Identifying Optimal Parameters for Approximate Randomized Algorithms

Vimuth Fernando, Keyur Joshi, Darko Marinov, Sasa Misailovic
University of Illinois at Urbana-Champaign
{wvf2,kpjoshi2,marinov,misailo}@illinois.edu

Abstract

Many modern applications require low-latency processing of large data sets, often by using approximate data-analytics algorithms that trade accuracy of the results for faster execution or less memory consumption. Although the algorithms have theoretical accuracy and performance guarantees, the guarantees are often too conservative. In contrast, empirical models may give tighter error bounds but often come without guarantees of any sort. In this paper, we study the differences between the *analytical* and *empirical* models of accuracy, performance, and resource consumption, and opportunities for combining the two approaches. Based on our initial study, we discuss several interesting directions to leverage both analytical and empirical models.

1 Introduction

Many modern applications require low-latency processing of massive data sets. To meet such demands, researchers have developed various approximate algorithms, data structures, and systems software that trade accuracy for performance and/or memory consumption. These applications often come with analytically derived accuracy and performance specifications, which are typically probabilistic. Probabilistic specifications have been proposed for applications in areas as diverse as theoretical computer science [7–9, 12, 18, 21], numerical computing [10, 11, 14, 20], and databases [3, 4, 13, 19, 23].

Probabilistic specifications often come in the form of *probability predicates*, which specify the expected probability that the output satisfies a particular condition, or *expectation predicates*, which specify the expected value of the output. The specifications usually contain *algorithm parameters* that control the accuracy but also directly impact performance, energy, and/or memory consumption. The user can set the *configuration* of such an algorithm (i.e., the values for the parameters) to obtain the results with guaranteed accuracy within guaranteed execution time.

We recently demonstrated how to leverage analytical specifications to check that an algorithm has been correctly implemented: AxProf [16] is our algorithmic profiling framework for randomized approximate programs. AxProf helps developers test if their implementations satisfy an algorithm's specifications provided in a formal notation. AxProf produces the code that (1) generates inputs according to a distribution that may reveal problems, and (2) selects an appropriate statistical test and how many trials to perform.

However, analytical probabilistic specifications are often too conservative as they have to take into account worst-case scenarios or perform average case analysis for a large input domain. We present one such example in Section 2. In addition, algorithm implementers can make design decisions that make the implementation behave differently from the algorithm specification. Developers commonly implement poly-algorithms which, based on the input parameters, may select and run one of many basic algorithms. Finally, for some applications it is impossible to derive analytical error models based on algorithm parameters due to complex interactions among parameters (e.g., [15]). These factors make it difficult for users to identify values for parameters to satisfy their accuracy and performance requirements.

Alternatively, empirical models based on dynamic executions of programs may yield algorithm configurations that satisfy accuracy requirements but provide better time or memory usage. Autotuning frameworks are commonly used for this purpose: they explore the space of program configurations, looking for those that satisfy various accuracy bounds on concrete inputs. However, empirical models cannot provide guarantees, and can substantially depend on the inputs.

We identify the opportunity to design new hybrid accuracy/performance/consumption models that enjoy majority of the benefits of both analytical and empirical models. In this paper, we present an initial study that shows the potential for the applications in this domain to tolerate noise and compare error bounds derived from analytical and empirical studies. We then present several challenges and promising research directions to develop more principled empirical models that would respect and adapt to analytical bounds.

2 Example: CountMin Sketch

The CountMin sketch algorithm [8] estimates the frequency of items in a dataset. The algorithm maintains a set of uniform hash functions whose range is divided into a set of bins. For every item in the dataset, the hash functions are calculated and a counter in each mapped bin is incremented. The frequency of an item is calculated as the minimum value of all the counters in corresponding bins.

The algorithm comes with a probabilistic specification that defines the errors to be expected, in this case the difference between the estimated count from the CountMin sketch and the actual count for each unique element:

$$P[Error < n * \epsilon] > 1 - \delta$$

for some user defined ϵ and δ (n is the size of the input).

