Optimizing PageRank on Spark

Manish Munikar[1], Ning Li[1]

Abstract

PageRank is widely adopted to quantify the relative importance of nodes in a graph (e.g., to sort web search results). To compute PageRanks of large graphs, many parallel and distributed algorithms have been proposed. Spark is the preferred framework for implementing distributed PageRank because of its in-memory computation and the iterative nature of the algorithm. However, in Spark, it is challenging to optimize PageRank computation according to the specified upper bound of completion time with minimized resource over-provisioning. On the one hand, we require optimized algorithm parameters to facilitate the efficient convergence of PageRank random walk processing. On the other hand, an efficient QoS guarantee mechanism is desired to enforce demanded service level objective (SLO) in terms of completion time for PageRank computation with efficient resource allocation. Our extensive evaluation, driven by three classic graph workloads, demonstrates the effectiveness of the PageRank algorithm's parameter optimization under the control of the proposed QoS guarantee framework (i.e., Para-Opt), which can provide 31.71% reservation of resources on average over all our 36 experimental cases compared with the Spark default resource allocator.

1. Introduction

During the last two decades, PageRank[1] has been widely adopted as a convincing and critical measure to evaluate the importance of nodes in a graph [1, 2]. To speed up its computation, many existing works [1–3] focus on parallelizing the PageRank computation in the parallel computing framework, such as Spark [4]. Compared with Hadoop [5], Spark can exploit the advantages of memory computing to facilitate a faster large-scale data processing by partitioning data of RDD [6]. A certain amount of computing resources (e.g., the specified number of cores and the size of memory) will be allocated to each partition to drive its stage-by-stage data processing.

However, it is challenging to determine the number of partitions of RDD [6] (i.e., *parallelism*[2]) for the PageRank computation, which can decide both its completion time and the amount of occupied resources, especially under service level objective (SLO) constraints of completion time. It is highly desirable to optimize resource allocation in Spark by minimizing parallelism under SLO constraints. Apparently, it requires two aspects of work. On the one hand, we need to optimize PageRank algorithm's parameters to fully utilize the given resources to speed up its data processing. On the other hand, a quality-of-service (QoS) guarantee framework is required to enforce the SLO of completion time with necessary but sufficient computing resources.

For the first work, we implement a distributed version of the PageRank algorithm in Spark and evaluate it using different parameter values and graphs of various domains and sizes. Then, to meet the SLO of completion time for PageRank computation with minimized resource over-provisioning, we propose a QoS guarantee framework (i.e., Para-Opt) that can automate parallelism optimization

[1]Dept. of Computer Science & Engineering, University of Texas at Arlington

[2]In this report, we assume each partition will be assigned an task executor configured with the same amount of computing resources to minimize the difference among tasks' execution time.

for Spark based on a hybrid modeling-and-measurement-based approach. Specifically, Para-Opt first samples the setup time and data-processing time for a fixed number of iterations (e.g., 3) under three different levels of parallelism (e.g., 4, 32, and 128). After that, Para-Opt can model the data-processing speed as a function of parallelism and obtain the optimized parallelism settings according to the SLO of completion time.

To meet the goal, Para-Opt faces two challenges: 1) Para-Opt can only model and estimate based on incomplete running process. Even for the run under a specific level of parallelism, Para-Opt only samples a small part of the entire data processing. It really helps to reduce the sampling complexity to $\mathcal{O}(1)$ by measuring a fixed number of iterations but it makes it hard to accurately predict the data-processing time and the number of iterations required to obtain PageRank values with qualified accuracy. 2) The processing time under different stages (e.g., setup and iterations) changing as a function of parallelism can show distinct trends largely due to their disparate tasks. To address these two challenges, Para-Opt models the entire data processing time by three critical factors, i.e., the setup time, average processing time per iteration, and the number of iterations required to meet the qualified convergence, which is measured by the accumulated error of PageRank value between two successive iterations for all the nodes in a graph (i.e., *asymptotic error* of PageRank) that should be below a specified threshold, i.e., the tolerant error (TE).

Our evaluation, based on three graphs with a distinct number of nodes and edges and disparate outgoing degree distribution, demonstrates the effectiveness and robustness of Para-Opt for minimizing resource over-provisioning for PageRank computation conditioned by the SLO constraint of completion time.

