System Performance Optimization via Design and Configuration Space Exploration

Chong Tang
Department of Computer Science
University of Virginia
ct4ew@virginia.edu

ABSTRACT

The runtime performance of a software system often depends on a large number of static parameters, which usually interact in complex ways to carry out system functionality and influence system performance. It's hard to understand such configuration spaces and find good combinations of parameter values to gain available levels of performance. Engineers in practice often just accept the default settings, leading such systems to significantly underperform relative to their potential. This problem, in turn, has impacts on cost, revenue, customer satisfaction, business reputation, and mission effectiveness. To improve the overall performance of the end-to-end systems, we propose to systematically explore (i) how to design new systems towards good performance through design space synthesis and evaluation, and (ii) how to auto-configure an existing system to obtain better performance through heuristic configuration space search. In addition, this research further studies execution traces of a system to predict runtime performance under new configurations.

CCS CONCEPTS

• **Software and its engineering** → *Software design tradeoffs*; *System administration*; *Search-based software engineering*;

KEYWORDS

Performance Optimization, Design Space, Configuration Space, Performance Prediction

ACM Reference Format:
Chong Tang. 2017. System Performance Optimization via Design and Configuration Space Exploration. In *Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4–8, 2017 (ESEC/FSE'17),* 4 pages.
https://doi.org/10.1145/3106237.3119880

1 INTRODUCTION AND PROBLEMS

Software performance is critical in many aspects of our lives. Slow applications can cause revenue loss, deterioration in customer satisfaction and brand reputation, and mission effectiveness. However, it is difficult to design and configure a system to achieve its potential because the runtime performance is determined by a large number of static parameters. For example, big data systems like Hadoop present hundreds of configuration parameters to engineers. Many of them affect performance, and some interact in complex ways. Engineers often have to accept the default settings which usually lead to significantly low performance compare to their potential. In this research, we propose and evaluate two approaches for dealing with two sub-problems in system design and configuration respectively.

Problem #1: It is difficult to design a system with desired performance. Engineers often develop a software system based on a specification that specifies various requirements and properties. However, such specifications are usually incomplete [4, 6] with respect to the totality of desirable properties. For example, while an object-oriented data model constrains the behavior of a persistent data store, it does not uniquely determine the database schema (design); nor does it express preferences for performance properties like read and write performance, which can vary greatly with the choice of design. Unspecified properties create degrees of freedom that give rise to design spaces [10, 20, 21], where sets of designs satisfy specified properties but vary in unspecified ones like scalability, security, and performance. Existing methods often produce only single design in such a space and hope the resulting system will be good enough.

This research proposes a design space analysis approach to find designs with valued runtime performance. First, we synthesize the entire design space and test cases from a given incomplete specification. Second, we dynamically profile this design space with synthesized test cases to obtain the runtime performance of all designs. Third, we find out the Pareto-optimal designs based on the profiling results. We plan to evaluate this approach in object-relational mapping (ORM) domain. We will compare the time and space performance of SQL schemas produced by our approach with those created by Rails and Django.

Problem #2: It is difficult to configure a system to achieve better performance. For most legacy systems, a more common question is how to configure them to improve performance. Our approach in systems design may not be applied directly to large-scale legacy systems because: 1) they often lack integrated specifications, and 2) changing their core designs is not always feasible. Typically, these systems come with a large number of configuration parameters. One can turn on/off system features or to fine tune a feature by adjusting parameter settings. Such flexibility enables users to tailor a system to meet their needs. However, it also brings uncertainty to the system performance. There are two challenges to configure a system to achieve expected performance. First, the size of a configuration space [8, 26] is huge because it increases

ESEC/FSE'17, September 4–8, 2017, Paderborn, Germany

ACM ISBN 978-1-4503-5105-8/17/09…$15.00
https://doi.org/10.1145/3106237.3119880

exponentially in proportion to the number of parameters. Second, configuration parameters often interact in complex ways. It is difficult to understand the structure of a configuration space, and thus exploring it is intractable in any conceivable cases.

