Talpidae: Tuning Library Calls in the Dark

Tomer Kaftan

Computer Science and Engineering, University of Washington
Seattle, Washington, USA

tomerk11@cs.washington.edu

1. PROBLEM AND MOTIVATION

Modern data analytics workloads process large datasets on shared-nothing clusters, and heavily rely on user-defined functions (UDFs) to express complex operations. These UDFs can bottleneck workloads even when executing in a perfectly parallel fashion. For example, when analyzing Magnetic Resonance Imaging (MRI) data from the Human Connectome Project [14] using Apache Spark's [1] Python API, Pyspark, as much as 85% of the 1.7 hour execution time is spent in a perfectly parallel image denoising step.

These UDFs in turn spend much of their execution time inside methods from external libraries. These libraries, such as NumPy [2] and SciPy [3] in Python, have already been heavily optimized and execute critical methods in C for performance. They additionally provide tuning parameters and method alternatives to allow users to optimize performance for their specific workload. For example, a linear algebra method may allow tuning the numerical precision used or the in-memory layout of a matrix. In many cases the tuning decisions that minimize runtime depend on the nature of the data and the system hardware, but certain tuning decisions may risk introducing numerical instability and other approximation errors. The denoising operation mentioned above, which comes from the computational neuroanatomy library Dipy [10], is a form of blockwise nonlocal means denoising [6] and allows tuning the block size. Shrinking the block size can improve performance by orders of magnitude, but introduces some approximation error.

Tuning such library calls is key to improving the performance of UDF-bottlenecked workloads. Unfortunately, even if users of a library method are aware it can be tuned, there is no clear tuning rule of thumb for speeding up execution. It can be even more challenging for users to reason about how much approximation error tuning decisions introduce.

To address this problem we introduce *Talpidae*: an extension to Apache Spark that statically analyzes Python UDFs, detects tunable library calls, and adaptively alters them for optimal performance while respecting any user-specified tolerances on approximation error. It does so by treating the

tuning process as a *multi-armed bandit problem* [15]. It uses an ϵ -decreasing strategy when there are no constraints on approximation error. In the case where there are constraints on approximation error, *Talpidae* explores all tuning alternatives for the first few data elements to decide which of the tuning options are safe to use, and then selects the one with the optimal speed-accuracy balance as specified by the user.

We use microbenchmarks to demonstrate *Talpidae*'s ability to adaptively tune UDFs in workloads with varying properties when approximation error is not a concern. We also explore the challenges introduced when tuning parameters risk introducing approximation error. Despite these challenges, we are able to demonstrate a 2.4x speedup in the MRI workload while in expectation remaining within specified approximation error tolerances.

2. BACKGROUND AND RELATED WORK

White Box UDF treatment: There have been a variety of data processing systems that utilize UDF properties rather than treating them as black boxes. *Tupleware* [7] uses LLVM UDFs and collects low-level statistics about the UDFs such as compute cycles and load cycles, which it uses to generate alternate high-level code in the query optimizer. It then compiles the workflows, gaining low-level compiler optimizations within the UDFs. *Stratosphere* [11] statically analyzes UDF bytecode in a map-reduce setting, and identifies properties that suggest the UDFs may safely be reordered in certain ways. The query optimizer then alters the query plan for better performance by reordering the UDFs when safe. *KeystoneML* [13] collects statistics about the inputs to each workload stage in advance, then uses developer-provided cost models to select which developer-provided UDF implementation to execute. When implementations approximate the results, developers must design appropriate cost models. *Talpidae* also treats UDFs as a white box, but uses multi-armed bandits during workload execution to tune UDFs.

Compile-time Autotuning: A variety of frameworks use offline empirical autotuning to create efficient and portable domain-specific libraries. As an example, *ATLAS* [16] produces optimized matrix multiply libraries, and *FFTW* [9] produces FFT solvers. *PetaBricks* [4] is a language and compiler that allows programmers to specify algorithmic alternatives. At compile time it synthesizes poly-algorithms that map inputs to the different algorithmic alternatives, potentially recursively, using input size. *OpenTuner* [5] is an open-source framework to build multi-objective autotuners. It allows tuning programs for a trade-off between speed,

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SIGMOD '17 Student Research Competition, May 14–19, 2017, Chicago, IL, USA.

