POSTER: In-situ Workflow Auto-tuning through Combining Component Models

Tong Shu
Southern Illinois University
Carbondale, IL, USA
tong.shu@siu.edu

Yanfei Guo
Argonne National Laboratory
Lemont, IL, USA
yguo@anl.gov

Justin Wozniak
Argonne National Laboratory
Lemont, IL, USA
woz@anl.gov

Xiaoning Ding
New Jersey Institute of Technology
Newark, NJ, USA
xiaoning.ding@njit.edu

Ian Foster
Argonne Natl. Lab and Univ. Chicago
Lemont and Chicago, IL, USA
foster@anl.gov

Tahsin Kurc
Stony Brook University
Stony Brook, NY, USA
tahsin.kurc@stonybrook.edu

Abstract

In-situ parallel workflows couple multiple component applications via streaming data transfer to avoid data exchange via shared file systems. Such workflows are challenging to configure for optimal performance due to the huge space of possible configurations. Here, we propose an in-situ workflow auto-tuning method, ALIC, which integrates machine learning techniques with knowledge of in-situ workflow structures to enable automated workflow configuration with a limited number of performance measurements. Experiments with real applications show that ALIC identify better configurations than existing methods given a computer time budget.

Keywords: in-situ workflow, auto-tuning, component model

1 Introduction

Emerging scientific workflows couple simulation tasks with analysis, visualization, learning, and other data processing tasks [5]. It is increasingly infeasible to use file systems to exchange intermediate results between tasks [9–13] due to the performance gap between the computational and I/O components of HPC systems. Thus, in-situ workflow solutions use network or shared memory to pass intermediate results [7].

However, in-situ workflows raise performance tuning challenges, making it difficult to fully exploit their performance advantages. A single component application running in isolation can be tuned by selecting good configurations with known auto-tuning methods [3, 4]. However, for in-situ workflows, it is insufficient to tune each component independently because components interact frequently and contend for resources during execution. Ideally, all parameters from all components should be optimized together, but this is rarely realistic with conventional methods due to the multiplicative increase in potential parameter combinations.

A key issue for automated optimization of in-situ workflows is thus how to produce high-performing configurations at an affordable cost. To this end, we propose an approach, namely Active Learning Initialized from Combining component models (ALIC), which combines the effectiveness of tuning a workflow as a whole and the simplicity of tuning individual components. It first trains models on each component separately and then leverages the trained component models to guide the search for optimal workflow configurations.

2 Algorithm: ALIC

Given a budget on the number m of workflow runs in auto-tuning, ALIC works as follows.

Initialization: We generate a set of p random samples from the workflow configuration space $C = C^{s(1)} \times C^{s(2)} \times \cdots \times C^{s(J)}$ as the workflow sample candidate pool C_{pool}, where $C^{s(j)}$ is the j^{th} component application's configuration space ($1 \le j \le J$).

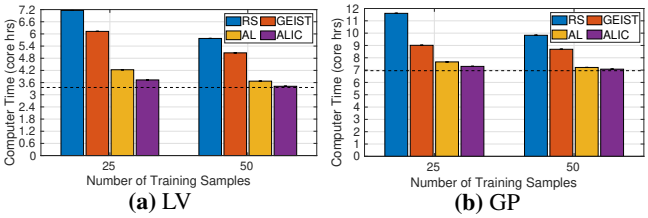
Component Sampling: If we previously have no enough training samples to build component surrogate models, we randomly select m_t samples from the configuration space of each component application to measure their performance, and incorporate the measured data into their existing measurements as the training data of respective component models.

Build Component Models: After training a third-party machine learning model, boosted tree [2], to build the surrogate model $M^{s(j)}$ of each component application j, we generate a component sample candidate pool $C^{s(j)}_{\text{pool}}$ of each application j from C_{pool} by extracting parameter dimensions only related to the j^{th} application and deleting duplicate configurations, and use the generated component model $M^{s(j)}$ to predict the performance of all configurations in $C^{s(j)}_{\text{pool}}$.

Estimate Performance: We use a low-fidelity approximator to estimate workflow performance for each configuration in C_{pool}. For example, workflow execution and computer times are approximated as $t^{\text{e}} = \max_j t^{\text{e}}_j$ and $t^{\text{c}} = \sum_j t^{\text{c}}_j$, where t^{e}_j and t^{c}_j are the execution and computer times of each stand-alone component application j, respectively.

PPoPP '21, February 27–March 3, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8294-6/21/02…$15.00
https://doi.org/10.1145/3437801.3441615



(a) LV **(b)** GP

Figure 1. Actual computer time at the best configuration auto-tuned (dashed lines: the best configuration in the test set)

Measure Performance: Initially, we measure the performance of m_0 samples randomly selected from C_{pool} as a workflow training data set D_{train}. Then, we sort the configurations in C_{pool} according to their predicted performance to obtain top $(m - m_0 - m_t)/I$ workflow configurations that have not been selected and measure their performance, where I is the preset maximum number of iterations. Also, we add the new measurements $D_{\text{train}}[i]$ $(1 \le i \le I)$ into D_{train}.