The first approach we propose is to learn good configurations from system usage patterns. Daily operation of a system creates a large amount of usage data which contains various configurations and corresponding resulting performance. We can build system performance prediction models by learning from these traces. In practice, it's possible that users never get close to good configurations and thus create biased training data. Therefore, we propose an alternative approach to explore the configuration space of a system to find out good settings using heuristic search algorithms. We plan to evaluate our approach by applying it to the Hadoop system. A current research [19] shows that its usage is still in adolescence. There is a large room for performance improvement. However, users rarely tune it for performance but only for errors. In this context, even moderate performance improvement than engineers can achieve today has a huge impact.

2 RELATED WORK

This section reviews related work in design synthesis, performance prediction, and configuration space exploration.

Design Space Synthesis. There is a large body of research on synthesis techniques. Dang [13] provided a tool for embedded software synthesis. Neema et al. [17] proposed a suite to synthesize an integrated embedded system from multiple models to meet "set design goals and performance targets". Andersen [1] provided a framework for mathematical problems synthesis. Le [14] provided a framework to synthesize program for data extraction from different kinds of sources. Gupta [9] provided a high-level synthesis framework to transfer a behavioral description in ANSI-C to register-transfer level VHDL with parallel compiler transformation technique. Different from all these techniques, our approach tackles the automated design space analysis through synthesizing spaces of design alternatives and common test cases which enables us to evaluate designs with the same amount of business data.

Performance Prediction Model. Zhang et al. [28] assumes all options are independent and can be converted to boolean variables. They thus formulated the problem of performance prediction as learning the Fourier coefficients of a function. However, converting arbitrary options to boolean values could largely increase the feature space. Besides, options usually interact with each other. Siegmund et al. [22] used step-wise linear regression to derive a performance influence model for a given configurable system. They added features hierarchically to the learning algorithm. However, due to interactions among them, selecting one/few options while ignoring others does not represent the real scenario. In this research, we first use the domain knowledge to select options that have somewhat influence on performance. Then, we then reduce the feature space based on the actual meaning and the relationships which we will discuss in section 3.2.

Heuristic Configuration Space Search. In embedded hardware design domain, Palermo et al. [18] adopted heuristic searching algorithms to find an approximation of the Pareto-optimal configurations to achieve better energy and delay trade-offs. Zhang et al. [27] used tuning heuristic to find better settings for single-level configurable cache. In robotic motion planning domain, Jaillet et al. [12] used stochastic sampling for path planning on given configuration space costmaps. Recently, many works in the software engineering field have shown that heuristic searching algorithms can be leveraged to solve various problems. Weimer et al. [24] used heuristic search techniques to help find candidate code snippets to repair buggy code. Some other works [15, 25] used search-based techniques in software testing. These works share with ours the common insight that given the vast size and complex structure of configuration spaces, heuristic sampling techniques seem promising to reveal high-value solutions. While most of them focus on system correctness, this work focuses on system performance.

3 APPROACH

To solve the performance issue in system design, we propose an approach that combines design space synthesis and dynamic evaluation to select desired designs. For the performance issue in system configuration, we propose a machine learning based approach and a heuristic search approach to obtain desired configuration settings.

3.1 ORM Design Space Exploration

Figure 1 shows the overall structure of the proposed approach. There are four main steps: formal schema and test cases synthesis, distributed concrete schema creation and test cases realization using MapReduce framework, distributed performance profiling, and Pareto-optimal schema selection.

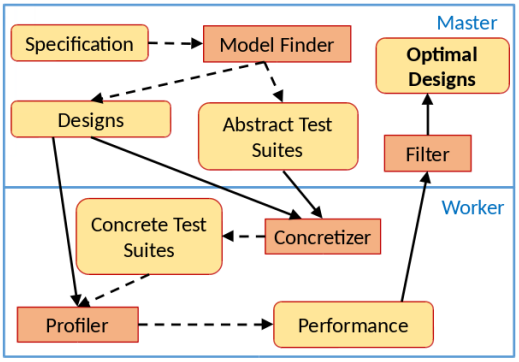

Figure 1: Design space exploration approach

First, the *Model Finder* solves a specification to synthesize all formal schemas and test cases. Second, the *Concretizer* generates real SQL schema. In abstract, there is only one set of test cases synthesized for all schemas. In order to successfully run them, we concretize test cases based on the structure of different schemas. Third, the *Profiler* runs test cases on all schemas and collect the performance data. There are three criteria we plan to collect: data storing time, retrieval time, and disk space consumption. We will run each measurement multiple times to obtain the average performance. In the last step, the *Filter* will find out schemas with Pareto-optimal performance.