2. PageRank Algorithm

Let $G = \langle V, E \rangle$ be a directed graph where V and E are the set of vertices and edges, respectively. Let $n = |V|$ and $m = |E|$ be the total number of vertices and edges, respectively. We define the PageRank of a vertex i as follows:

$$\pi_i = d \left(\sum_{j \in N_i} \frac{\pi_j}{\deg_{\text{out}}(j)} \right) + \frac{(1-d)}{n} \tag{1}$$

where, N_i is the set of out-neighbors of node i, and $\deg_{\text{out}}(j)$ is the out-degree of node j. The first term is the contribution from backlinks and the second term is the contribution from the *rank source*[3]. The factor d (called the damping factor) controls the relative weight of these two contributions. The damping factor is almost always assumed to be 0.85, as suggested in [1].

Vectorically, it can be represented as follows:

$$\pi = dA\pi + (1-d)E \tag{2}$$

where, A is the normalized adjacency matrix (all non-zero rows sum to one), and E is the rank source vector of size n. For global PageRank computation, all elements of E are set to $1/n$. For *personalized* PageRank, E can be customized to give more weight to certain nodes. This recursive definition of PageRank can be *approximated* iteratively using the power-iteration approach, as shown in Algorithm 1.

[3]Some ranks added to all nodes to offset the ranks lost from the system due to nodes with out-degree of zero.

Algorithm 1: Computing PageRank using power-iteration approach.

```
function ComputePageRank(A):
    π := {1/n for all n nodes};
    E := {1/n for all n nodes};
    repeat
        π_old := π;
        π := dAπ_old + (1 − d)E;
        π := π + (1 − sum(π))E;
    until max(π − π_old) < ϵ;
    return π;
```

2.1. Distributed PageRank

The aforementioned algorithm can be implemented in a distributed manner as follows:

1. Initialize the PageRanks of all the nodes, parallelly, to $1/n$.
2. For each node, compute its contribution to the PageRanks of its outlinks.
3. For each node, sum the contributions towards it to compute the new PageRank.
4. Repeat Steps 2 and 3 until convergence.

Even though the various stages and iterations have a strict causal dependency, the loop over all the nodes, in each stage, can be computed in a parallel/distributed way and the results can be combined later. Particularly, the Steps 2 and 3 can be implemented as a *map* and *reduce* task, respectively, as shown in Algorithm 2.

Algorithm 2: Map-Reduce task for computing PageRank contributions.

```
function map(node, outlinks, rank):
    yield (node, 0);
    outDegree := len(outlinks);
    for neighbor in outlinks do
        yield (neighbor, rank/outDegree);
function reduce(node, contributions):
    yield (node, sum(contributions));
```

3. Para-Opt Design and Implementation

The ultimate goal of Para-Opt is to support SLO-driven resource allocation that targets at not only improving end-users' experience but also minimize resource over-provisioning. From the users' perspective, their individual SLOs in terms of completion time would be effectively guaranteed while the computing resources allocated to their workloads will be optimally configured by Para-Opt only enough for enforcing their SLOs. Thus, Para-Opt can theoretically help Spark to efficiently serve more users with completion time targets.

3.1. Deployment Model and Use Case

Para-Opt suits the deployment model of data-parallel computing framework (e.g., Spark [4]), as illustrated in Figure 1. The deployment process involves three steps: 1) The user needs to submit a PageRank job including its algorithm and data sets to Para-Opt and specify the SLO of completion

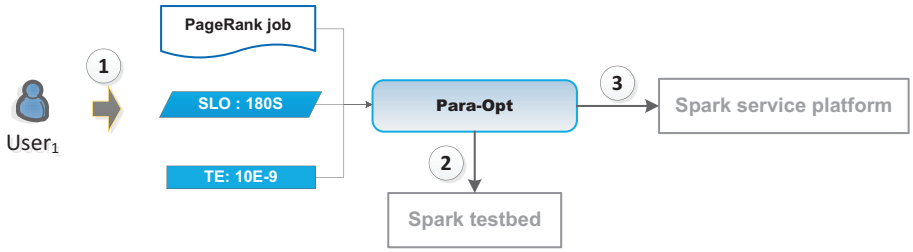

Figure 1: Para-Opt deployment model.

