Publication status: Not informed by the submitting author

Machine learning approach to multicore data structures

Hrishitva Patel

https://doi.org/10.1590/SciELOPreprints.5006

Submitted on: 2022-11-03
Posted on: 2022-11-30 (version 1)
(YYYY-MM-DD)

Machine learning approach to multicore data structures

Name of the author

Hrishitva Patel

Name of the University

SUNY (State University of NY)

Binghamton

Email:

hpatel51@binghamton.edu

ORC ID: 0000-0001-7887-6641

Abstract

In this study, a novel method for constructing self-aware data structures using online machine learning is proposed. This research introduced a novel category of data structures called Smart Data Structures, which continuously and automatically improve themselves to help simplify the complexity of manually modifying data structures for varied systems, applications, and workloads. This study also concluded that online machine learning is useful for autonomous data structure modification. For the online machine learning algorithm, I have proposed a reinforcement machine learning algorithm that benefits from the reward system and optimize the knobs accordingly. Online learning, in my opinion, offers a trustworthy and efficient framework for assessing intricate dynamic tradeoffs. Many of the possible difficulties that programmers may encounter in their daily work may be eliminated by using intelligent multicore data structures.

Keywords: Machine learning, automation, computer science, Data structures

Introduction:

Programming complexity is rising as multicores become common. Programmers put a lot of work into parallelizing issues and putting them on hardware in a way that maintains all threads active and collaborating well. The most daunting design challenge in many applications is effectively managing thread cooperation via shared data structures. The selection of parallel data structure methods and algorithm parameter choices, unfortunately, is becoming more and more sensitive to application performance. The design of the machine's memory system, as well as application-specific factors like the load on the data structure, can have complex effects on the optimal method and parameter choices. Even worse, a lot of programs employ input-dependent processing, which makes the load change dynamically.

Self-aware computing has recently been put out as one automated method of relieving the programmer of this burden. Self-aware systems aim to automatically monitor themselves and dynamically adjust their behavior at runtime, in contrast to typical systems that need the programmer to manually balance system restrictions. Closed-loop optimization is a technique used by self-aware systems to obtain the highest performance possible at any given moment. These systems measure performance feedback and continuously modify as system circumstances change. They have been used on a variety of platforms, including embedded and real-time, desktop, server, and cloud computing settings. They are frequently referred to as autonomic, auto-tuning, adaptive, etc. Hence it is really important as well as one of the most difficult aspects to design in such a way that all the Threads are arranged and collaborated through a shared data structure so that they can work efficiently. It is worthwhile to mention that the most effective algorithm, as well as parameter settings, mostly depends on the memory system architecture of the machine backed by the criteria of being specific to the application (Conradihoffmann and Frohlich 2021). It is a humongous task to code the complexities manually for the programmers so many applications have been set up to be input-dependent

which helps to distribute the load dynamically to derive good performance from a variety of inputs and machines.

In this work, a new category of self-aware parallel data structures called "Smart Data Structures" is introduced. These structures automatically fine-tune themselves using cutting-edge techniques based on online machine learning. Smart Data Structures relieve programmers of the stress of manually adjusting for the optimum performance across various computers, applications, and inputs through learning and automated tuning.

Functioning of the Data Structure:

Standard data structures can be replaced with smart data structures, which self-optimize. In order to deploy them, a common data structure is layered with an online learning engine. To give an example, the basic architecture of a common data structure to that of an intelligent data structure are compared in the subsequent part.

The elements of a common data structure are depicted in Figure 1. Algorithms, an interface, and data storage make up standard data structures. The data is organized by the storage, the actions that threads may do on the data to alter or query it are specified by the interface, and the interfaces are implemented by the algorithms while maintaining accurate concurrent semantics. Thresholds and other parameters that are used to regulate storage and algorithms are termed as knobs (Rosa et al.,2019). Knobs are often set up using universal static defaults that were created by the library programmer. The knobs must be manually adjusted by programmers when the default settings don't work well. The trial and error method is frequently used for this, which can lengthen the development process and cause-specific situations in the code to become less readable (Hoffmann 2020). Due to its complexity, programmers generally overlook runtime tweaking even though it is immensely important.

Smart Data Structures are depicted in Figure 2. The same storage, interfaces, and methods are maintained by smart data structures. The distinction is that Smart Data Structures add an online learning engine to conventional data structures, which automatically and dynamically adjusts the knobs to maximize storage and algorithmic behavior. Smart Data Structures respond to modifications to the system or inputs that influence these tradeoffs by balancing complicated tradeoffs to discover the best knob settings (Pagani et al., 2018).

