.

VIII. Conclusion and Future Work

Designing one single scheduling policy to achieve compound performance goals is complex and risky. Our system uses the advantages of portfolio scheduling to cater to this use case. We have found that it is best to decouple such a system from the MapReduce framework itself and keep the architecture modular and make processing statistics available

[4] https://hadoop.apache.org/docs/r2.4.1/hadoop-sls/SchedulerLoadSimulator.html

at all times and in a consistent fashion. Also, current simulators are not equipped for online prediction of task behavior on the newer Hadoop stack. We, thus, offer our own implementation of one.

An experimental assessment of the simulator shows that it can follow the general trend of task distribution on the cluster, but its accuracy suffers when reduce task runtimes exhibit considerable variability, calling for a more complex prediction model for this task type that takes this aspect into account.

When evaluating the system as a whole, we have concluded that POSUM cannot out-perform policies that target a single performance objective specifically. However, it does a good job at balancing performance goals, even without flawless task runtime predictions. Another observation is that the end score of the workload run does not always reflect the exact balance of BC job to DC job priority specified in the configuration. Consequently, the evaluation function's scaling factors should be tuned periodically by a human administrator or an adaptive algorithm that can account for changes in the order of magnitude of the constituent evaluation metrics. This is left for future work. As is the exploration of cluster resizing policies in combination with the allocation policies to optimize for a third performance objective: cumulated node lease cost.

Acknowledgments

Work supported by the projects Vidi MagnaData and COMMIT/.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] A. Rowstron, D. Narayanan, A. Donnelly, G. O'Shea, and A. Douglas, "Nobody ever got fired for using hadoop on a cluster," in *Proceedings of the 1st International Workshop on Hot Topics in Cloud Data Processing*, ser. HotCDP '12. New York, NY, USA: ACM, 2012, pp. 2:1–2:5. [Online]. Available: http://doi.acm.org/10.1145/2169090.2169092

[3] C. He, Y. Lu, and D. Swanson, "Matchmaking: A new mapreduce scheduling technique," in *CloudCom*, 2011, pp. 40–47.

[4] P. Nguyen, T. A. Simon, M. Halem, D. Chapman, and Q. Le, "A hybrid scheduling algorithm for data intensive workloads in a mapreduce environment," in *UCC*. IEEE Computer Society, 2012, pp. 161–167.

[5] N. Lim, S. Majumdar, and P. Ashwood-Smith, "A constraint programming based hadoop scheduler for handling mapreduce jobs with deadlines on clouds," in *ICPE*, 2015, pp. 111–122.

[6] G. Ananthanarayanan, S. Kandula, A. G. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri," in *OSDI*, 2010, pp. 265–278.

[7] N. Yigitbasi, K. Datta, N. Jain, and T. Willke, "Energy efficient scheduling of mapreduce workloads on heterogeneous clusters," in *GCM*. ACM, 2011, p. 1.

[8] A. Verma, L. Cherkasova, and R. H. Campbell, "Resource provisioning framework for mapreduce jobs with performance goals," in *Middleware*, 2011, pp. 165–186.

[9] K. Deng, R. Verboon, K. Ren, and A. Iosup, "A periodic portfolio scheduler for scientific computing in the data center," in *JSSPP*, 2013, pp. 156–176.

[10] V. van Beek, J. Donkervliet, T. Hegeman, S. Hugtenburg, and A. Iosup, "Self-expressive management of business-critical workloads in virtualized datacenters," *IEEE Computer*, vol. 48, no. 7, pp. 46–54, 2015.

[11] L. Columbus, "53% of companies are adopting big data analytics, 2017," https://www.forbes.com/sites/louiscolumbus/2017/12/24/53-of-companies-are-adopting-big-data-analytics/#1b06a86c39a1.

[12] G.-H. Kim, S. Trimi, and J.-H. Chung, "Big-data applications in the government sector," *Communications of the ACM*, vol. 57, no. 3, pp. 78–85, 2014.

[13] R. Munné, "Big data in the public sector," in *New Horizons for a Data-Driven Economy*. Springer, 2016, pp. 195–208.