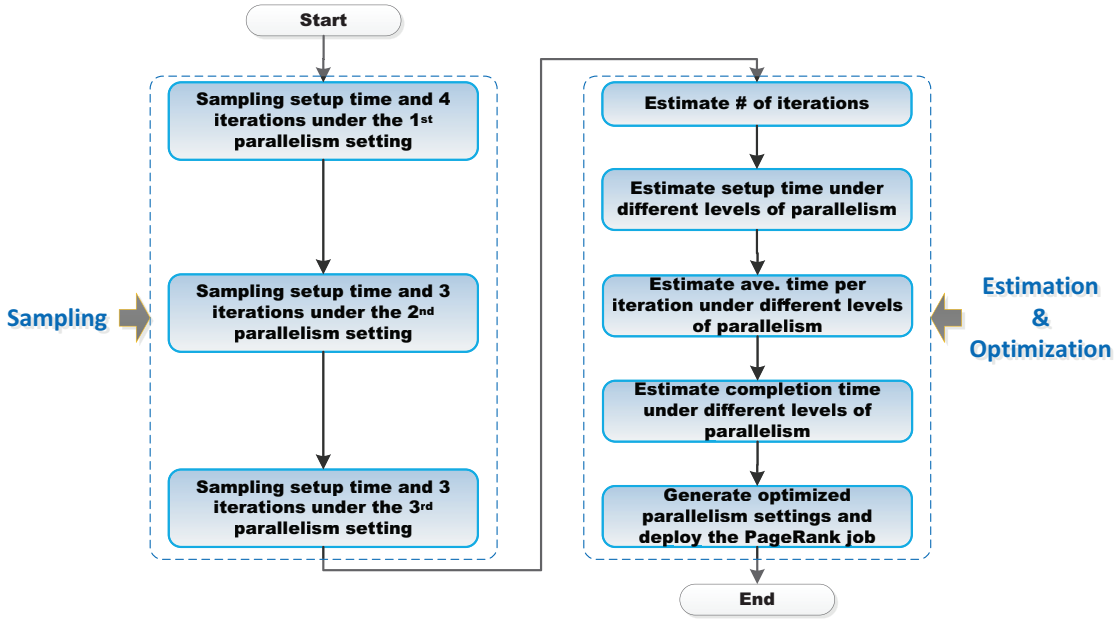

Figure 2: The typical workflow for Para-Opt optimization.

time (e.g., 180 seconds) under a specific TE (e.g., $10E-9$). 2) Para-Opt optimizes the parallelism setting for the job by the necessary sampling and estimation. This offline test running is carried out on a testbed, which should be powerful enough to accommodate an individual workload and typically a subset of the real Spark service platform. For simplicity, our experiments take the real Spark service platform as the testbed. 3) Once optimized parallelism settings have been generated by Para-Opt, the PageRank job will be delivered to the real Spark service platform.

3.2. The Para-Opt Optimization Workflow

The typical workflow of the Para-Opt optimization can be divided into two stages, i.e., sampling stage and estimation/optimization stage, as illustrated in Figure 2. During the sampling stage, Para-Opt is designed to sample setup time, average time per iteration, and asymptotic error of PageRank under three different levels of parallelism settings. For the first level of parallelism, there are 4 iterations for Para-Opt to sample since Para-Opt needs at least three asymptotic error of PageRank value that can only be obtained between two successive iterations while 3 iterations are required to be sampled for the other two because 3 samples are commonly considered the shortest data series to predict their trends. Thus, Para-Opt only samples a fixed-number of iterations for a PageRank job.

In the estimation/optimization stage, Para-Opt first derive the number of iterations under the threshold of TE and then estimate setup time and the average time per iteration respectively based on the corresponding samples obtained under the three levels of parallelism. Based on this work,

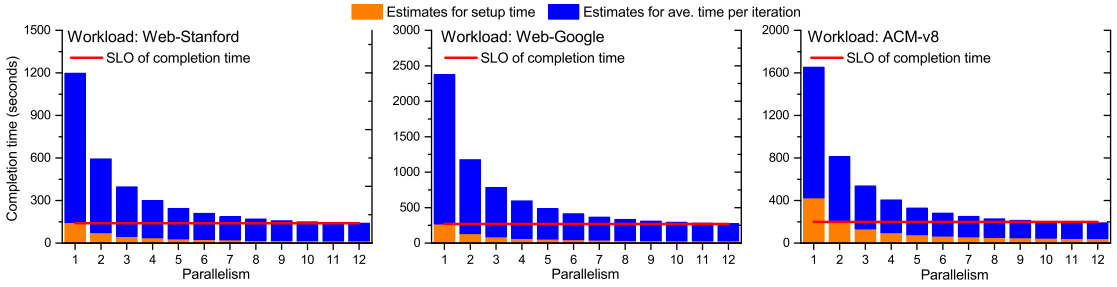

Figure 3: The illustration of Para-Opt optimization.