Evaluation Plan. We plan to evaluate our approach to check whether our approach can find better schemas in the tested performance metrics comparing to Ruby on Rails and Django. We

will evaluate it with 7 different object models. They have different sizes and inheritance and association relationships. The hardware platform will be a Spark cluster.

3.2 Performance Prediction for MapReduce

The problem of predicting system performance using configurations is a supervise learning task, where the system configuration is the input object, and the performance is a desired supervisory signal. The main approach includes data collection, data cleaning, and model training and verification.

Data Collection. The job tracker (or application master in MR2) in Hadoop framework is designed to track job execution and log the cluster and job configuration. It serves information about completed jobs. From the job tracker, we will gather performance and configuration data of successfully completed jobs. The gathered data is a list of <*configuration, performance*> pairs.

Data Preprocessing. Data preprocessing is important when building a prediction model [5, 7, 29]. It affects almost all aspects of the training phase, like convergence rate, prediction accuracy [16]. In addition the standard techniques, we also utilize the semantic meaning of parameters to select features. Specifically, we study the common relationships among parameters and leverage that knowledge to decompose interactions. For example, while a feature is turned off, the parameters used to fine tune it do not affect performance anymore.

Model Training and Verification. The models we plan to train are listed in Table 1. For each trained model, we will check the performance criteria listed in Table 2. We will check the accuracy of trained models with k-fold cross validation.

Table 1: Models to train

Prediction Models
Linear Regression
Ridge Regression
Support Vector Regression (with different kernels)
Random Forests

Table 2: Metrics to check

Performance Criteria
Explained Variance Score
Mean Absolute Error
Mean Squared Error
Median Absolute Error
R^2 Score

Evaluation Plan. We plan to evaluate our approach on three different clusters: a production cluster in an e-commerce company, a cluster built on AWS, and an in-house cluster. We will run various MapReduce jobs in HiBench [11] to collect training data.

3.3 Heuristic Configuration Space Exploration

The main idea is to shrink a configuration space using domain knowledge first and then to search near-optimal settings with heuristic searching algorithms.

Dimension Reduction with Domain Knowledge. In practice, some parameters define system behaviors that are not related to system performance. For example, the log level (INFO or WARN) and the default user name or access key do not impact system performance. Therefore, we can remove those parameters to reduce the space dimension. Our preliminary work shows that this approach could reduce more than half of the parameters in the Hadoop system.

Configuration Search with Heuristic Algorithms. Configuration parameters usually interact with each other in complex ways. Therefore, heuristic searching is an appropriate approach to search configuration spaces. Here we illustrate this approach with the genetic algorithm. We represent all parameters as a $n-bit$ vector, with each bit represents a configuration parameter[1]. A standard genetic algorithm can be used in following four steps:

- **Initialization:** Randomly generate N parameter vectors and benchmark their system performance.
- **Crossover and Mutation:** These two steps generate an offspring setting $offspring'_0$, as shown in Figure 2. We will benchmark the system with the new generated setting.
- **Fitness function:** We accept a new setting if its performance is better than its parents. Otherwise we discard it.
- **Termination:** This procedure terminates if a desired configuration is found. If not, it will terminate after a given number of iterations and return the best-so-far configuration.

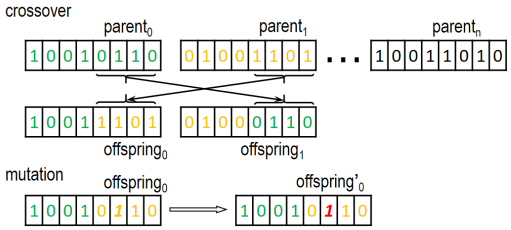

Figure 2: Genetic algorithm for configuration exploration

We also plan to implement other heuristic search algorithms like MCMC and compare their performance in the context of exploring configurations with better performance.