To achieve this accuracy specification, the algorithm needs to set two parameters, w (the width of each hash table), and d (the number of hash tables to use) to the following values:

$$w = \lceil e/\epsilon \rceil, d = \lceil ln(1/\delta) \rceil$$

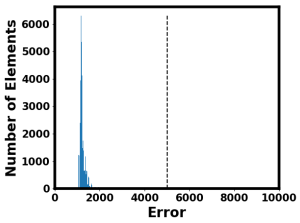

Figure 1. Errors: CountMin Sketch (default)

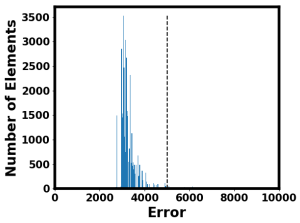

Figure 2. Errors: CountMin Sketch (tuned)

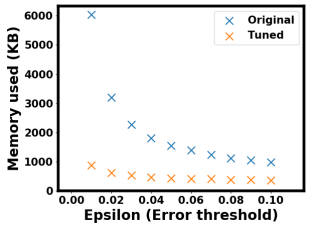

Figure 3. Memory usage of CountMin Sketch

While sound, these parameter assignments are often too conservative, especially for data sets with fewer unique items.

Figure 1 shows a histogram of errors we observed by running an implementation of the algorithm called *awn* [2] for a randomly generated dataset of 100,000 items. The figure shows on the x-axis the error and on the y-axis the number of elements with that error. A user setting ϵ to 0.05 and δ to 0.1 would expect the error in the count to be at most 5,000 for most elements (the vertical line) based on the analytical specification. As can be seen from the figure, the observed errors are much lower than this expected threshold.

How conservative is the specification? We start by identifying a representative input set or input generator that can provide expected inputs, the configuration space (the parameters to tune and the legal range), and the tuning objective for each algorithm (optimize memory usage or runtime). For our example, we used an input generator from AxProf to generate lists of numbers drawn from Zipf distributions with multiple skews as the representative inputs. The configuration space is made out of potential values for w (1–1000) and d (1–10). We then used Opentuner [6] to tune the algorithm to find the best values for w and d that satisfied the error threshold for all the generated inputs, while minimizing memory consumption. To test that the algorithm satisfied the error threshold, we used AxProf's automatically generated statistical testing code [16].

Figure 2 shows errors we observed by setting the parameters of the algorithm based on an empirical model learned using autotuning. The observed error is now closer to the expected threshold but still below it. The algorithm finishes 30% faster and uses 50% less memory compared to setting the parameters according to the analytical specification.

Figure 3 shows the memory consumption model we created using our autotuner for a CountMin implementation [2]. It shows the memory usage (y-axis) by the algorithm implementation to achieve the error guarantees (x-axis). The results show that the memory consumption can be significantly reduced using algorithm parameters with a tuned model compared to using the analytical specification. The benefits are larger for small error thresholds when the conservative analytical bounds result in higher resource consumption. We observed similar benefits for runtime.

3 Challenges and Future Directions

Adaptive Algorithms. When the algorithm accuracy is data dependent, the algorithm parameters can be set based on the input to achieve optimal performance. However, if the input is very large, pre-analyzing the data may not be possible. An interesting direction in run-time adaptation was pioneered by IRA [17], which generates approximations for image processing applications, while also identifying a slack region between the prediction on the smaller input and the full input. This idea was later extended by VideoChef [22] for streaming videos. A challenge for applying this approach to data-streaming applications is finding a similar notion of small "input sequence" or "time window". However, accuracy specifications can guide development of inexpensive off-line models that extrapolate the runtime accuracy predictions.

Runtime Monitoring. An implementation can periodically estimate the error at runtime. If that estimate starts to exceed an acceptable threshold, it can issue a warning or change to a more accurate configuration of the algorithm. Recently, Albarghouthi and Vinitsky [5] presented a monitoring of probabilistic constraints to check fairness specifications. A key challenge for this direction will be balancing the overhead of runtime monitoring and the benefits provided by alternative algorithm selection.

Reducing Offline Training Time. The time required to generate models can be high. For each configuration, we must test multiple inputs, and for each input, we must execute the algorithm multiple times to check conformance to accuracy specifications. Sequential sampling methods can be used to reduce the number of executions while maintaining a sufficient degree of statistical confidence.