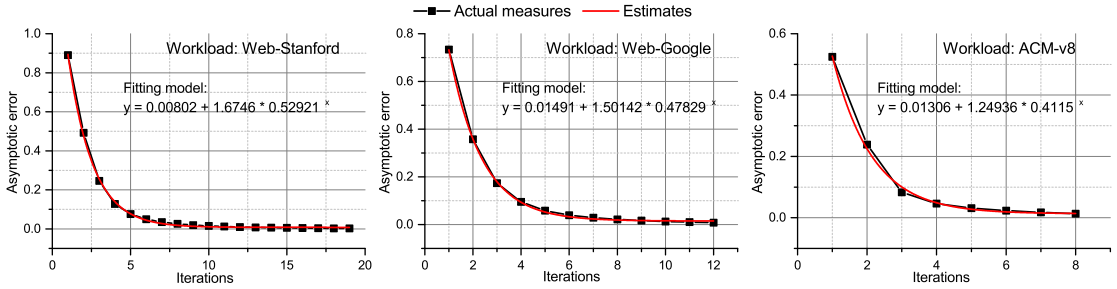

Figure 4: The accumulated pagerank error between two successive iterations obtained under each iteration.

Para-Opt can derive the completion time of the PageRank job as a function of different levels of parallelism. According to the SLO of completion time, Para-Opt will minimize the parallelism but enough to enforce the SLO with satisfactory accuracy, as illustrated by Figure 3. Finally, Para-Opt will submit the job with optimized parallelism settings to the real Spark service platform.

3.3. The Estimation for the Number of Iterations

As one of the three key factors for predicting completion time, it is non-trivial to estimate the number of iterations under a specific TE since it is essentially workload-dependent. Fortunately, based on our experimental observations, we found that it is feasible to use the model of $y = a + be^c$ to accurately estimate the asymptotic error of PageRank value as a function of iterations despite of disparate characteristics of different workloads, as illustrated by Figure 4.

We further adopt *reduced Chi-Square* [7], *residual Sum of Squares* [8] and *adjusted R-Square* [9] that are widely used to evaluate prediction accuracy. The value of adjusted R-Square ranges from 0 to 1, with 1 and 0 representing the highest and lowest fitting accuracy respectively. For reduced Chi-Square and residual Sum of Squares, a value closer to 0 suggests a better fit for the estimation model. As shown in Table 1, the accuracy of the asymptotic error estimation model is adequate for Para-Opt to predict the number of iterations under a specific TE.

To derive the asymptotic error estimation model, Para-Opt samples three asymptotic error of PageRank value under any parallelism and then use them as input to pin down the three parameters of the model. Finally, the number of iterations under a specific TE can be accurately estimated, as demonstrated by Figure 7.

Data sets	Reduced Chi-Square	Residual Sum of Squares	Adj. R-Square
Web-Stanford	$4.99E-05$	$7.99E-04$	0.9990
Web-Google	$2.53E-05$	$2.27E-04$	0.9994
ACM-v8	$1.06E-04$	$5.31E-04$	0.9967

Table 1: The estimate accuracy for asymptotic error.

3.4. The Estimation for setup time and the time of iterations

Although the setup time often occupies a small portion of the completion time, it often exhibits distinct function of parallelism from the time of iterations. Thus, we need to estimate them separately. To minimize the samples required to derive setup time and the average time per iteration respectively as a function of parallelism, we adopt Eq. (3) proposed by [10] to estimate the processing speed in the setup stage and the iteration stage respectively, which is defined as the reciprocal of the processing time, under different levels of parallelism.

$$\lambda(P) = \frac{P \cdot \lambda(P_0)}{P_0 + B \cdot P_0 \cdot (e^{-\gamma P_0} - e^{-\gamma P}) \cdot (P - 1)}. \tag{3}$$

To pin down the parameters in Eq. (3), including $\lambda(P_0)$, P_0, B, and γ, we resort to a measurement approach by sampling three processing speed under different levels of parallelism. $\lambda(P_0)$ denotes the processing speed under the parallelism of P_0 that can afford a high ratio of parallel processing, typically close to 100%. Another sample $\lambda(P_2)$ doesn't need to constrain P while $\lambda(P_1)$ can be sampled at a medium value. After that, we can resolve the parameters B and γ and finally derive the estimation models for setup time and the time of iterations separately. By default, Para-Opt adopts 4, 32, and 128 for P_0, P_1, and P_2 respectively for the following experiments and there are totally 10 iterations that are required to be sampled under these three levels of parallelism.