[14] A. Iosup, A. Uta, L. Versluis, G. Andreadis, E. van Eyk, T. Hegeman, S. Talluri, V. van Beek, and L. Toader, "Massivizing computer systems: a vision to understand, design, and engineer computer ecosystems through and beyond modern distributed systems," in *IEEE ICDCS*, 2018.

[15] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi *et al.*, "Spark sql: Relational data processing in spark," in *SIGMOD*, 2015, pp. 1383–1394.

[16] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *MSST*, 2010, pp. 1–10.

[17] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.

[18] Y. Wang and W. Shi, "Budget-driven scheduling algorithms for batches of mapreduce jobs in heterogeneous clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 306–319, 2014.

[19] B. Ghit and D. Epema, "Reducing job slowdown variability for data-intensive workloads," in *MASCOTS*. IEEE, 2015, pp. 61–70.

[20] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types." in *Nsdi*, vol. 11, no. 2011, 2011, pp. 24–24.

[21] J. Tan, X. Meng, and L. Zhang, "Delay tails in mapreduce scheduling," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 1, pp. 5–16, 2012.

[22] Í. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "Greenhadoop: leveraging green energy in data-processing frameworks," in *EuroSys*. ACM, 2012, pp. 57–70.

[23] Y. Chen, A. Ganapathi, R. Griffith, and R. H. Katz, "The case for evaluating mapreduce performance using workload suites," in *MASCOTS*, 2011, pp. 390–399.

[24] Y. Tao, Q. Zhang, L. Shi, and P. Chen, "Job scheduling optimization for multi-user mapreduce clusters," in *PAAP*, 2011, pp. 213–217.

[25] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *EuroSys*, 2010, pp. 265–278.

[26] N. Cheriere, P. Donat-Bouillud, S. Ibrahim, and M. Simonin, "On the usability of shortest remaining time first policy in shared hadoop clusters," in *SAC*, S. Ossowski, Ed. ACM, 2016, pp. 426–431.

[27] K. Kc and K. Anyanwu, "Scheduling hadoop jobs to meet deadlines," in *CloudCom*, 2010, pp. 388–392.

[28] J. Polo, D. Carrera, Y. Becerra, M. Steinder, and I. Whalley, "Performance-driven task co-scheduling for mapreduce environments," in *NOMS*, 2010, pp. 373–380.

[29] X. Dong, Y. Wang, and H. Liao, "Scheduling mixed real-time and non-real-time applications in mapreduce environment," in *ICPADS*, 2011, pp. 9–16.

[30] K. Deng, J. Song, K. Ren, and A. Iosup, "Exploring portfolio scheduling for long-term execution of scientific workloads in iaas clouds," in *SC*, 2013, pp. 55:1–55:12.

[31] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li, and B. Qiu, "Bigdatabench: A big data benchmark suite from internet services," in *HPCA*, 2014.

[32] S. Huang, J. Huang, Y. Liu, L. Yi, and J. Dai, "Hibench: A representative and comprehensive hadoop benchmark suite," in *Proc. ICDE Workshops*, 2010.

[33] G. Wang, A. R. Butt, P. Pandey, and K. Gupta, "A simulation approach to evaluating design decisions in mapreduce setups," in *MASCOTS*. IEEE Computer Society, 2009, pp. 1–11.

[34] S. Hammoud, M. Li, Y. Liu, N. K. Alham, and Z. Liu, "Mrsim: A discrete event based mapreduce simulator," in *FSKD*, 2010, pp. 2993–2997.

[35] W. Kolberg, P. de B. Marcos, J. C. S. dos Anjos, A. K. S. Miyazaki, C. F. R. Geyer, and L. Arantes, "MRSG - A mapreduce simulator over simgrid," *Parallel Computing*, vol. 39, no. 4-5, pp. 233–244, 2013.

[36] N. Liu, X. Yang, X. Sun, J. Jenkins, and R. B. Ross, "Yarnsim: Simulating hadoop YARN," in *CCGrid*, 2015, pp. 637–646.