© 2017 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4199-8/17/05..

DOI: <http://dx.doi.org/10.1145/3055167.3055183>

memory usage, accuracy, and any other user-specified objective. Talpidae uses an online approach and can adapt to workloads without requiring offline tuning on data with similar properties.

Run-time Autotuning: Other approaches have focused on dynamically tuning systems for a single objective at run-time. Similarly to Talpidae, *Micro Adaptivity* [12] in *Vectorwise* works by adaptively selecting the fastest of several ‘flavors’ of built-in primitives by treating the decision as a multi-armed bandit problem to balance exploration and exploitation. Another example is *Active Harmony* [8], an infrastructure that allows libraries and applications built on top of Active Harmony to expose tunable parameters. At runtime it searches the parameter space using an approach based on simplex optimization. Talpidae explores online tuning in a distributed data processing setting, and considers approximation error as well as raw speed.

3. APPROACH AND UNIQUENESS

Talpidae uses static code analysis to identify what parts of UDFs are tunable, then uses that information to dynamically optimize UDFs in a distributed big data processing system at execution time, using multi-objective tuning that balances speed and accuracy.

3.1 Extending Apache Spark with Talpidae

Apache Spark [1] is a distributed big data processing system similar to mapreduce. A Spark cluster consists of a master and multiple workers, which all coordinate with each other to process datasets. Spark’s `map` operator takes a UDF with one input and one output, and applies it to every single item in a dataset to produce a new dataset. Spark executes `map` in a perfectly parallel fashion, with each worker process going through a different subset of the data. To build Talpidae, we modified `map` to additionally take a string of library annotations that specifies which methods have tunable parameters, and optionally for a user to provide how they want approximation error to be treated in the tuning process. We then insert a static code analysis step before the workload is run. This step checks to see if the UDF uses any tunable library calls, and generates an equivalent UDF that uses a multi-armed bandit solver [15] to tune the code while `map` is executing.

3.2 Tuning Strategy

Talpidae tunes the library calls within UDFs by treating each worker’s tuning as a separate multi-armed bandit problem. We model each option for a library call with an initially unknown cost distribution. Observations from this distribution are sampled i.i.d. at random whenever we take an *action* of executing an option. Our goal is to process the `map` task while minimizing the total cost of all actions. The bandit problem is a classic setting from probability theory with many well-studied online learning solutions. When users specify that they are unconcerned about approximation error, Talpidae uses an ϵ -decreasing strategy [15] by default. With probability ϵ , which decays over time, it explores the tuning space by uniformly at random selecting an action, and otherwise it exploits the best-performing action. We also provide configurable ϵ -first and ϵ -greedy strategies. Because Talpidae tunes workers separately, it does not require any coordination across Spark workers. This simplifies the implementation of Talpidae and allows Talpidae to tune

Figure 1: Timings for the MRI analysis workload using different approximation thresholds.

more effectively when data properties differ across nodes. The drawback is that if data properties are similar across workers, tuning may take longer than when nodes communicate observed samples to each other.

3.3 Approximation Error Tolerance

When approximation errors are a concern, users label an action as **low-error** and provide a method to compute approximation error by comparing to the output of the **low-error** action. During the exploration phase, Talpidae will use an ϵ -first strategy that tests everything but only returns results from the **low-error** action. During the exploitation phase, Talpidae considers both expected approximation error and speed to select an action.

4. RESULTS AND CONTRIBUTIONS

We have implemented an extension to Apache Spark that is able to statically analyze UDF code, and uses multi-armed bandits to adaptively tunes the code for multiple objectives at runtime, balancing both speed and approximation error according to user-specified settings. It requires only library annotations of tunable parameters and method alternatives.

We test Talpidae on a real MRI image volume processing workload in Apache Spark over 33.6 GB of data on a cluster of four r3.2xlarge workers and one master from Amazon Web Services Cloud. Each r3.2xlarge instance has 8 virtual CPUs, 61 GB of Memory and 160 GB SSD storage. The results are shown in Figure 1. Talpidae selects the fastest tuning parameter that meets the approximation error threshold, with a constant overhead spent exploring the expected errors of each tuning parameter.