Refine the Workflow Model: We use the newly collected training data in D_{train} to incrementally train a third-party machine learning model, boosted tree [2], generating the corresponding workflow surrogate model M_i in the i^{th} iteration. Next, we use the generated workflow model M_i to predict the performance of all configurations in C_{pool}. Again, we predict the top $(m - m_0 - m_t)/I$ unselected configurations in C_{pool}, and proceed in this way until the I^{th} iteration.

3 Experimental Evaluation

We conducted experiments on the Argonne Bebop cluster with 36 CPU cores per node, and ran each workflow with exclusive access to node resources, on allocation sizes up to 32 nodes. We used two in-situ workflows coupled via the I/O library ADIOS [1]. **LV** couples two full-featured realistic applications: LAMMPS [8], a molecular dynamics simulator, and Voro++ [14], a Voronoi tesselator for analysis/visualization. **GP** for multi-purpose analysis in chemical reaction dynamics couples four applications: Gray-Scott reaction-diffusion simulation; an analysis application, PDF calculator, applied to the Gray-Scott output; a visualization application, G-Plot, also applied to the Gray-Scott output; and a second visualization application, P-Plot, applied to the PDF output. Here, our goal is to optimize the computer time of each workflow, so computer time achieved by the best configuration predicted is used as the metric to evaluate auto-tuning algorithms.

Application configuration options form a total of 2.3×10^{10} possible configurations for LV and 8.5×10^7 for GP. For each workflow, we generated, as C_{pool} and a test set, 2000 configurations of randomly selected parameter values. For each configuration in the test set, we launched all workflow components at once and recorded each component's end-to-end wall-clock time. We used the longest component execution time as the configuration's *execution time*, and the product of execution time, number of computing nodes used, and number of cores per node as the configuration's *computer time*. We also measured the computer time of 500 configurations randomly selected from the parameter space of each configurable component, and used these samples as component measurements, from which ALIC may select training samples for component models.

We compare ALIC with three existing auto-tuning algorithms: **RS** selects training data by random sampling. **AL** is an active learning algorithm that iteratively selects a batch of the best configurations predicted by gradually refined models as training samples [3]. **GEIST** is a state-of-the-art AL-based auto-tuning algorithm guided by a parameter graph [4]. We incorporated these four auto-tuning algorithms into a HPC auto-tuner system based on a robust framework in [6], and run each algorithm 100 times to report averages.

We measured the actual computer time of the best configurations of LV and GP predicted by RS, GEIST, AL, and ALIC, and plot the measurements for m of 25 and 50 in Fig. 1, which shows that the computer times achieved by ALIC are always better than by RS, GEIST, and AL. ALIC outperforms AL, because surrogate models trained with the same number of training samples are much more accurate for component applications than in-situ workflows, and our method of estimating workflow performance provides relatively accurate configuration ranking over top configurations.

Acknowledgments

This research was supported by the Exascale Computing Project (17-SC-20-SC) from U.S. Department of Energy and start-up funds from Southern Illinois University Carbondale.

References

[1] ADIOS. 2021. https://csmd.ornl.gov/adios.
[2] T. Chen and C. Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *ACM SIGKDD*. 785–794.
[3] B. Behzad et al. 2019. Optimizing I/O Performance of HPC Applications with Autotuning. *ACM TOPC* 5, 4 (2019), 15:1–15:27.
[4] J. Thiagarajan et al. 2018. Bootstrapping Parameter Space Exploration for Fast Tuning. In *ACM ICS*. 385–395.
[5] J. Wozniak et al. 2018. Scaling Deep Learning for Cancer with Advanced Workflow Storage Integration. In *MLHPC in conjunction with ACM/IEEE SC*. Dallas, TX, USA, 114–123.
[6] J. Wozniak et al. 2019. MPI Jobs within MPI Jobs: a Practical Way of Enabling Task-level Fault-tolerance in HPC Workflows. *Elsevier FGCS* 101 (2019), 576–589.
[7] Y. Fu, F. Li, F. Song, and Z. Chen. 2018. Performance Analysis and Optimization of In-situ Integration of Simulation with Data Analysis: Zipping Applications up. In *ACM HPDC*. 192–205.
[8] LAMMPS. 2021. https://lammps.sandia.gov.
[9] T. Shu. 2017. *Performance Optimization and Energy Efficiency of Big-data Computing Workflows*. Dissertation. NJIT, Newark, NJ, USA.
[10] T. Shu and C. Wu. 2016. Energy-efficient Mapping of Big Data Workflows under Deadline Constraints. In *WORKS in conjunction with ACM/IEEE SC*. Salt Lake City, UT, USA, 34–43.
[11] T. Shu and C. Wu. 2017. Energy-efficient Dynamic Scheduling of Deadline-constrained MapReduce Workflows. In *IEEE e-Science*. Auckland, New Zealand, 393–402.
[12] T. Shu and C. Wu. 2017. Performance Optimization of Hadoop Workflows in Public Clouds through Adaptive Task Partitioning. In *IEEE INFOCOM*. Atlanta, GA, USA, 2349–2357.
[13] T. Shu and C. Wu. 2020. Energy-efficient Mapping of Large-scale Workflows under Deadline Constraints in Big Data Computing Systems. *Elsevier FGCS* 110 (2020), 515–530.
[14] Voro++. 2021. http://math.lbl.gov/voro++.