Poster: Searching for High-performing Software Configurations with Metaheuristic Algorithms

Chong Tang, Kevin Sullivan, Baishakhi Ray
University of Virginia, Charlottesville, VA USA
{ct4ew,sullivan,rayb}@virginia.edu

ABSTRACT

Modern systems often have complex configuration spaces. Research has shown that people often just use default settings. This practice leaves significant performance potential unrealized. In this work, we propose an approach that uses metaheuristic search algorithms to explore the configuration space of Hadoop for high-performing configurations. We present results of a set of experiments to show that our approach can find configurations that perform significantly better than defaults. We tested two metaheuristic search algorithms—coordinate descent and genetic algorithms—for three common MapReduce programs—Wordcount, Sort, and Terasort—for a total of six experiments. Our results suggest that metaheuristic search can find configurations cost-effectively that perform significantly better than baseline default configurations.

CCS CONCEPTS

• **Software and its engineering → Software performance**; **Search-based software engineering**;

KEYWORDS

configuration, metaheuristic, optimization

ACM Reference Format:
Chong Tang, Kevin Sullivan, Baishakhi Ray. 2018. Poster: Searching for High-performing Software Configurations with Metaheuristic Algorithms. In *Proceedings of 40th International Conference on Software Engineering Companion, Gothenburg, Sweden, May 27-June 3, 2018 (ICSE '18 Companion)*, 2 pages.
https://doi.org/10.1145/3183440.3195006

1 INTRODUCTION

Many software systems are highly configurable and can be tailored to meet different needs in different environments and to be expected perform well when doing so. In practice, people often find it hard to understand how to take advantage of these rich degrees of configurability. They often just accept the default configurations, leaving significant performance improvement potential unrealized.

Configuration spaces are usually vast, with multiple possible values for each of dozens or hundreds of parameters. They are often not just flat vectors of numerical values, but rather are complex data structures, with fields of multiple types. We observe dependent substructures among parameters, e.g., where setting one parameter to *true* enables a subsystem requires additional parameter settings be provided for that subsystem. People often don't know key properties of configuration-to-performance functions: e.g., how parameters interact, whether properties analogous to convexity and linearity apply, etc. Usually, one has to profile a system given a configuration to get its performance. Profiling is generally expensive, so we need for ways to find high-performing configurations without computing the objective function too many times.

Much previous work focuses on learning general prediction models and then using them in finding good configurations. To learn models, one must profile a system under varying configurations and then generalize from the resulting samples. The challenge is to learn suitable models with as few profiling runs as possible. Siegmond et al. focus on reducing the cost to learn accurate models in the presence of feature interactions [5, 6]. Nair et al. [3] aim to relax the requirement for accurate predictions of performance to one for accurate *ordering* by performance.

In this paper, we present an approach that uses metaheuristic search algorithms to find much better configurations for Hadoop than baseline defaults. We choose Hadoop because it is a fundamental component in today's big-data infrastructures. Hadoop has a complicated configuration space, where more than nine hundred configuration parameters distributed across multiple subsystems. Besides, recent research [4] shows that users barely change default configuration for performance purpose.

We present data from six experiments on three jobs from the HiBench [2] (Wordcount, Sort, and Terasort) and two search algorithms (coordinate descent and genetic algorithms). Our approach found configurations that improve performance by amounts ranging from 6.5% to 56.7%, with configurations found using small instances sometimes providing even greater improvements as input size is scaled by factors of 10X and 100X. We detail these results in Section 4.

2 APPROACH

Our approach has several parts. First, we use domain knowledge to eliminate parameters that are not relevant to the objective performance (CPU time in this work). Second, we discretize the space by defining sampling methods for each parameter according to their data type. Boolean parameters are sampled for true and false values. Numerical parameters have a few values less and greater than their defaults. For string-valued parameters (e.g., as Java virtual machine settings), we provide lists of alternative values. Third, we undertake an automated meta-heuristic algorithm to search high-performing configurations in the reduced space.

Our sampling+profiling framework accepts a discretized configuration space model as input. It supports an arbitrary metaheuristic search algorithms and can accommodate varying discretization and sampling strategies. In this work, we used two search algorithms: cyclic coordinate descent and a genetic algorithm. That involves sampling of system configurations, re-configuring a Hadoop cluster, and then profiling the CPU time by running a benchmark job. We run the sampling-and-benchmarking procedure until a convergence criterion is met or the search budget is exhausted. We return the best configuration found by the search procedure.