Smart Data Structures are very different from these earlier studies. Prior research has the ability to adapt to various machine architectures and runtime situations, such as input size, but these decisions are often based on thresholds derived at compile-time or install-time characterization. The issue is that in contemporary systems, such characterizations may not accurately represent true runtime settings. Smart Data Structures use an online approach to improve decisions to account for these complexities. They gather real-time data on system dynamics and performance trade-offs. Through online learning, they intelligently balance such tradeoffs at runtime. To obtain the highest performance possible, Smart Data Structures learn to respond to shifts in the system, program, or inputs.

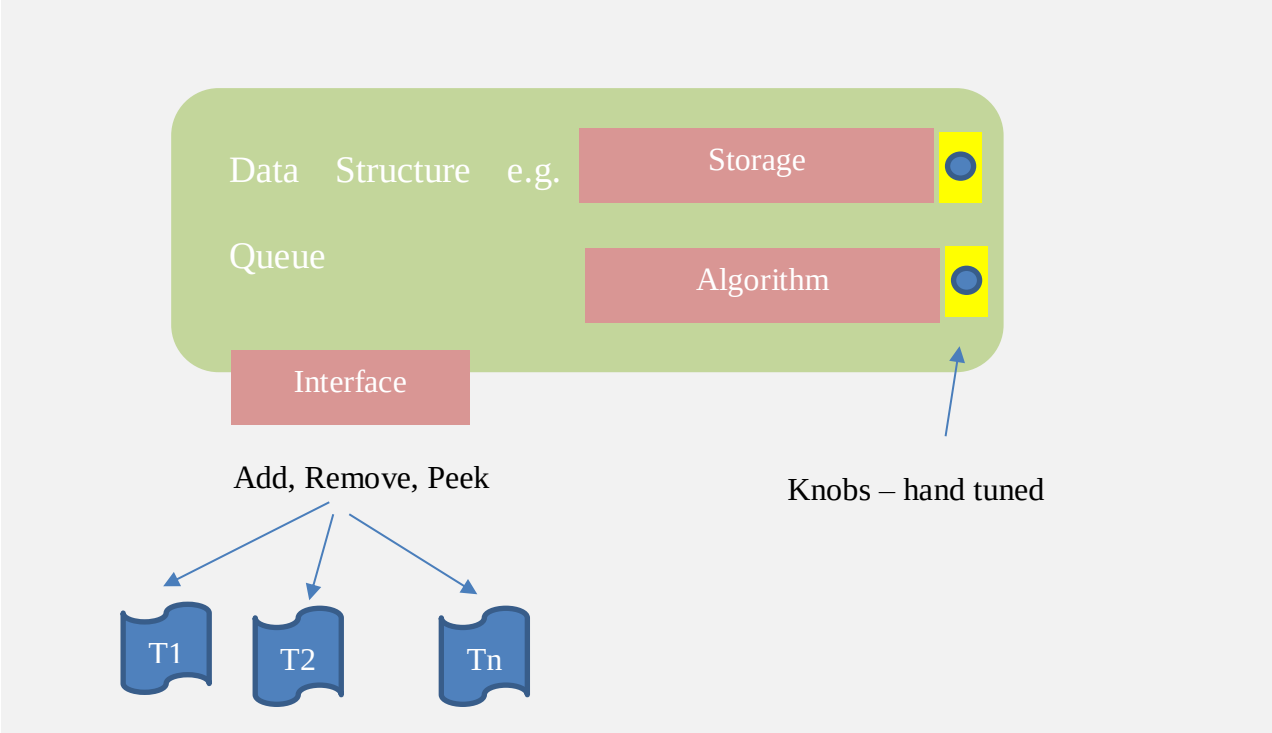

Fig 1: Parallel Data Structure

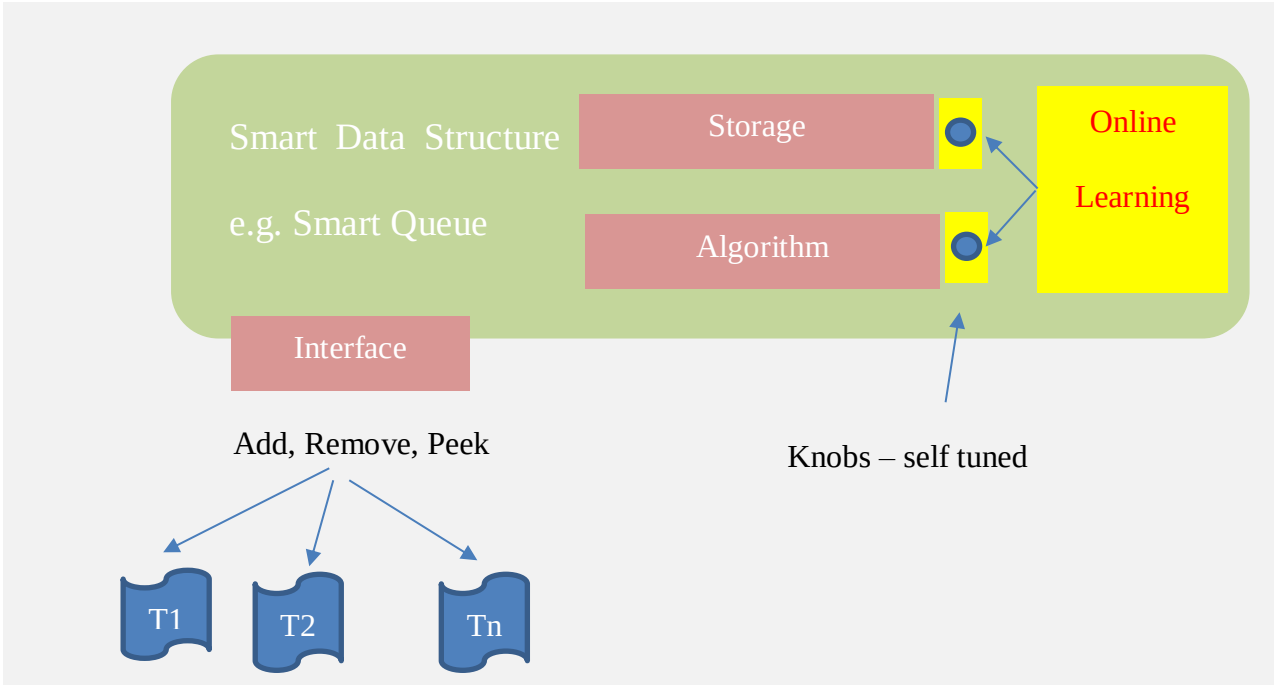

Fig 2: Smart Multi-core Data Structure

Internally, a Smart Data Structure uses online machine learning to learn knob settings that best optimize its storage and algorithms. Each Smart Data Structure attaches to an online learning engine that optimizes its knobs. That learning engine runs in a learning thread separate from the application threads. The number of learning threads is parameterizable. Optimization within each learning engine is driven by performance feedback, i.e. the reward. The reward signal must reflect application performance accurately without perturbing it (Iranfar et al., 2019). The prototype of smart data structures has been outlined in this study that can fine-tune itself based on online machine learning.

Function of Online Machine Learning Algorithm

As discussed in the previous section, an online machine learning algorithm is used to monitor and control the knob settings for the Smart Data Structures, let's have a deep dive into how the Online Machine Learning algorithm should work.