Evaluation Plan. We plan to take Hadoop as the study object. We will evaluate it on three different clusters similar to the performance prediction approach. We will use HiBench [11] as the benchmark tool. It covers common use cases of Hadoop systems in the industry, like web indexing, machine learning, and SQL-like jobs. There are two research questions we will answer. First, how much performance we can improve compare to the default settings? Since the previous study [19] shows that users just accept the default settings in terms of performance. Second, How much overhead/cost our approach will create? We will analyze the performance gain normalized by the resource to get such improvement, including the computation time, the monetary cost on AWS.

4 PRELIMINARY RESULTS

In this section, we show some preliminary results of the first and second approach.

4.1 ORM Design Space Exploration Result

Figure 3 illustrates the tradeoff between data storing and retrieval, where the triangles are Pareto-optimal solutions found by our approach, the diamonds are Rails solutions, and the stars are Django solutions. This figure shows that our approach reduces the data storing and retrieval time by almost 40%.

[1]To simplify the illustration, we assume all parameters are boolean variables. But it could have any data types in practice.

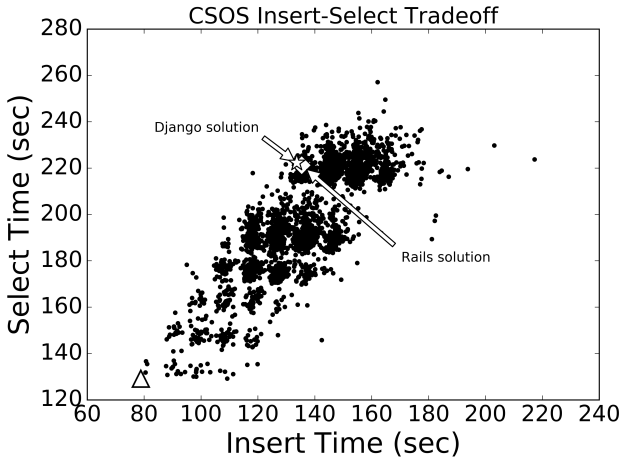

Figure 3: Tradeoff between time to store and retrieve data

4.2 Performance Prediction Result

I gathered a large data set from a production cluster at WalmartLabs. This dataset contains around 13K records. They span from simple data transfer jobs to large machine learning jobs. We trained four different models. Table 3 shows the performance of the trained random forests model.

Table 3: CPU time prediction result

Metrics	Value
R^2 Score	0.962
Cross Validation	0.947 (+/-0.034)
Mean Absolute Error	0.001
Mean Squared Error	8.663e-05
Median Absolute Error	6.904e-05

5 CONTRIBUTION

This research creates a comprehensive solution to explore design and configuration space for performance optimization. To this end, we provide a framework that combines design space synthesis and distributed computation to help design better systems from a partial specification. We also provide a novel approach to find better configurations using machine learning prediction models and heuristic search algorithms. Some preliminary results [2, 3, 23] have shown the effectiveness of our approach.

REFERENCES

[1] Erik Andersen, Sumit Gulwani, and Zoran Popovic. A Trace-based Framework for Analyzing and Synthesizing Educational Progressions. In *CHI '13*. NY, USA.

[2] Hamid Bagheri, Chong Tang, and Kevin Sullivan. 2014. Trademaker: Automated dynamic analysis of synthesized tradespaces. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 106–116.

[3] Hamid Bagheri, Chong Tang, and Kevin Sullivan. 2017. Automated synthesis and dynamic analysis of tradeoff spaces for object-relational mapping. *IEEE Transactions on Software Engineering* 43, 2 (2017).

[4] Barry W. Boehm. 1988. A spiral model of software development and enhancement. *Computer* 21, 5 (1988), 61–72.

[5] Sven F Crone, Stefan Lessmann, and Robert Stahlbock. 2006. The impact of preprocessing on data mining: An evaluation of classifier sensitivity in direct marketing. *European Journal of Operational Research* 173, 3 (2006), 781–800.

[6] Alan Davis, Scott Overmyer, Kathleen Jordan, Joseph Caruso, Fatma Dandashi, Anhtuan Dinh, Gary Kincaid, Glen Ledeboer, Patricia Reynolds, Pradip Sitaram, et al. 1993. Identifying and measuring quality in a software requirements specification. In *Software Metrics Symposium*. IEEE, 141–152.