4. Evaluation

Test Environment: A server with Spark 3.1.2 is equipped with Intel Xeon X5650 (12 cores) and 48 GB RAM. By default, 1 CPU core and 1 GB RAM are allocated to each RDD partition. Thus, the number of partitions (i.e., parallelism) will determine the amount of computing resources allocated to the workload.

Workloads: We adopt three representative workloads (i.e., Web-Stanford [11], Web-Google [12], and ACM-v8 [13]) with distinct distributions of outgoing degrees and different numbers of nodes and edges, as listed in Table 2.

Dataset	# of nodes	# of edges
Web-Stanford[11]	282K	2.31M
Web-Google [12]	875K	5.10M
ACM-v8 [13]	1.37M	8.65M

Table 2: Datasets used for experiments.

Objectives and Methodology: We aim to evaluate the effectiveness and robustness of the Para-Opt optimization under different TE values for three distinct workloads. Specifically, we first assess the estimation accuracy for the number of iterations, setup time, and average time per iteration respectively, which are the three key factors to predict the completion time under different levels of parallelism. Second, we verify the effectiveness of Para-Opt to minimize resource allocation under the constraint of the SLO of completion time. To this end, we set three different TE values (i.e., 10e–8, 10e–9, and 10e–10), under each of which we specify 4 distinct SLOs that can be afforded by the given resources.

4.1. Speedup with Parallelism

To explore the impact of parallelism on speeding up PageRank computation, we measure the completion time under different levels of parallelism for three distinct data sets, as listed in Table 2. As shown in Figure 5, the trend of completion time as a function of parallelism obtained for 3 data sets can be broadly divided into 3 stages. During the first stage, the completion time decreases

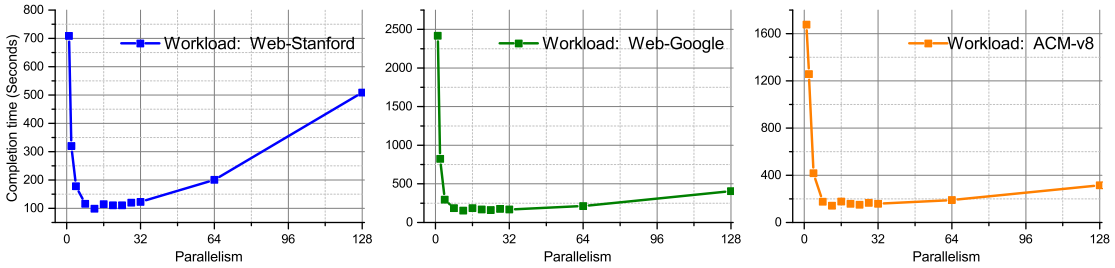

Figure 5: The impact of Spark parallelism settings on speeding up PageRank computation.

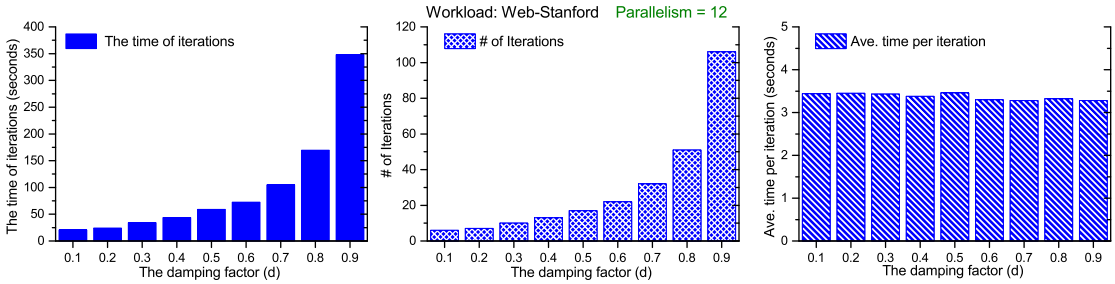

Figure 6: The impact of damping factor on PageRank iterations.

rapidly with parallelism increasing. And then, the completion time can be kept at a low level during the 2nd stage. After that, the completion time will be lengthened again. Note that the lowest completion time happens around the parallelism of 12, which is the number of CPU cores of the testbed, for all the three data sets. This means a number of partitions exceeding the the number of cores wouldn't significantly speed up the PageRank computation due to resource contention. Thus, Para-Opt aims to optimize the parallelism setup within the range from 1 to the number of cores on the testbed for the targeted workload.