The Reinforcement Learning (RL) method used by our suggested Smart Data Structures reads a reward signal and aims to maximize it. I choose an average reward optimality criterion and utilize policy gradients to train an effective policy because we require an algorithm that is both a) quick enough for online use and b) tolerant of severe partial observability. Specifically, the Natural Actor-Critic method was employed. A policy is a conditional distribution over "actions," and the purpose of policy gradients is to enhance a policy given by a state. The agent conducts a sampled action from this policy at each timestep. Actions are a vector of discrete-valued scan counts, one for each Smart Data Structure, for the Smart Queue, Pairing Heap, and Skip List. Installing each scan count in its appropriate Smart Data Structure constitutes performing the action.

I calculate the average reward received by executing a given policy to determine the quality of that policy. The average reward attained by carrying out the policy's instructions is a function of its parameters. I specified the typical reward as:

$$\eta(\theta) = \mathrm{E}\{R\} = \lim_{i-0} 1/i \sum_{t=1}^{i} rt$$

where rt is a specific reward at time t, collected from the aggregate of throughputs from all Smart Data Structures or from an external monitor and smoothed across a brief window of time, and R is a random variable signifying reward. Because various settings result in a distinct distribution of actions, and because different actions alter how the system state evolves over time, the average reward is a function of the parameters.

Our solution adds one thread to the program to operate the learning engine. The benefit is that it lessens interruptions to the program and permits background optimization while it is running. The employment of a second thread entails a trade-off because the program may have used the second thread for parallelism. Only if there is a net performance improvement is the additional thread warranted. The learning thread is a separate, additional dedicated thread that operates independently of the application's threads. Creating knob optimizations as rapidly as feasible and responding to sudden changes in the system or application that alter the ideal knob settings over time are the driving forces behind this approach.