We have built microbenchmarks that demonstrate Talpidae is adaptive to data and query properties. When tuning matrix multiplication with sparser matrices on some workers than others, the independent tuning on each worker allows Talpidae to process the data 2x faster than possible when all workers are required to learn the same action.

5. ACKNOWLEDGEMENTS

We would like to thank Parmita Mehta and Ariel Rokem for providing an MRI data processing workload implemented on top of Apache Spark. This work is supported in part by the National Science Foundation under grants IIS-1247469, IIS-1546083, & CNS-1563788; DARPA award FA8750-16-2-0032; DOE award DE-SC0016260; the Intel Science & Technology Center for Big Data, and a Weil Family Endowed Fellowship in Computer Science & Engineering.

6. REFERENCES

- [1] Apache Spark. <http://spark.apache.org/>.
- [2] NumPy. <http://www.numpy.org/>.
- [3] SciPy. <http://www.scipy.org/>.
- [4] S. P. Amarasinghe. Petabricks: a language and compiler based on autotuning. In *High Performance Embedded Architectures and Compilers, 6th International Conference, HiPEAC 2011, Heraklion, Crete, Greece, January 24-26, 2011. Proceedings*, page 3, 2011.
- [5] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U. O'Reilly, and S. P. Amarasinghe. Opentuner: an extensible framework for program autotuning. In *International Conference on Parallel Architectures and Compilation, PACT '14, Edmonton, AB, Canada, August 24-27, 2014*, pages 303–316, 2014.
- [6] P. Coupe, P. Yger, S. Prima, P. Hellier, C. Kervrann, and C. Barillot. An optimized blockwise nonlocal means denoising filter for 3-D magnetic resonance images. *IEEE Trans. Med. Imaging*, 27(4):425–441, Apr. 2008.
- [7] A. Crotty, A. Galakatos, K. Dursun, T. Kraska, C. Binnig, U. Çetintemel, and S. Zdonik. An architecture for compiling udf-centric workflows. *PVLDB*, 8(12):1466–1477, 2015.
- [8] C. Țăpuș, I.-H. Chung, and J. K. Hollingsworth. Active harmony: Towards automated performance tuning. In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, SC '02*, pages 1–11, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.
- [9] M. Frigo and S. G. Johnson. FFTW: an adaptive software architecture for the FFT. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98, Seattle, Washington, USA, May 12-15, 1998*, pages 1381–1384, 1998.
- [10] E. Garyfallidis, M. Brett, B. Amirbekian, A. Rokem, S. van der Walt, M. Descoteaux, I. Nimmo-Smith, and Dipy Contributors. Dipy, a library for the analysis of diffusion MRI data. *Front. Neuroinform.*, 8:8, 21 Feb. 2014.
- [11] F. Hueske, M. Peters, M. J. Sax, A. Rheinländer, R. Bergmann, A. Krettek, and K. Tzoumas. Opening the black boxes in data flow optimization. *Proc. VLDB Endow.*, 5(11):1256–1267, July 2012.
- [12] B. Raducanu, P. A. Boncz, and M. Zukowski. Micro adaptivity in vectorwise. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 1231–1242, 2013.
- [13] E. R. Sparks, S. Venkataraman, T. Kaftan, M. J. Franklin, and B. Recht. KeystoneML: Optimizing Pipelines for Large-Scale Advanced Analytics. *ArXiv e-prints*, Oct. 2016.
- [14] D. C. Van Essen, S. M. Smith, D. M. Barch, T. E. J. Behrens, E. Yacoub, K. Ugurbil, and WU-Minn HCP Consortium. The WU-Minn human connectome project: an overview. *Neuroimage*, 80:62–79, 15 Oct. 2013.
- [15] J. Vermorel and M. Mohri. *Multi-armed Bandit Algorithms and Empirical Evaluation*, pages 437–448. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.
- [16] R. C. Whaley and J. J. Dongarra. Automatically tuned linear algebra software. In *Proceedings of the ACM/IEEE Conference on Supercomputing, SC 1998, November 7-13, 1998, Orlando, FL, USA*, page 38, 1998.