4.2. Effect of Damping Factor

The damping factor (d) can be considered as the *click-through rate*, which is the probability of a walk on the graph to follow the links randomly. The compliment ($1 - d$) is the probability of the walk to jump to a random node. The value of d must be strictly between 0 and 1. On one hand, if d is too small, some nodes may not be visited. On the other hand, if d is close to 1, the *sink* nodes—that have no outgoing edges—will keep accumulating all the ranks in the system. We experimented with different values of d to see how it affects the convergence time of PageRank. The results are shown in Figure 6. As we increase the damping factor, the time (or the number of iterations) required to reach the same level of convergence increases exponentially. However, small value of d is also not desirable due to aforementioned reasons. d is almost always set to a value of 0.85, which is estimated from the frequency that an average surfer uses his or her browser's bookmark feature [14]. However, some researchers claim that a value of 0.7 can also give very similar Top 25 results while converging 25–30% faster [15].

4.3. Para-Opt Estimation for Three Factors

As shown in Figure 7, we can observe that Para-Opt can effectively estimate the number of iterations required to meet different TE values. The average absolute estimation error over all the 36 cases is 8.54%. Since parallel processing essentially cannot impact the convergence of PageRank computation, the number of iterations with the same TE configuration but under different SLOs

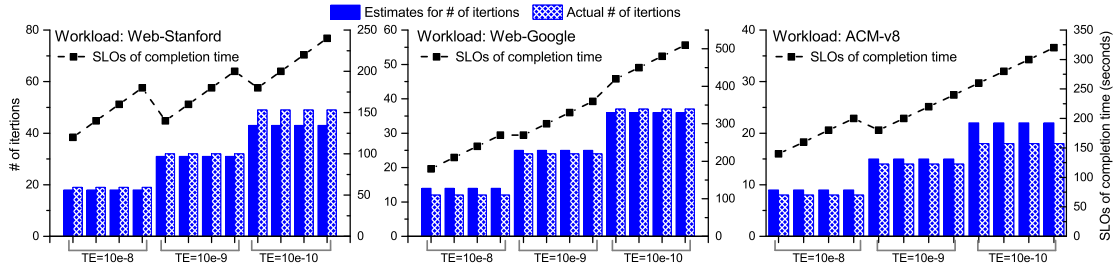

Figure 7: The estimation for the number of iterations.

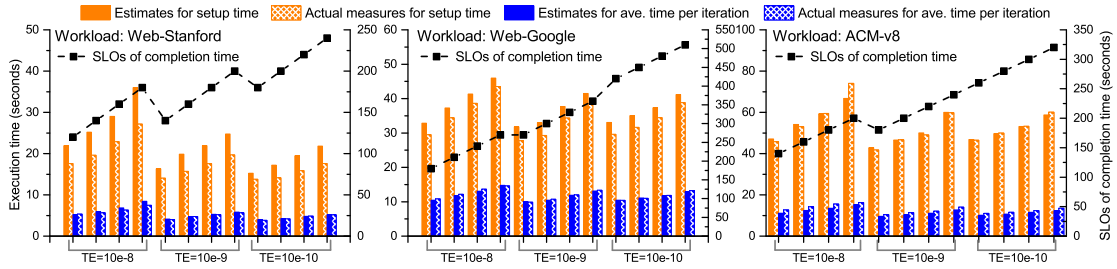

Figure 8: The estimation for setup time and average time per iteration.

basically keeps the same value. Similarly, we also find that Para-Opt can precisely predict the trend of setup time and the average time per iteration as the SLO increases, as shown in Figure 8. The average absolute estimation error over all the cases for the average time per iteration is 4.96% while that obtained for the setup time is 11.74%.

4.4. Para-Opt Optimization

This section aims to assess the estimation error and the control error under the Para-Opt optimization as well as verify the effectiveness of minimizing resource over-provisioning. As shown in Figure 9, we observed that Para-Opt can accurately estimate completion time and effectively enforce the SLOs under different TE configurations. Specifically, the average absolute control error, which is defined as the difference between the actual completion time obtained under the Para-Opt optimization and the SLO targets averaged over all our 36 experimental cases is 6.07% while the average estimation error for the completion time is 5.23% across all the cases. Further, from the Figure 10, we find that the resource provisioning governed by the parallelism settings can effectively adapt to the change of the SLOs of completion time. In the view of accurate SLO enforcement demonstrated in Figure 9, any further reduction in resource provisioning will increase the control error in a unpredictable manner and thus violate SLOs. Thus, parallelism settings optimized by Para-Opt have minimized the resource provisioning conditioned by the SLOs of completion time. Specifically, Para-Opt provides 31.71% reservation of resources on average over the Spark default resource allocator that will statically assign all the cores to the application.