Fortunately, net gains are easy to achieve in common scenarios. While various auto-tuning libraries have experimented with offline machine learning to statically tune the library, to our knowledge, Smart Data Structures is the first work to apply online machine learning successfully. Online machine learning is an effective optimization strategy for adaptive data structures. Further, I contribute a learning algorithm based on Policy Gradients Reinforcement

Learning that is simultaneously high-performance and efficient enough for online use in adaptive data structures.

Conclusion:

This research presented a unique approach to designing self-aware data structures through online machine learning. Based on this technique, I presented a new class of data structures called Smart Data Structures that continually and automatically optimize themselves to assist reduce the complexity of manually adjusting data structures for various systems, applications, and workloads. The approach suggests that using online learning to improve other systems is also a good strategy. For the online machine learning algorithm, I have proposed a reinforcement machine learning algorithm that benefits from the reward system and optimize the knobs accordingly. In my opinion, online learning provides a reliable and effective framework for evaluating complex dynamic tradeoffs, and it will be crucial in the creation of future systems. Smart multicore data structures can eradicate many of the potential challenges that programmers face in their routine tasks.

References:

Cassales, G., Gomes, H., Bifet, A., Pfahringer, B. and Senger, H., 2020, December. Improving parallel performance of ensemble learners for streaming data through data locality with mini-batching. In *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)* (pp. 138-146). IEEE.

Conradihoffmann, J.L. and Frohlich, A.A., 2021. Online machine learning for energy-aware multicore real-time embedded systems. *IEEE Transactions on Computers*.

da Rosa, F.R., Garibotti, R., Ost, L. and Reis, R., 2019. Using machine learning techniques to evaluate multicore soft error reliability. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *66*(6), pp.2151-2164.

Hoffmann, J.L.C., 2020. Optimizing energy consumption of multicore real-time embedded systems using machine learning.

Iranfar, A., De Souza, W.S., Zapater, M., Olcoz, K., de Souza, S.X. and Atienza, D., 2019, October. A machine learning-based framework for throughput estimation of time-varying applications in multi-core servers. In *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)* (pp. 211-216). IEEE.

Pagani, S., Manoj, P.S., Jantsch, A. and Henkel, J., 2018. Machine learning for power, energy, and thermal management on multicore processors: A survey. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *39*(1), pp.101-116.

ul Islam, F.M.M., Lin, M., Yang, L.T. and Choo, K.K.R., 2018. Task aware hybrid DVFS for multi-core real-time systems using machine learning. *Information Sciences*, *433*, pp.315-332.

Zhou, J., Yang, W., Dai, M., Cai, Q., Wang, H. and Li, K., 2021, November. Parallel Sparse LU Factorization With Machine-Learning Method on Multi-core Processors. In *2021 7th International Conference on Systems and Informatics (ICSAI)* (pp. 1-11). IEEE.

Conflict of Interest:

There have been no conflict of interests with any individual or organization to declare.

This preprint was submitted under the following conditions:

- The authors declare that they are aware that they are solely responsible for the content of the preprint and that the deposit in SciELO Preprints does not mean any commitment on the part of SciELO, except its preservation and dissemination.
- The authors declare that the necessary Terms of Free and Informed Consent of participants or patients in the research were obtained and are described in the manuscript, when applicable.
- The authors declare that the preparation of the manuscript followed the ethical norms of scientific communication.
- The authors declare that the data, applications, and other content underlying the manuscript are referenced.
- The deposited manuscript is in PDF format.
- The authors declare that the research that originated the manuscript followed good ethical practices and that the necessary approvals from research ethics committees, when applicable, are described in the manuscript.
- The authors declare that once a manuscript is posted on the SciELO Preprints server, it can only be taken down on request to the SciELO Preprints server Editorial Secretariat, who will post a retraction notice in its place.
- The authors agree that the approved manuscript will be made available under a Creative Commons CC-BY license.
- The submitting author declares that the contributions of all authors and conflict of interest statement are included explicitly and in specific sections of the manuscript.
- The authors declare that the manuscript was not deposited and/or previously made available on another preprint server or published by a journal.
- If the manuscript is being reviewed or being prepared for publishing but not yet published by a journal, the authors declare that they have received authorization from the journal to make this deposit.
- The submitting author declares that all authors of the manuscript agree with the submission to SciELO Preprints.