Comparing Implementations. Building principled hybrid analytical/empirical models offers an opportunity to more precisely compare different implementations that satisfy the same specification but with different properties (actual accuracy, performance, memory consumption). We found two implementations of CountMin sketch on GitHub—*awn* [2] and *alabid* [1]—that request memory quite differently. Such differences can be discovered by empirical models we described here. Constructing the empirical tradeoff frontiers may reveal that one implementation dominates the other, or more likely, that they operate better for different input sets. In such situations, we envision selecting the implementation based on the statistics of a small initial data set, thus automatically constructing approximate poly-algorithms.

Acknowledgements

The research presented in this paper was funded in part by NSF Grants CCF-1629431 and CCF-1703637.

References

[1] 2018. Count-Min Sketch `github.com/alabid/countminsketch`.

[2] 2018. CountMinSketch `github.com/AWNystrom/CountMinSketch`.

[3] Sameer Agarwal, Henry Milner, Ariel Kleiner, Ameet Talwalkar, Michael Jordan, Samuel Madden, Barzan Mozafari, and Ion Stoica. 2014. Knowing when you're wrong: building fast and reliable approximate query processing systems. In *SIGMOD*.

[4] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*.

[5] Aws Albarghouthi and Samuel Vinitsky. 2019. Fairness-Aware Programming. In *FAT*.

[6] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U. M. O'Reilly, and S. Amarasinghe. 2014. Opentuner: An extensible framework for program autotuning *(PACT)*.

[7] Burton H. Bloom. 1970. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM* 13 (1970).

[8] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the Count-Min sketch and its applications. *Journal of Algorithms* 55 (2005).

[9] Erik D Demaine, Alejandro López-Ortiz, and J Ian Munro. 2002. Frequency estimation of internet packet streams with limited space. In *European Symposium on Algorithms*.

[10] Petros Drineas, Ravi Kannan, and Michael W Mahoney. 2006. Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication. *SIAM J. Comput.* 36 (2006).

[11] Petros Drineas and Michael W Mahoney. [n. d.]. RandNLA: randomized numerical linear algebra. *Commun. ACM* 59, 6 ([n. d.]).

[12] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *AofA: Analysis of Algorithms*.

[13] I. Goiri, R. Bianchini, S. Nagarakatte, and T. Nguyen. 2015. ApproxHadoop: Bringing Approximations to MapReduce Frameworks *(ASPLOS)*.

[14] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* 53 (2011).

[15] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. 2012. Simple and Practical Algorithm for Sparse Fourier Transform. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '12)*.

[16] Sasa Misailovic Keyur Joshi, Vimuth Fernando. 2019. Statistical Algorithmic Profiling for Randomized Approximate Programs. In *ICSE*.

[17] Michael A. Laurenzano, Parker Hill, Mehrzad Samadi, Scott Mahlke, Jason Mars, and Lingjia Tang. 2016. Input Responsiveness: Using Canary Inputs to Dynamically Steer Approximation. In *PLDI*.

[18] Jayadev Misra and David Gries. 1982. Finding repeated elements. *Science of computer programming* 2 (1982).

[19] Hagit Shatkay and Stanley B Zdonik. 1996. Approximate queries and representations for large data sequences. In *ICDE*.

[20] Joel Tropp. 2015. An introduction to matrix concentration inequalities. *Foundations and Trends in Machine Learning* 8 (2015).

[21] Jeffrey S. Vitter. 1985. Random Sampling with a Reservoir. *ACM Trans. Math. Softw.* 11 (1985).

[22] Ran Xu, Jinkyu Koo, Rakesh Kumar, Peter Bai, Subrata Mitra, Sasa Misailovic, and Saurabh Bagchi. 2018. Videochef: efficient approximation for streaming video processing pipelines. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 43–56.

[23] Kai Zeng, Shi Gao, Jiaqi Gu, Barzan Mozafari, and Carlo Zaniolo. 2014. ABS: a system for scalable approximate queries with accuracy guarantees. In *SIGMOD*.