5. Related Work and Discussion

Existing solutions focusing on predicting the duration or ensuring the SLO of completion time for parallel computation workloads running on Spark can be roughly divided into four categories according to their distinct training efforts or sampling cost. The first is machine learning based approaches, e.g., Gulino et al. [20], Wang et al. [21], and Javaid et al. [22], which typically need a high training complexity and thus make it hard to be widely accepted as low-cost solutions.

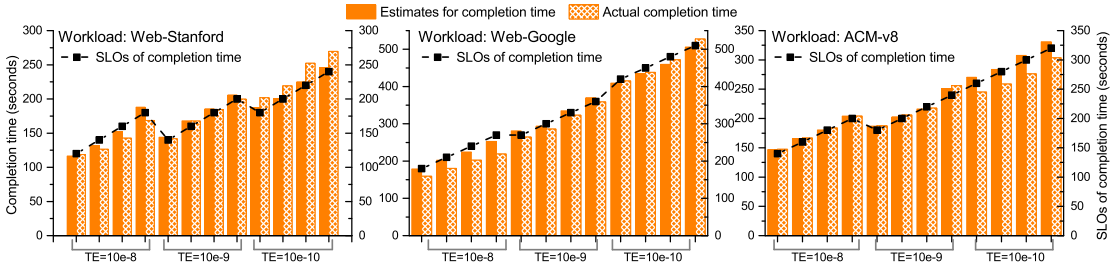

Figure 9: The estimation error and the control error for different SLOs of completion time under three TE configurations.

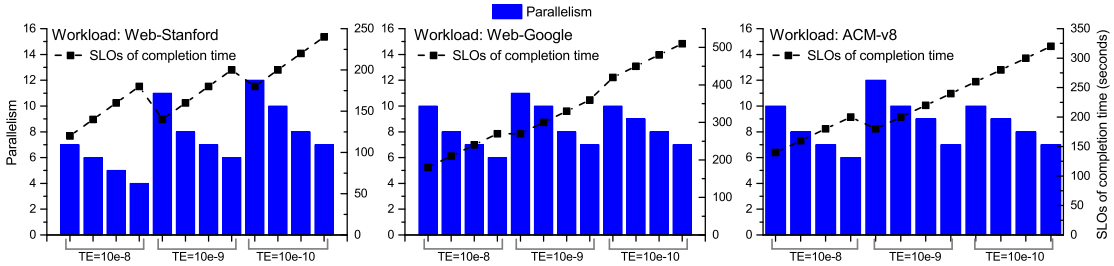

Figure 10: The optimized parallelism settings for different SLOs of completion time under three TE configurations.

The second resorts to trace-based profiling or brute-force experimentation, e.g., FPN [16], which requires to collect sufficient records to characterize the variation of parallel data processing for a high prediction precision and thus can lead to a high preliminary processing and profiling overhead. As a result, to further improve the precision of estimation and control model with low profiling cost, the third type of approaches, e.g, dynaSpark [18], adopts intrusive implementation by modifying Spark system to realize per-stage deadline. This type of solutions, although having a low estimation/control error, is considered unpractical for production platforms. The fourth type of solutions make more efforts on cutting down the profiling cost required for estimation/optimization by sampling at some specific levels of parallelism [17] or tailoring input dataset [19]. However, they still need to profile all the iterations for each run. Thus, the profiling complexity is $O(n)$ where n refers to the total number of iterations for each run. In contrast, as listed in Table 3, based on precisely estimating the number of iterations and stage-based execution time prediction for PageRank computation, our solution only needs to sample a fix-number of iterations for different workloads, thus reducing the profiling complexity to $O(1)$. We also observe from Table 3 that Para-Opt's estimation/control error is close to or even better than Doppio [17] or PERIDOT [19]. In addition, Para-Opt is potential to adopt the technique of tailoring input dataset [19] to further reduce the sampling cost.

6. Conclusion

In this project, we implemented a distributed PageRank algorithm in Spark, evaluated the performance impact of various parameters, and proposed a way to optimize Spark parallelism settings under the control of a QoS framework (i.e., Para-Opt). Para-Opt can optimize Spark configurations for the targeted workload conditioned by the SLO of completion time with $O(1)$ sampling complexity. Our evaluation, driven by distinct data sets, verifies Para-Opt's effectiveness on minimizing resource over-provisioning under the SLO constraints of completion time for different levels of tolerable asymptotic error.