[7] A Famili, Wei-Min Shen, Richard Weber, and Evangelos Simoudis. 1997. Data preprocessing and intelligent data analysis. *Intelligent data analysis* (1997).

[8] Pascal Felber, Christof Fetzer, and Torvald Riegel. 2008. Dynamic performance tuning of word-based software transactional memory. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*. ACM, 237–246.

[9] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau. 2003. SPARK: a high-level synthesis framework for applying parallelizing compiler transformations. In *16th International Conference on VLSI Design, 2003. Proceedings.* Institute of Electrical & Electronics Engineers (IEEE).

[10] Cheng Huang, Minghua Chen, and Jin Li. 2013. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. *ACM Transactions on Storage (TOS)* 9, 1 (2013), 3.

[11] 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, 41–51.

[12] Léonard Jaillet, Juan Cortés, and Thierry Siméon. 2010. Sampling-based path planning on configuration-space costmaps. *IEEE Transactions on Robotics* 26, 4 (2010), 635–646.

[13] D.-I. Kang, R. Gerber, L. Golubchik, J. K. Hollingsworth, and M. Saksena. 1999. A Software Synthesis Tool for Distributed Embedded System Design. *SIGPLAN Not.* 34, 7 (May 1999).

[14] Vu Le and Sumit Gulwani. FlashExtract: A Framework for Data Extraction by Examples. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '14)*. ACM, New York, NY, USA.

[15] Vu Le, Chengnian Sun, and Zhendong Su. 2015. Finding deep compiler bugs via guided stochastic program mutation. In *ACM SIGPLAN Notices*, Vol. 50. ACM.

[16] Oded Maimon and Lior Rokach. 2002. Improving supervised learning by feature decomposition. In *International Symposium on Foundations of Information and Knowledge Systems*. Springer, 178–196.

[17] Sandeep Neema, Janos Sztipanovits, Gabor Karsai, and Ken Butts. 2003. Constraint-based design-space exploration and model synthesis. In *International Workshop on Embedded Software*. Springer, 290–305.

[18] Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. 2005. Multi-objective design space exploration of embedded systems. *Journal of Embedded Computing* 1, 3 (2005), 305–316.

[19] 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.

[20] Adam Michael Ross. 2006. Managing unarticulated value: changeability in multiattribute tradespace exploration. *Engineering Systems Division* 361 (2006).

[21] Adam M Ross, Daniel E Hastings, Joyce M Warmkessel, and Nathan P Diller. 2004. Multi-attribute tradespace exploration as front end for effective space system design. *Journal of Spacecraft and Rockets* 41, 1 (2004), 20–28.

[22] 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.

[23] Chong Tang, Hamid Bagheri, Sarun Paisarnsrisomsuk, and Kevin Sullivan. 2017. Towards designing effective data persistence through tradeoff space analysis. In *Proceedings of the 39th International Conference on Software Engineering Companion*. 353–355.

[24] Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. 2009. Automatically finding patches using genetic programming. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society.

[25] James A Whittaker and Michael G Thomason. 1994. A Markov chain model for statistical software testing. *IEEE Transactions on Software engineering* (1994).

[26] Cemal Yilmaz, Myra B Cohen, and Adam A Porter. 2006. Covering arrays for efficient fault characterization in complex configuration spaces. *IEEE Transactions on Software Engineering* 32, 1 (2006), 20–34.

[27] Chuanjun Zhang, Frank Vahid, and Roman Lysecky. 2004. A self-tuning cache architecture for embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)* 3, 2 (2004), 407–425.

[28] Yi Zhang, Jianmei Guo, Eric Blais, and Krzysztof Czarnecki. Performance Prediction of Configurable Software Systems by Fourier Learning. In *ASE '15*. IEEE, IEEE Computer Society, Washington, DC, USA.

[29] Thomas Zimmermann and Peter Weißgerber. 2004. Preprocessing CVS data for fine-grained analysis. In *Proceedings of the First International Workshop on Mining Software Repositories*. sn, 2–6.