Coordinate descent [1] is an iterative optimization method in which each iteration of an inner loop optimizes the objective function in one variable and the outer loop iterates through the variables. This algorithm empirically performs well for many practical problems. Its benefits include being easy to understand and implement, often converges quickly, and admits different optimization strategies for individual parameters. A weakness is that its innermost loop does not account for parameters interactions. It is known that configuration parameters can interact, thereby defeating naive optimization methods. Genetic algorithms [7] are heuristic search algorithms that account for interactions more directly. Such an algorithm starts with *N* first generation subjects, which are configurations in our case. It measures the quality of these subjects and selects good ones. It then generates offspring subjects by mutating good parents. This procedure repeats until a satisfying solution is found or the budget is exhausted.

3 EXPERIMENTAL SETUP

The experimental cluster we used for this work was constructed with the AWS EMR service. The EMR cluster we built has one master node and four slave nodes, all running *m4.xlarge* EC2 instances. The Hadoop version we used is *V2.7.3*. The default configurations for different sized EMR clusters are different on AWS, leading us to infer that these are already reasonably well-designed, so we do not measure our approach against an artificially degraded baseline.

In total, we choose 109 performance related parameters. We generate the values of a parameter according to its data type. For a boolean parameter, the possible values are *[True, False]*. For categorical parameters, we grab their values from the official documents. For numeric parameters, we define two smaller and two larger values around the default one with a Δ of 10% of the default value. Some numeric parameters have 0 as their default values. In this case, we referred the official documents to define a list of alternative values that meet its semantic meaning.

We selected WordCount, Sort, and TeraSort provided by HiBench as our benchmark jobs. We conduct the search procedure with *Small* input data defined by HiBench, and verify the resulting configurations with *Large* and *Huge* data sets.

4 RESULTS

This section presents our experimental results. Table 1 shows the performance improvement for each of the three sizes of input and jobs. The results show that our approach can find much better configurations than the baseline. Both algorithms found configurations that improve the CPU time range from 6.5% to 56.7%. We can also tell that genetic algorithm performs better than coordinate descent, as the configurations found can yield better performance in all three input data sets.

Table 1: Performance improvement by jobs and datasets

	Small	**Large**	**Huge**
	Coordinate Descent		
WordCount	12.1%	1.5%	0%
Sort	20.0%	-16%	-13.8%
TeraSort	56.7%	70%	71.5%
	Genetic Algorithm		
WordCount	6.5%	35.3%	12.8%
Sort	31.9%	44.4%	58.8%
TeraSort	34.3%	73.7%	73.1%

Table 2 compares the cost of our algorithms regarding the number of dynamic profiling operations. Take the *Sort* and **GA** as an example, *1452/1500/G2* means that the 1452*th* of 1500 sampled configuration is the best solution. *G2* means that we found the best one in the second generation. The lower half of this table shows the dollar cost of running experiments to explore configurations for corresponding jobs on AWS.

Table 2: Exploration cost on our experiment cluster

Algorithm	**WordCount**	**Sort**	**TeraSort**
CD	536/686/C2	977/1030/C3	582/686/C2
GA	564/1500/G1	1452/1500/G2	1215/1500/G2
Money Cost			
CD	\$41.85	\$82.09	\$62.97
GA	\$91.50	\$119.55	\$137.7

5 CONCLUSIONS

This paper shows that heuristic search methods can find configurations that provide significant performance improvements for common Hadoop jobs. We also show that this approach is cost-effective. With just tens/around one hundred dollars, we found configurations that can improve job performance by more than 50% in average. We conclude that the experimental data we have reported show that our approach is one worth developing and evaluating further.

REFERENCES

[1] Adrian A Canutescu and Roland L Dunbrack. 2003. Cyclic coordinate descent: A robotics algorithm for protein loop closure. *Protein science* 12, 5 (2003), 963–972.
[2] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. 2010. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*. IEEE, 41–51.
[3] V. Nair, T. Menzies, N. Siegmund, and S. Apel. 2017. Using Bad Learners to find Good Configurations. *ArXiv e-prints* (Feb. 2017).
[4] Kai Ren, YongChul Kwon, Magdalena Balazinska, and Bill Howe. 2013. Hadoop's adolescence: an analysis of Hadoop usage in scientific workloads. *Proceedings of the VLDB Endowment* 6, 10 (2013), 853–864.
[5] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. 2015. Performance-influence models for highly configurable systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 284–294.
[6] Norbert Siegmund, Sergiy S. Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. 2012. Predicting Performance via Automated Feature-interaction Detection. In *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*. IEEE Press, Piscataway, NJ, USA, 167–177.
[7] Darrell Whitley. 1994. A genetic algorithm tutorial. *Statistics and computing* 4, 2 (1994), 65–85.