Approach	Implementation	Sampling	EST Err	CTL Err
FPN [16]	Non-intrusive EST	Trace-driven profiling	Ave. = 10%	—
Doppio [17]	Non-intrusive EST+OPT	4 * (setup + # of iterations)	Ave. = 5.2% (PageRank)	—
dynaSpark [18]	Intrusive EST+OPT	Dynamic	—	Med. = 0.9% (PageRank)
PERIDOT [19]	Non-intrusive EST+OPT	Tailored datasets	Ave. = 7.7%	—
Para-Opt	Non-intrusive EST+OPT	3 * setup + 10 * iteration	Ave. = 5.2% Med. = 3.4% (PageRank)	Ave. = 6.1% Med. = 4.8% (PageRank)

Table 3: The relevant solutions of Para-Opt characterized by different implementation and sampling strategies and modeling accuracy in terms of estimation error (or EST Err) and control error (or CTL Err), which mainly focus on PageRank computation on Spark if any.

References

[1] L. Page, S. Brin, R. Motwani, T. Winograd, The pagerank citation ranking: Bringing order to the web, Tech. rep., Stanford InfoLab (1999).

[2] A. D. Sarma, A. R. Molla, G. Pandurangan, E. Upfal, Fast distributed pagerank computation, in: Proceedings of the International Conference on Distributed Computing and Networking, 2013.

[3] Y. Zhu, S. Ye, X. Li, Distributed pagerank computation based on iterative aggregation-disaggregation methods, in: Proceedings of the ACM international conference on Information and knowledge management (CIKM), 2005.

[4] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoic, Spark: Cluster computing with working sets, in: In 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10), 2010.

[5] Hadoop, http://hadoop.apache.org (2022).

[6] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. Mccauley, M. J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: Proceedings of the USENIX conference on Networked systems design and implementation (NSDI), 2012.

[7] Reduced chi-squared statistic, https://en.wikipedia.org/wiki/Reduced_chi-squared_statistic (2021).

[8] Residual sum of squares, https://en.wikipedia.org/wiki/Residual_sum_of_squares (2013).

[9] Coefficient of determination, https://en.wikipedia.org/wiki/Coefficient_of_determination#Adjusted_R2 (2022).

[10] N. Li, H. Jiang, H. Che, Z. Wang, M. Q. Nguyen, Improving scalability of database systems by reshaping user parallel I/O, in: Proceedings of European conference on Computer systems (EuroSys), 2022.

[11] Web-Stanford, https://snap.stanford.edu/data/#web (2002).

[12] Web-Google, https://snap.stanford.edu/data/web-Google.html (2002).

[13] ACM-v8, https://lfs.aminer.org/lab-datasets/citation/citation-acm-v8.txt.tgz (2002).

[14] PageRank: Damping Factor, https://en.wikipedia.org/wiki/PageRank#Damping_factor.

[15] A. K. Srivastava, R. Garg, P. Mishra, Discussion on damping factor value in PageRank computation, International Journal of Intelligent Systems and Applications 9 (9) (2017) 19–28.

[16] E. Gianniti, A. M. Rizzi, E. Barbierato, M. Gribaudo, D. Ardagna, Fluid petri nets for the performance evaluation of mapreduce and spark applications, ACM SIGMETRICS Performance Evaluation Review 44 (4) (2017) 23–36.

[17] P. Zhou, Z. Ruan, Z. Fang, M. Shand, D. Roazen, J. Cong, Doppio: I/O-aware performance analysis, modeling and optimization for in-memory computing framework, in: Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2018.

[18] L. Baresi, A. Leva, G. Quattrocchi, Fine-grained dynamic resource allocation for big-data applications, IEEE Transactions on Software Engineering 47 (8) (2021) 1668–1682.

[19] S. Shah, Y. Amannejad, D. Krishnamurthy, M. Wang, Peridot: Modeling execution time of spark applications, IEEE Open Journal of the Computer Society 47 (2) (2021) 346–359.

[20] A. Gulino, A. Canakoglu, S. Ceri, D. Ardagna, Performance prediction for data-driven workflows on apache spark, in: IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2020.

[21] G. Wang, J. Xu, B. He, A novel method for tuning configuration parameters of spark based on machine learning, in: IEEE International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016.

[22] M. U. Javaid, A. A. Kanoun, F. Demesmaeker, A. Ghrab, S. Skhiri, A performance prediction model for spark applications, in: In International Conference on Big Data, 2020.